O‘ZBEKISTON RESPUBLIKASI OLIY VA
O‘RTA MAXSUS TA’LIM VAZIRLIGI
O‘RTA MAXSUS, KASB-HUNAR TA’LIMI MARKAZI

Sh.Nazirov, R.Qobulov, F.Nuraliyev

OBYEKTGA MO‘LJALLANGAN DASTURLASH

WEB-SAHIFALARNI DASTURLASH

Axborot-kommunikatsiya texnologiyalari sohasidagi kasb-hunar kollejlarining «Axborot-kommunikatsiya tizimlari (3521916)» mutaxassisligi o‘quvchilari uchun o‘quv qo‘llanma

«SHARQ» NASHRIYOT-MATBAA
AKSIYADORLIK KOMPANIYASI
BOSH TAHRIRIYATI
TOSHKENT — 2007

N18

Mazkur o‘quv qo‘llanma Germaniya texnik hamkorlik tashkiloti (GTZ) hamda Germaniya taraqqiyot banki (KfW) ishtirokidagi «Axborot-kommunikatsiya texnologiyalari sohasida kasb-hunar ta’limini rivojlantirishga ko‘maklashish» loyihasi doirasida ishlab chiqilgan.

O‘zbekiston Respublikasi Oliy va o‘rta maxsus ta’lim vazirligi O‘rta maxsus, kasb-hunar ta’limi markazi tomonidan axborot-kommunikatsiya texnologiyalari sohasidagi kasb-hunar kollejlari uchun tavsiya etilgan.

T a q r i z ch i l a r: *A.XOLDJIGITOV* — O‘zMU professori, fizika-matematika fanlari doktori.

M.E.ZAYNUTDINOVA — Mirzo Ulug‘bek tumani informatika va hisoblash texnikasi kasb-hunar kolleji «Informatika va dasturlash» kafedrasi mudiri, maxsus fan o‘qituvchisi

Nazirov Sh. va boshq.
Obyektga mo‘ljallangan dasdurlash. Web-sahifalarni dasturlash: Kasb-hunar kollejlari uchun o‘quv qollanma/ —Toshkent, «Sharq», — 2007, — 136 bet.

I. Qobulov R.V. II. Nuraliev F.M.

BBK 32.973.202-018.2ya722

ISBN 978-9943-00-184-8

MUNDARIJA

KIRISH

Obyektga mo‘ljallangan yondashuv (OMY) bir kunda o‘ylab topilgan emas. Uning paydo bo‘lishi dasturiy ta’minotning tabiiy rivojidagi navbatdagi pog‘ona xolos. Vaqt o‘tishi bilan qaysi uslublar ishlash uchun qulay-u, qaysinisi noqulay ekanini aniqlash oson bo‘lib bordi. OMY eng muvaffaqiyatli, vaqt sinovidan o‘tgan uslublarni o‘zida samarali mujassam etadi.

Dasturga Obyektlar atamalari bilan ta’rif berish dasturiy ta’minotni ishlab chiqishning eng tushunarli usulidir. Obyektlar hamma narsani Obyekt nima qilayotgani nuqtayi nazardan idrok etishga, ya’ni uning hatti-harakatlarini hayolan modellashtirishga majbur qiladi. Shu tufayli Obyektga yondashishda u dasturning bajarilishi jarayonida qanday ishlatiladi degan nuqtayi nazardan biroz e’tiborni chalg‘itish mumkin. Shunday qilib, dasturni yozish jarayonida haqiqiy dunyoning tabiiy atamalaridan foydalanish mumkin. Dasturni alohida protseduralar va ma’lumotlar shaklida (kompyuter dunyosi atamalarida) qurish o‘rniga Obyektlardan iborat dastur qurish mumkin. Obyektlar otlar, fe’llar va sifatlar yorlamida haqiqiy dunyoni dasturda modellashtirishga imkon beradi.

Ushbu qo‘llanma Obyektga mo‘ljallangan dasturlashga bag‘ishlangandir.

Qo‘llanma uch qismdan iborat bo‘lib, birinchi qismda Obyektga mo‘ljallangan yondashuv tarixi, afzalliklari va maqsadlari; Obyektga mo‘ljallangan yondashuvning uchta tamoyili: Inkapsulyatsiya, Vorislik, Polimorfizm; Obyektga mo‘ljallangan tahlil va loyihalash; Unifikatsiyalangan modellash tili – UML asoslari; UML diagrammalari haqida batafsil ma’lumotlar keltirilgan.

Ikkinchi qismda C++da Obyektga mo‘ljallangan dasturlash asoslari berilgan bo‘lib Sinflar va Obyektlar; Konstruktorlar va destruktorlar; Vorislik tushunchasi; Vorislik turlari; Polimorfizm; Virtual funksiyalar; Oqimlar turlari; C++da fayllar bilan ishlash sinflari; Shablonlar va Istisnolardan foydalanish; Try...Throw...Catch operatorlari ko‘rib chiqilgan.

Uchinchi qism Web-dasturlashga bag‘ishlangan bo‘lib, unda DHTMLga kirish; Scriptlar tili; HTML obyektlari; Form va Image obyekti; Frames kolleksiyasi bayon qilingan.

Ushbu o‘quv qo‘llanma kasb-hunar kollejlari o‘qituvchilari va o‘quvchilari uchun mo‘ljallangan bo‘lib, undan oliy o‘quv yurtlari professor o‘qituvchilari talabalari, shu sohaga qiziquvchi barcha boshqa foydalanilishi mumkin.

1.1. Obyektga mo‘ljallangan yondashuvning afzalliklari va maqsadlari

Obyektga mo‘ljallangan yondashuv (OMY) dasturiy ta’minotni ishlab chiqishda oltita asosiy maqsadni ko‘zlaydi. OMY paradigmasiga muvofiq ishlab chiqilgan dasturiy ta’minot quyidagi xususiyatlarga ega bo‘lmog‘i lozim:

1) tabiiylik;
2) ishonchlilik;
3) qayta qo‘llanish imkoniyati;
4) kuzatib borishda qulaylik;
5) takomillashishga qodirlik;
6) yangi versiyalarni davriy chiqarishning qulayligi.

Tabiiylik

OMY yordamida tabiiy dasturiy ta’minot yaratiladi. Tabiiy dasturlar tushunarliroq bo‘ladi. Dasturlashda «massiv» yoki «xotira sohasi» kabi atamalardan foydalanish o‘rniga yechilayotgan masala mansub bo‘lgan soha atamalaridan foydalanish mumkin. Ishlab chiqilayotgan dasturni kompyuter tiliga moslash o‘rniga OMY aniq bir sohaning atamalaridan foydalanish imkonini beradi.

Ishonchlilik

Yaxshi dasturiy ta’minot boshqa har qanday mahsulotlar, masalan, muzlatgich yoki televizorlar kabi ishonchli bo‘lmog‘i lozim.

Puxta ishlab chiqilgan va tartib bilan yozilgan Obyektga mo‘ljallangan dastur ishonchli bo‘ladi. Obyektlarning modulli tabiati dastur qismlaridan birida, uning boshqa qismlariga tegmagan holda, o‘zgartishlar amalga oshirish imkonini beradi. Obyekt tushunchasi tufayli axborotga ushbu axborot kerak bo‘lgan shaxslar egalik qiladi, mas’uliyat esa berilgan funktsiyalarni bajaruvchilar zimmasiga yuklatiladi.

Qayta qo‘llanish imkoniyati

Quruvchi uy qurishga kirishar ekan, har gal g‘ishtlarning yangi turini ixtiro qilmaydi. Radiomuxandis yangi sxemani yaratishda, har gal rezistorlarning yangi turini o‘ylab topmaydi. Unda nima uchun dasturchi «G‘ildirak ixtiro qilaverishi kerak»? Masala o‘z yechimini topgan ekan, bu yechimdan ko‘p martalab foydalanish lozim.

Malakali ishlab chiqilgan Obyektga mo‘ljallangan sinflarni bemalol takroran ishlatish mumkin. Xuddi modullar kabi Obyektlarni ham turli dasturlarda takroran qo‘llash mumkin. Modulli dasturlashdan farqli o‘laroq, OMY mavjud Obyektlarni kengaytirish uchun vorislikdan, sozlanayotgan kodni yozish uchun esa polimorfizmdan foydalanish imkonini beradi.

Kuzatib borishda qulaylik

Dasturiy mahsulotning ish berish davri uning ishlab chiqilishi bilan tugamaydi. Dasturni ishlatish jarayonida *kuzatib borish* deb nomlanuvchi tirgak kerak. Dasturga sarflangan 60 foizdan 80 foizgacha vaqt kuzatib borishga ketadi. Ishlab chiqish esa ish berish siklining 20 foizinigina tashkil etadi.

Puxta ishlangan Obyektga mo‘ljallangan dastur ishlatishda qulay bo‘ladi. Xatoni bartaraf etish uchun to‘g‘rilashni faqat bitta o‘ringa kiritish kifoya qiladi. Chunki ishlatishdagi o‘zgarishlar tiniq, boshqa barcha Obyektlar takomillashtirish afzalliklaridan avtomatik ravishda foydalana boshlaydi. O‘zining tabiiyligi tufayli dastur matni boshqa ishlab chiquvchilar uchun tushunarli bo‘lmog‘i lozim.

Kengayishga qodirlik

Foydalanuvchilar dasturni kuzatib borish paytida tizimga yangi funksiyalarni qo‘shishni iltimos qiladilar. Obyektlar kutubxonasini tuzishning o‘zida ham ushbu Obyektlarning funksiyalarini kengaytirishga to‘g‘ri keladi.

Dasturiy ta’minot statik (qotib qolgan) emas. Dasturiy ta’minot foydali bo‘lib qolishi uchun uning imkoniyatlarini muttasil kengaytirib borish lozim. OMY da dasturni kengaytirish usullari ko‘p. Vorislik, polimorfizm, qayta aniqlash, vakillik hamda ishlab chiqish jarayonida foydalanish mumkin bo‘lgan ko‘plab boshqa shablonlar shular jumlasidandir.

Zamonaviy dasturiy mahsulotning ish berish davri ko‘p hollarda haftalar bilan o‘lchanadi. OMY tufayli dasturlarni ishlab chiqish davrini qisqartirishga erishildi, chunki dasturlar ancha ishonchli bo‘lib bormoqda, kengayishi osonroq hamda takroran qo‘llanishi mumkin.

Dasturiy ta’minotning tabiiyligi murakkab tizimlarning ishlab chiqilishini osonlashtiradi. Har qanday ishlanma hafsala bilan yondashuvni talab qiladi, shuning uchun tabiiylik dasturiy ta’minotning ishlab chiqish davrlarini qisqartirish imkonini beradi, chunki butun diqqat-e’tiborni yechilayotgan masalaga jalb qildiradi.

Dastur qator Obyektlarga bo‘lingach, har bir alohida dastur qismini boshqalari bilan parallel ravishda ishlab chiqish mumkin bo‘ladi. Bir nechta ishlab chiquvchi sinflarni bir-biridan mustaqil ravishda ishlab chiqish mumkin bo‘ladi. Ishlab chiqishdagi bunday parallellik ishlab chiqish vaqtini qisqartiradi.

1.2. Obyektga mo‘ljallangan yondashuvning uchta tamoyili

Obyektga mo‘ljallangan yondashuv (OMY) ni tushunib etish hamda undan foydalanishni o‘zlashtirib olish uchun, avvalam bor, puxta bazaviy bilimlarni egallab olish lozim. Bazaviy tushunchalarni puxta anglab yetibgina dasturlarni yaratishda OMY ni qo‘llash mumkin. Inkapsulyatsiyalash, vorislik va polimorfizm Obyektga mo‘ljallangan dasturlash (OMD) ning uchta bazaviy tushunchasi hisoblanadi.

Inkapsulyatsiyalash

Inkapsulyatsiyalash dasturni qandaydir monolit, bo‘linmas narsa sifatida olib qaramay, ko‘plab mustaqil elementlarga bo‘lish imkonini beradi. Har bir element o‘z funksiyalarini boshqa elementlardan mustaqil ravishda bajara oladigan alohida modul sifatida olib qaraladi. Aynan inkapsulyatsiyalash tufayli mustaqillik darajasi ortadi, chunki ichki detallar interfeys ortida yashiringan bo‘ladi.

Inkapsulyatsiyalash modullikning Obyektga mo‘ljallangan tavsifidir. Inkapsulyatsiyalash yordamida dasturiy ta’minotni ma’lum funksiyalarni bajaruvchi modullarga bo‘lib tashlash

mumkin. Bu funksiyalarni amalga oshirish detallari esa tashqi olamdan yashirin holda bo‘ladi.

Mohiyatan *inkapsulyatsiyalash* atamasi «germetik berkitilgan; tashqi ta’sirlardan himoyalangan dastur qismi» degan ma’noni bildiradi.

Agar biron-bir dasturiy Obyektga inkapsulyatsiyalash qo‘llangan bo‘lsa, u holda bu Obyekt qora quti sifatida olib qaraladi. Siz qora quti nima qilayotganini uning tashqi interfeysini ko‘rib turganingiz uchungina bilishingiz mumkin. Qora quti biron narsa qilishga majburlash uchun unga xabar yuborish kerak. Qora quti ichida nima sodir bo‘layotgani ahamiyatli emas, qora quti yuborilgan xabarga adekvat (mos ravishda) munosabatda bo‘lishi muhimroqdir.

Interfeys tashqi olam bilan tuzilgan o‘ziga xos bitim bo‘lib, unda tashqi Obyektlar ushbu Obyektga qanday talablar yuborishi mumkinligi ko‘rsatilgan bo‘ladi. Interfeys — obyetni boshqarish pulti.

Ommaviy, xususiy va himoyalangan kirish.

Qandaydir bir elementni ommaviy interfesga kiritish yoki, aksincha, undan chiqarish uchun kalit so‘zdan foydalanish kerak. OMD ning har bir tilida kalit so‘zlar to‘plami belgilangan, biroq bu so‘zlar asosan bir xil funksiyalarni bajaradi.

Obyektga mo‘ljallangan tillarning ko‘pchiligida kirishning uchta darajasi mavjud.

— Ommaviy (public) — barcha Obyektlar uchun kirish uchun ruxsat bor.

— Himoyalangan (protected) — faqat ushbu ekzemplyarga va har qanday tarmoq sinflarga kirishga ruxsat bor.

— Xususiy (private) — faqat ushbu ekzemplyarga kirishga ruxsat bor.

Loyihada kirish darajasini to‘g‘ri tanlab olish g‘oyat muhimdir. Ko‘rinadigan qilinishi lozim bo‘lgan barcha narsa ommaviy bo‘lmog‘i lozim. Berkitilishi lozim bo‘lgan har qanday narsa himoyalangan yoki xususiy kirishga ega bo‘lmog‘i kerak.

Inkapsulyatsiyalashning asosiy afzalliklari

Inkapsulyatsiyalash yordamida mas’uliyatni inson nuqtayi nazaridan tabiiy ko‘ringan usul bilan taqsimlash mumkin. Abstraksiyadan foydalanib, masala yechimini joriy qilish

atamalarida emas, balki ushbu yechilayotgan masala mansub bo‘lgan soha atamalarida ifodalash mumkin. Abstraksiya masaladagi muhim jihatni ajratib ko‘rsatish imkonini beradi.

Kodning muhim uchastkalarini to‘sib va joriy qilinishni berkitib, har bir alohida komponentning to‘g‘riligini tekshirib ko‘rish mumkin. Tekshirilgan komponent qo‘llanganda har bir modulni sinchiklab tekshirish imkoni tug‘iladi, bu esa butun dasturning ishonchli ekaniga shubha qoldirmaydi. Shunday bo‘lsa-da, dastur to‘g‘ri ishlayotganiga amin bo‘lish uchun umumiy tekshiruv zarur.

Takroran qo‘llash imkoniyati: abstraksiya yordamida turli vaziyatlarda qo‘llash uchun yaroqli bo‘lgan oson o‘zgartiriladigan dasturni yaratish mumkin.

Kuzatib borishdagi qulaylik: himoyalangan dasturni kuzatib borish oson. Tobe kodni o‘zgartirmay turib, sinfning joriy qilinishiga har qanday kerakli o‘zgartirishlarni kiritish mumkin. Bu o‘zgartirishlar joriy qilinishdagi o‘zgartirishlarni ham, interfeysga yangi usullarni qo‘shishni ham o‘z ichiga olishi mumkin. Faqat interfeys semantikasi (mazmuni) ning o‘zgartirishlari tobe koddagi o‘zgarishlarni talab qiladi.

Takomillashtirish: dasturni buzmay turib, joriy qilinishni o‘zgartirish mumkin. Boshqacha qilib aytganda, mavjud kodning ishga layoqatliligini saqlagan holda funksional tavsiflarni takomillashtirish mumkin. Buning ustiga, joriy qilish berkitilgan ekan, takomillashtirilgan komponentdan foydalanayotgan kodning ishga tushirilish tavsiflari avtomatik tarzda yaxshilanadi: axir kod, garchi u o‘zgarmagan bo‘lsa-da, takomillashtirilgan komponentlardan foydalanadi-ku! Biroq o‘zgartirishlar kiritilganidan so‘ng yana modulni tekshirish kerak bo‘ladi. Obyektning o‘zgarishi ushbu Obyekt foydalanayotgan butun kodda domino effektini keltirib chiqarishi mumkin.

Yangi versiyalarni davriy chiqarish (nashr etish) qulayligi: dasturni mustaqil modullarga bo‘lib, kodni ishlab chiqish bilan bog‘liq vazifani bir nechta ishlab chiquvchilar o‘rtasida taqsimlash hamda shu yo‘l bilan ishlab chiqish jarayonini tezlashtirishga erishish mumkin.

Komponentlarni ishlab va tekshirib chiqib, ularni yangidan qaytadan o‘zgartirish kerak bo‘lmaydi. Shunday qilib, dasturchi bu komponentlarni takroran qo‘llashi hamda ularni yana «nol»dan boshlab yaratish uchun vaqt sarflamasligi mumkin.

Vorislik

Vorislik mavjud bo‘lgan sinfning ta’rifi asosidayoq yangi sinfni yaratish imkonini beradi. Yangi sinf boshqasi asosida yaratilgach, uning ta’rifi avtomatik tarzda mavjud sinfning barcha xususiyatlari, xulq-atvori va joriy qilinishiga vorislik qiladi. Avval mavjud bo‘lgan sinf interfeysining barcha metodlari va xususiyatlari avtomatik tarzda voris interfeysida paydo bo‘ladi. Vorislik voris sinfida biron-bir jihatdan to‘g‘ri kelmagan xulq-atvorni avvaldan ko‘ra bilish imkonini beradi. Bunday foydali xususiyat dasturiy ta’minotni talablarning o‘zgarishiga moslashtirish imkonini beradi. Agar o‘zgartirishlar kiritishga ehtiyoj tug‘ilsa, bu holda eski sinf funksiyalariga vorislik qiluvchi yangi sinf yozib qo‘ya qolinadi. Keyin o‘zgartirilishi lozim bo‘lgan funksiyalarga qaytadan ta’rif beriladi hamda yangi funksiyalar qo‘shiladi. Bunday o‘rniga o‘rin qo‘yishning mazmuni shundan iboratki, u dastlabki sinf ta’rifini o‘zgartirmay turib, Obyekt ishini o‘zgartirish imkonini beradi. Axir bu holda qayta test sinovlaridan puxta o‘tkazilgan asosiy sinflarga tegmasa ham bo‘ladi-da.

Agar siz ko‘p martalab qo‘llash yoki boshqa biron maqsadlarga ko‘ra vorislikni qo‘llashga ahd qilsangiz, avval har gal qarang: merosxo‘r-sinf bilan vorislikni berayotgan sinfning turlari o‘zaro mos keladimi? Vorislikda turlarning mos kelishi ko‘pincha «Is-a» testi deb ataladi. Ikkita sinf bir xil turga ega bo‘lgandagina, o‘zaro «Is-a» munosabatida turibdi deb hisoblanadi.

Birinchi sinf o‘zida ikkinchi sinfning ekzemplyariga ega bo‘lgandagina ikkita sinf o‘zaro «Has-a» munosabatida turibdi deb hisoblanadi.

Boshqa sinfga merosxo‘r bo‘layotgan sinf meros berayotgan sinf bilan shunday munosabatda bo‘lmog‘i lozimki, bunda natijaviy munosabatlar o‘z ma’nosiga ega bo‘lmog‘i, ya’ni vorislik tabaqalanishiga amal qilinishi kerak.

Vorislik — sinflar o‘rtasida «Is-a» munosabatlarini o‘rnatish imkonini beradigan mexanizm. Merosxo‘r sinf o‘z ajdodi bo‘lgan sinfdan xususiyatlar va xulq-atvorni meros qilib olayotganida, u shuningdek o‘z ajdodi bo‘lgan sinf ehtimol boshqa sinflardan meros qilib olgan xususiyatlar va xulq-atvorga ham ega bo‘ladi.

Vorislik tabaqalanishi qandaydir ma’no kasb etishi uchun ajdodlar ustidan qanday amallar bajarilgan bo‘lsa, avlodlar ustidan

ham shunday amallar bajarilish imkoniyati bo‘lishi lozim. Bu «Is-a» testi yordamida tekshiriladi. Merosxo‘r sinfga funksiyalarni kengaytirish va yangilarini qo‘shish uchun ruxsat beriladi. Ammo unga funksiyalarni chiqarib tashlashga ruxsat yo‘q.

Vorislik yordamida qurilgan sinf metodlar va xususiyatlarning uchta ko‘rinishiga ega bo‘lishi mumkin:

— O‘rniga o‘rin qo‘yish (almashtirish): yangi sinf ajdodlarining metodi yoki xususiyatini shunchaki o‘zlashtirib olmaydi, balki unga yangi ta’rif ham beradi;

— Yangi: yangi sinf butunlay yangi metodlar yoki xususiyatlarni qo‘shadi;

— Rekursiv: yangi sinf o‘z ajdodlari metodlari yoki xususiyatlarini to‘g‘ridan to‘g‘ri olib qo‘ya qoladi.

Obyektga mo‘ljallangan tillarning ko‘pchiligi ta’rifni ma’lumot uzatilgan Obyektdan qidiradilar. Agar u yerdan ta’rif topishning iloji bo‘lmasa, biron ta’rif topilmaguncha qidiruv tabaqalar bo‘yicha yuqoriga ko‘tarilaveradi. Ma’lumotni boshqarish aynan shunday amalga oshiriladi hamda aynan shu tufayli o‘ringa o‘rin qo‘yish jarayoni ish ko‘rsatadi.

Vorislik turlari

Vorislik uch asosiy hollarda qo‘llaniladi:

1) ko‘p martalab foydalanishda;
2) ajralib turish uchun;
3) turlarni almashtirish uchun.

Vorislikning ayrim turlaridan foydalanish boshqalaridan ko‘ra afzalroq hisoblanadi. Vorislik yangi sinfga eski sinfning amalda qo‘llanishidan ko‘p martalab foydalanish imkonini beradi. Kodni qirqib tashlash yoki kiritish o‘rniga vorislik kodga avtomatik tarzda kirishni ta’minlaydi, ya’ni kodga kirishda, u yangi sinfning bir qismidek olib qaraladi. Ko‘p martalab qo‘llash uchun vorislikdan foydalanar ekansiz, siz meros qilib olingan realizatsiya (joriy qilinish) bilan bog‘liq bo‘lasiz. Vorislikning bu turini ehtiyotkorlik bilan qo‘llash lozim. Yaxshisi bu o‘rinda «Has-a» munosabatidan foydalanish kerak.

Farqlash uchun vorislik faqat avlod-sinf va ajdod-sinf o‘rtasidagi farqlarni dasturlash imkonini beradi. Farqlarni dasturlash g‘oyat qudratli vositadir. Kodlash hajmining kichikligi va kodning oson boshqarilishi loyiha ishlanmasini osonlashtiradi. Bu holda kod satrlarini kamroq yozishga to‘g‘ri keladiki, bu qo‘shiladigan xatolar miqdorini ham kamaytiradi.

Almashtirish imkoniyati — OMY da muhim tushunchalardan biri. Merosxo‘r sinfga uning ajdodi bo‘lmish sinfga yuboriladigan xabarlarni yuborish mumkin bo‘lgani uchun ularning har ikkalasiga bir xil munosabatda bo‘lish mumkin. Aynan shuning uchun merosxo‘r sinfni yaratishda xulq-atvorni chiqarib tashlash mumkin emas. Almashtirish imkoniyatini qo‘llab, dasturga har qanday tarmoq turlarni qo‘shish mumkin. Agar dasturda ajdod qo‘llangan bo‘lsa, bu holda u yangi Obyektlardan qanday foydalanishni biladi.

Polimorfizm

Agar Inkapsulyatsiyalash va vorislikni OMY ning foydali vositalari sifatida olib qarash mumkin bo‘lsa, polimorfizm — eng universal va radikal vositadir. Polimorfizm Inkapsulyatsiyalash va vorislik bilan chambarchas bog‘liq bo‘lib, boz ustiga, polimorfizmsiz OMY samarali bo‘lolmaydi. Polimorfizm — OMY paradigmasida markaziy tushunchadir. Polimorfizmni egallamay turib, OMY dan samarali foydalanish mumkin emas.

Polimorfizm shunday holatki, bunda qandaydir bitta narsa ko‘p shakllarga ega bo‘ladi. Dasturlash tilida «ko‘p shakllar» deyilganda, bitta nom avtomatik mexanizm tomonidan tanlab olingan turli kodlarning nomidan ish ko‘rishi tushuniladi. Shunday qilib, polimorfizm yordamida bitta nom turli xulq-atvorni bildirishi mumkin.

Vorislik polimorfizmning ayrim turlaridan foydalanish uchun zarurdir. Aynan o‘rindoshlik imkoniyati mavjud bo‘lgani uchun, polimorfizmdan foydalanish mumkin bo‘ladi. Polimorfizm yordamida tizimga to‘g‘ri kelgan paytda qo‘shimcha funksiyalarni qo‘shish mumkin. Dasturni yozish paytida hatto tahmin qilinmagan funksionallik bilan yangi sinflarni qo‘shish mumkin, buning ustiga bularning hammasini dastlabki dasturni o‘zgartirmay turib ham amalga oshirish mumkin. Yangi talablarga osongina moslasha oladigan dasturiy vosita deganda mana shular tushuniladi.

Polimorfizmning uchta asosiy turi mavjud:

— qo‘shilish polimorfizmi;
— parametrik polimorfizm;
— ortiqcha yuklanish.

Qo‘shilish polimorfizmini ba’zida sof polimorfizm deb ham ataydilar. Qo‘shilish polimorfizmi shuning bilan qiziqarliki, uning tufayli tarmoq sinf nusxalari o‘zini turlicha tutishi

mumkin. Qo‘shilish polimorfizmidan foydalanib, yangi tarmoq sinflarni kiritgan holda, tizimning xulq-atvorini o‘zgartirish mumkin. Uning bosh afzalligi shundaki, dastlabki dasturni o‘zgartirmay turib, yangi xulq-atvorni yaratish mumkin.

Aynan polimorfizm tufayli joriy qilishdan takroran fodalanishni vorislik bilan aynanlashtirish kerak emas. Buning o‘rniga vorislikdan avvalam bor o‘zaro almashinish munosabatlari yordamida polimorf xulq-atvorga erishish uchun foydalanish lozim. Agar o‘zaro almashinish munosbatlari to‘g‘ri belgilansa, buning ortidan albatta takroran qo‘llash chiqib keladi. Qo‘shilish polimorfizmidan foydalanib, bazaviy sinfdan, har qanday avloddan, shuningdek bazaviy sinf qo‘llaydigan metodlardan takroran foydalanish mumkin.

Parametrik polimorfizmdan foydalanib, turdosh metodlar va turdosh (universal) turlar yaratish mumkin. Turdosh metodlar va turlar dalillarning ko‘plab turlari bilan ishlay oladigan dasturni yozish imkonini beradi. Agar qo‘shilish polimorfizmidan foydalanish Obyektni idrok etishga ta’sir ko‘rsatsa, parametrik polimorfizmdan foydalanish qo‘llanayotgan metodlarga ta’sir ko‘rsatadi. Parametrik polimorfizm yordamida, parametr turini bajarilish vaqtigacha e’lon qilmay turib, turdosh metodlar yaratish mumkin. Metodlarning parmetrik parametrlari bo‘lganidek, turlarning o‘zi ham parametrik bo‘lishi mumkin. Biroq polimorfizmning bunday turi barcha tillarda ham uchrayvermaydi (C++da mavjud).

Ortiqcha yuklanish yordamida bitta nom turlicha metodlarni bildirishi mumkin. Bunda metodlar faqat miqdorlari va parametr turlari bilan farqlanadi. Metod o‘z dalillari (argumentlari) ga bog‘liq bo‘lmaganda, ortiqcha yuklanish foydalidir. Metod o‘ziga xos parametrlar turlari bilan cheklanmaydi, balki har xil turdagi parametrlarga nisbatan ham qo‘llanadi. Masalan max metodini ko‘rib chiqaylik. Maksimal — turdosh tushuncha bo‘lib, u ikkita muayyan parametrlarni qabul qilib, ularning qaysi biri kattaroq ekanini ma’lum qiladi. Ta’rif butun sonlar yoki suzuvchi nuqtali sonlar qiyoslanishiga qarab o‘zgarmaydi.

Polimorfizmdan samarali foydalanish sari qo‘yilgan birinchi qadam bu Inkapsulyatsiyalash va vorislikdan samarali foydalanishdir. Inkapsullashsiz dastur osongina sinflarning joriy qilinishiga bog‘liq bo‘lib qolishi mumkin. Agar dastur sinflarning joriy qilinish aspektrlaridan biriga bog‘liq bo‘lib qolsa, tarmoq sinfda bu joriyni to‘g‘rilash mumkin bo‘lmaydi.

Vorislik — qo‘shilish polimorfizmining muhim tarkibiy qismi. Hamma vaqt bazaviy sinfga imkon darajada yaqinlashtirilgan darajada dasturlashga uringan holda, o‘rinbosarlik munosabatlarini o‘rnatishga harakat qilish kerak. Bunday usul dasturda ishlov berilayotgan Obyektlar turlari miqdorini oshiradi.

1.3. Unifikatsiyalangan modellash tili — UMLga kirish

UML (Uniform Modeling Language) vizual modellashtirish tili bo‘lib, tizimlash arxitektorlariga tizim qay darajada standart va tushunish uchun oson shaklga ega ekani haqidagi o‘z tasavvurlarini namoyon etishga imkon beradi. UML loyihaviy yechimlar va ishlab chiquvchilarning o‘zaro aloqalarini qo‘shgan holda birgalikda foydalanishning samarali mexanizmini taqdim etadi.

Atamalar:

— Diagramma — elementlar va ular o‘rtasidagi bog‘lanishlarning grafik tasviri.

— Tizim — qo‘yilgan vazifaning bajarilishini ta’minlaydigan dasturiy va apparat vositalar kombinatsiyasi.

— Tizimni ishlab chiqish deganda uni mijoz uchun, ya’ni biron-bir muammoni hal qilishi lozim bo‘lgan inson uchun yaratish jarayoni tushuniladi.

Tarix

UML ning mualliflari Grady Booch, James Rumbaugh & Ivar Jacobson lardir. 1980-yillar va 1990-yillarning boshlarida ular bir-birlaridan mustaqil ravishda Obyektga mo‘ljallangan tahlil va loyihalashning metodologiyasini o‘ylab topdilar. Keyin 1994—1995-yillarda ular uchalasi Rational Software Corporation da birga bo‘ldilar.

Qolgani — endi tarix. UML ning dastlabki versiyalari dasturiy ta’minot tayyorlash sohasida qo‘llana boshladi, iste’molchilarning bildirgan fikrlari asosida esa ularga jiddiy o‘zgartirishlar kiritilib bordi. Bu UML (DEC, Hewlett-Parkard, Microsoft, Ration, ...) konsorsiumining yuzaga kelishiga zamin bo‘ldi. 1997-yilda konsorsium UMl ning dastlabki versiyasini qabul qildi hamda uni ko‘rib chiqish uchun OMG ga taqdim etdi.

1997-yilning oxirida 1.0 versiyasi chiqdi. Shundan so‘ng OMG guruhi kuzatib borishga kirishdi hamda 1988-yilning oxirida uning yana ikkita yangi versiyasini chiqardi. UML tili dasturiy ta’minot ishlab chiqish sohasida de-fakto standarti bo‘lib qoldi. Hozirgi paytda til faol rivojlanishda davom etmoqda. 2002-yilda joriy hisoblanib kelayotgan 2.0 versiyasi chiqdi.

1.4. UML asoslari

Boshqa har qanday til kabi, UML ham lug‘at va qoidalardan iborat bo‘lib, ular UML tiliga kiritilayotgan so‘zlarni kombinatsiyalab, mazmunli konstruksiyalar olish imkonini beradi. Modellashtirish tilida lug‘at va qoidalar tizimni konseptual va jismoniy jihatdan taqdim etishga yo‘naltirilgan. UML ga o‘xshash modellashtirish tili dasturiy ta’minotning «chizmalari»ni tuzish uchun standart vosita bo‘lib xizmat qiladi.

UML lug‘ati asosiy konstruksiyalarning uchta turini o‘z ichiga oladi:

— *mohiyatlar* — modelning asosiy elementlari bo‘lgan abstraksiyalar;

— *munosabatlar* — mohiyatlar o‘rtasidagi aloqalar;

— ko‘plab mohiyatlar va munosabatlarning manfaatlarini ko‘zlovchi, guruhlashtiruvchi *diagrammalar*.

Endi hamma narsa haqida batafsilroq so‘z yuritamiz.

UML ning mohiyatlari

UML da mohiyatlarning to‘rtta turi mavjud:

— tuzilmaviy;

— hulqiy;

— guruhlashtiruvchi;

— annotatsiyali.

Mohiyatlar tilning Obyektga mo‘ljallangan asosiy elementlaridir. Ularning yordamida to‘g‘ri modellarni yaratish mumkin. Tuzilmaviy mohiyatlar — bu UML tilidagi otlarning modeli. Odatda, bular modelning statik qismlari bo‘lib, tizimning konseptual yoki jismoniy elementlariga mos keladi. Yuqorida aytib o‘tilganidek, tuzilmaviy mohiyatlarning yettita turli ko‘rinishi mavjud bo‘lib, tabiiyki, ularning hammasi UML da o‘z aksini topdi. Tuzilmaviy mohiyatlarning ta’riflarini eslatib o‘tamiz hamda ularga mos keladigan UML grafik obrazining tavsifini keltiramiz.

Sinf (class) — atributlari, operatsiyalari, munosabatlari va ma'nolari umumiy bo'lgan Obyektlar majmuyining tavsifi. Grafik jihatdan sinf to'g'ri to'rtburchak ko'rinishida ifodalanib, unda ushbu sinfning nomi, atributlari va operatsiyalari yozilgan bo'ladi.

Interfeys (interface) — sinf yoki komponent (tarkibiy qism) taqdim etadigan ma'lum xizmatlar (servis, xizmatlar to'plami) ni belgilaydigan operatsiyalar majmuyi. Diagrammalarda interfeys ostida ushbu interfeysning nomi yozilgan doira ko'rinishida tasvirlanadi. Interfeys g'oyat kam hollarda (deyarli hech bir holda desa ham bo'ladi) o'z holicha mavjud bo'lmaydi, odatda u o'zini joriy qiladigan sinf yoki komponentga qo'shilib keladi.

Kooperatsiya (collaboration) o'zaro aloqalarni belgilaydi. U turli rollar va boshqa elementlardan iborat bo'lib, bu rol va elementlar birgalikda ishlab, qandaydir kooperativ effekt sodir qiladi. Grafik tarzda kooperatsiya punktir chiziqlar bilan chegaralangan ellips ko'rinishida tasvirlanadi, ichida faqat uning nomi yozilgan bo'ladi.

Pretsedent (use case) — bu tizim bajarayotgan xatti-harakatlar ketma-ketligining tavsifi bo'lib, ushbu ketma-ketlik ma'lum bir akter (actor) uchun muhim bo'lgan kuzatsa bo'ladigan natija beradi. Grafik jihatdan pretsedent ham uzluksiz chiziq bilan chegaralangan ellips shaklida ifodalanib, uning ichida faqat pretsedent nomi yoziladi.

Faol sinf (active class) deb Obyektlari bir yoki bir necha jarayonlarga, yoki iplar (threads) ga jalb qilingan, shuning evaziga boshqaruvchilik ta'sirini ko'rsata oladigan sinflarga aytiladi. Grafik jihatdan faol sinf oddiy sinf kabi tasvirlanadi, faqat yo'g'on chiziq bilan chegaralangan to'g'ri to'rtburchak shaklida bo'ladi, ichida nomi, atributlari va poeratsiyalar yozilgan bo'ladi.

Komponent (component) yoki tarkibiy qism tizimning jismoniy o'rin almashadigan qismi bo'lib, u ma'lum bir interfeyslar yig'indisiga mos keladi hamda uning joriy qilinishini ta'minlaydi. Grafik jihatdan komponent ichida faqat nomi yozilgan qistirmali to'g'ri to'rtburchak ko'rinishida tasvirlanadi.

Uzel (node) — tizimning dasturiy mahsulot faoliyat ko'rsayotgan vaqtda mavjud bo'lgan elementi. Bu element ma'lum bir xotira hajmiga, shuningdek ishlov berish imkoniyatiga ega bo'lgan hisoblash resursidir. Grafik tasviri kub ko'rinishida bo'lib, ichida uzelning nomi yoziladi, xolos.

Yuqorida sanab o‘tilgan yettita bazaviy element (sinflar, interfeyslar, kooperatsiyalar, pretsedentlar, faol sinflar, komponetlar va uzellar) UML modelida qo‘llanishi mumkin bo‘lgan asosiy tuzilmaviy mohiyatlardir. Mohiyatlarning boshqa turlari ham mavjud: akterlar, signallar, utilitlar (sinflar turlari), jarayonlar va iplar (faol sinf turlari), ilovalar, hujjatlar, fayllar, kutubxonalar, sahifalar va jadvallar (komponentlar turlari).

Endi mohiyatlarning biz avval aytib o‘tmagan boshqa turlarini aniqlaymiz. *Xulqiy mohiyatlar (behavioral trings)* UML modelining dinamik tashkil qiluvchisidir. Bu tildagi fe’llardir. Ular modelning vaqt va fazodagi xulq-atvorini tavsiflaydi. Xulqiy mohiyatlarning faqat ikkita turi mavjud.

O‘zaro aloqa (interaction) — xulq-atvor bo‘lib, uning mohiyati ma’lum maqsadga erishish uchun aniq bir kontekst chegarasida Obyektlar o‘rtasidagi ma’lumotlar (messages) almashinuvidan iborat. O‘zaro aloqa yordamida alohida operatsiyani ham, Obyektlar to‘plamining xulq-atvorini ham tavsiflash mumkin. O‘zaro aloqa xabarlar, xatti-harakatlar ketma-ketligi va aloqalar (Obyektlar o‘rtasida) kabi boshqa elementlarning bo‘lishini ham talab qiladi. Grafik jihatdan xabar strelka ko‘rinishida tasvirlanib, uning tepasida deyarli hamma vaqt tegishli operatsiyaning nomi yozilgan bo‘ladi.

Avtomat (state machine) — holatlar ketma-ketligini belgilovchi xulq-atvor bo‘lib, Obyektlar yoki o‘zaro aloqalar o‘z yashash davri davomida turli voqea-hodisalarga javoban ushbu holatlardan o‘tadi, yoki ularga o‘z munosabatini bildiradi. Avtomatlar yordamida alohida sinf yoki sinflar kooperatsiyasining xulq-atvori tavsiflanadi. Avtomat bilan qator boshqa elementlar ham bog‘liq. Bular: holatlar, bir holatdan boshqasiga o‘tishlar, voqea-hodisalar — xatti-harakatlarning o‘tishini va turlarini nomlovchi mohiyatlar — o‘tishlarga javoblar. Grafik jihatdan holatlar burchaklari sal yumaloqlangan to‘g‘ri to‘rtburchak shaklida tasvirlanadi, ichida holat nomi yoziladi.

O‘zaro aloqalar va avtomatlar UML modeliga kiruvchi asosiy xulqiy mohiyatlardir. Semantik (ma’no) jihatdan ular turli tuzilmaviy elementlar, birinchi navbatda sinflar, kooperatsiyalar va Obyektlar bilan bog‘langan bo‘ladi.

Guruhlanuvchi mohiyatlar UML modelining tashkil qiluvchi qismlaridir. Bular modelni yoyib qo‘yish mumkin bo‘lgan bloklardir. Bunday birlamchi mohiyat bitta nusxada mavjud bo‘ladi. Bu — paket.

Paketlar (packages) — elementlarni guruhlarga tashkil qiluvchi universal mexanizm. Paketga tuzilmaviy, xulqiy va boshqa turdagi guruhlovchi mohiyatlarni solib qo‘yish mumkin. Dastur ishlayotgan paytda haqiqatan mavjud bo‘lgan komponentlardan farqli o‘laroq, paketlar sof kontseptual xarakterga ega, ya’ni faqat ishlab chiqish jarayonida mavjud bo‘ladi. Paketni tasvirlash uchun papkaning qistirmali piktogrammasi qo‘llanadi va unda faqat papka nomi, ba’zida esa uning ichidagilar ko‘rsatilgan bo‘ladi.

Annotatsiyali mohiyatlar — UML modelining izohlovchi qismi. U modelning har qanday elementiga qo‘shimcha tavsif, ta’rif yoki tanbehlar uchun sharhdan iborat. Annotatsiyali elementlarning faqat bitta bazaviy turi mavjud — izoh.

Izoh (note) — sharh yoki cheklashlarni tasvirlash uchun element yoki elementlar guruhiga qo‘shilgan oddiy simvol. Grafik jihatdan ilova chekkasi qayrilgan to‘g‘ri to‘rtburchak ko‘rinishida tasvirlanadi hamda matniy yoki grafik sharhga ega bo‘ladi.

Bu element siz UML modelida qo‘llashingiz mumkin bo‘lgan asosiy annotatsiyali mohiyat hisoblanadi. Ko‘proq o‘rinda ilovalardan diagrammalarni noformal yoki formal matn ko‘rinishida ifodalasa bo‘ladigan sharhlar yoki cheklashlar bilan ta’minlash uchun foydalaniladi. Bu elementning variantlari, masalan, talablar mavjud bo‘lib, ularda tizimning modelga nisbatan tashqi istakdagi xulq-atvori tavsiflanadi.

UML ning o‘zaro munosabatlari

UML tilida munosabatlarning to‘rtta turi belgilangan:

— tobelik;
— assotsiatsiya;
— umumlashtirish;
— joriy qilish.

Bu munosabatlar UML da asosiy bog‘lovchi konstruksiyalar hisoblanadi hamda to‘g‘ri modellarni qurishda qo‘llanadi. Endi aytilganlarni tartib bilan bayon qilsak.

Tobelik (dependency) — ikkita mohiyat o‘rtasidagi semantik (ma’no jihatdan) munosabat bo‘lib, bunda ulardan birining, tobe bo‘lmaganining, o‘zgarishi boshqasining, tobe bo‘lganining, o‘zgarishiga ta’sir ko‘rsatishi mumkin. Grafik jihatdan tobelik punktir chiziq bilan ifodalanadi, bu chiziq odatda belgisi bo‘lgan strelkaga ega bo‘ladi.

Assotsiatsiya (association) — tuzilmaviy munosabat bo‘lib, aloqalar majmuyini tavisflaydi. Bunda aloqa deganda Obyektlar o‘rtasidagi biron-bir ma’noli aloqa tushuniladi. *Agregatlash (aggregation)* assotsiyatsiyaning ko‘rinishlaridan biridir. Butun va uning qismlari o‘rtasidagi tuzilmaviy munosabat shunday ataladi. Grafik jihatdan assotsiatsiya chiziq ko‘rinishida ifodalanadi (ba’zida bu chiziq strelka bilan tugallanadi yoki biron belgiga ega bo‘ladi). Chiziq yonida qo‘shimcha belgilar ham bo‘lishi mumkin, masalan, karralilik va rollarning nomi.

Umumlashtirish (generalization) — bu «ixtisoslashuv/umumlashtirish» munosabati. Bunda ixtisoslashgan element Obyekti (oddiyroq qilib aytganda, avlod) umumlashtirilgan element (ajdod) Obyekti o‘rniga qo‘yilishi mumkin. Obyektli mo‘ljallangan dasturlashda qabul qilinganidek, *avlod (childe)* o‘z *ajdodi (parent)* ning tuzilishi va xulq-atvorini meros qilib oladi. Umumlashtirish munosabati grafik jihatdan bo‘yalmagan strelkali chiziq ko‘rinishida ifodalanadi. Strelka ajdodni ko‘rsatib turadi.

Va, nihoyat, *joriy etish (realization)* — klassifikatorlar o‘rtasidagi semantik munosabat. Bunda bitta klassifikator majburiyatni belgilaydi, ikkinchisi uning bajarilishini kafolatlaydi. Joriy etish munosabatlari ikkita vaziyatda uchraydi: birinchidan, interfeyslar hamda ularni joriy qiluvchi sinflar yoki komponentlar o‘rtasida, ikkinchidan, pretsedentlar hamda ularni joriy qiluvchi kooperatsiyalar o‘rtasida. Joriy qilish munosabatlari bo‘yalmagan strelali punktir chiziq ko‘rinishida ifodalanadi.

Biz UML ning to‘rtta elementining tavsifini berdik. Ular UML modellarida asosiy munosabat turlari bo‘lib xizmat qiladi. Ularning variantlari ham mavjud bo‘lib, ulardan biri, masalan, *aniqlashtirish (refinement), trassalash (trace)*, tobeliklar uchun *ulanish* va *kengaytirish*.

1.5. UML diagrammalari

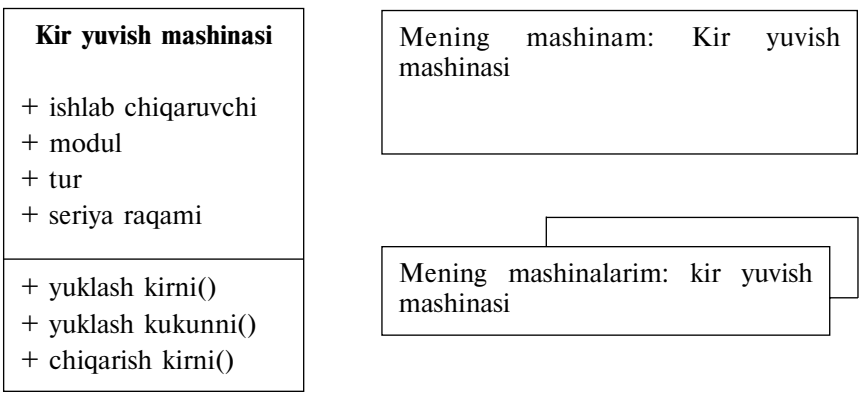
UML tili diagrammalarda qo‘llanadigan grafik elementlar to‘plamini o‘z ichiga oladi. Til bo‘lgani uchun, UML ushbu elementlarni birlashtiruvchi qoidalarga ega.

Diagrammalar tizimning turli tasvirlarini aks ettirish uchun qo‘llanadi. Turli tasvirlarning ushbu to‘plami model deb ataladi. UML tizimi modelini binoning badiiy bezatilgan modeli bilan

qiyoslash mumkin. Shuni ta'kidlab o'tish muhimki, UML modeli tizim nima qilishi lozimligini tavsiflaydi. Shu vaqtning o'zida, bu qanday joriy qilinishi ma'lum qilinmaydi.

Sinflar diagrammasi

Bizni o'rab turgan barcha buyumlar toifalari bo'yicha ajralib turishini sezish qiyin emas (avtomobillar, mebel, kir yuvish mashinasi). Biz bu toufalarga sinflar sifatida murojaat qilamiz. Sinf — bu o'xshash atributlar va umumiy xususiyatlarga ega bo'lgan buyumlar toifasi yoki guruhi.



1.1-rasm. Sinflar diagrammasi.

Sinflar diagrammasi ishlab chiqish jarayonining boshlang'ich nuqtasidir (1.1-rasm).

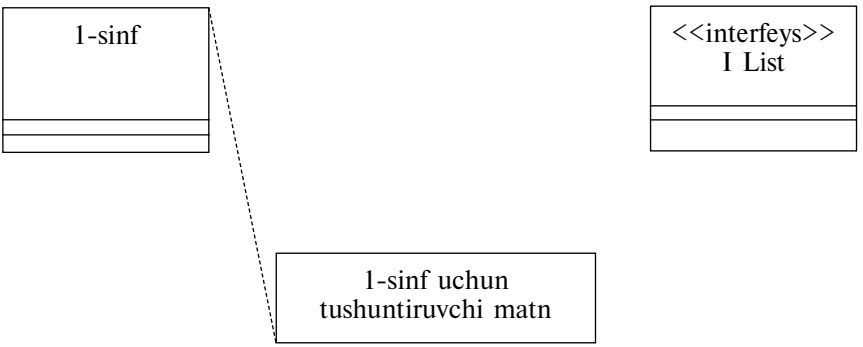
Obyektlar diagrammasi

Obyekt sinf nusxasi bo'lib, u atributlar va operatsiyalarning avvaldan berilgan qiymatlariga ega. Agar kir yuvish mashinasi misolida ko'rib chiqsak, bu holda uning atributlari quyidagi ko'rinishga ega bo'ladi: ishlab chiqaruvchi kompaniya — «Santexnika», model nomi — «Moydodir», servis raqami — «13-666-13» va hajmi — 16 funt.

Tilning kengayishi

Izohlar nimaga xizmat qiladi? Diagrammaning ushbu qismi nima uchun aynan shu yerda joylashgan va u bilan qanday ishlash kerak degan savollarga javob beradi. Stereotiplar UML ning mavjud elementlarini qo'llash va qayta o'zgartirish imkonini

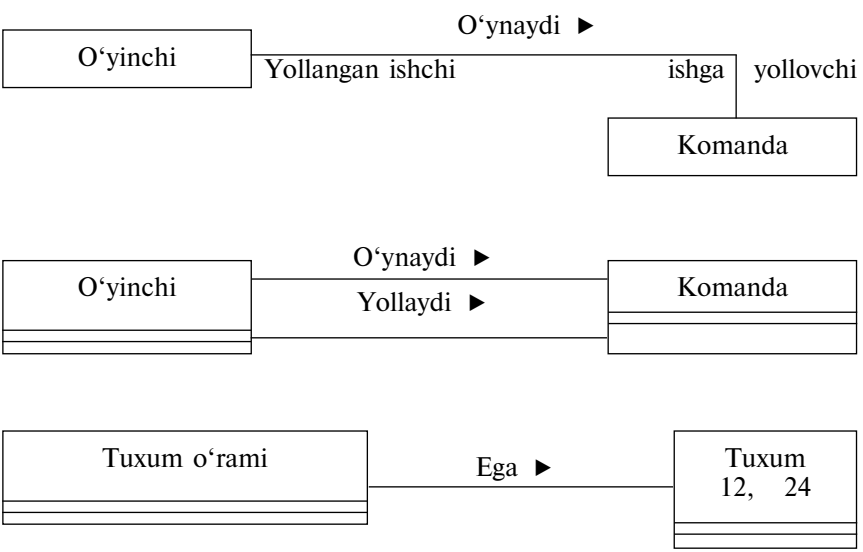
beradi. Interfeys kontsepsiyasi, ya'ni atributlarga ega bo'lmagan sinf bunga yaxshi misol bo'la oladi (1.2-rasm).



1.2-rasm. Izoh va Interfeys.

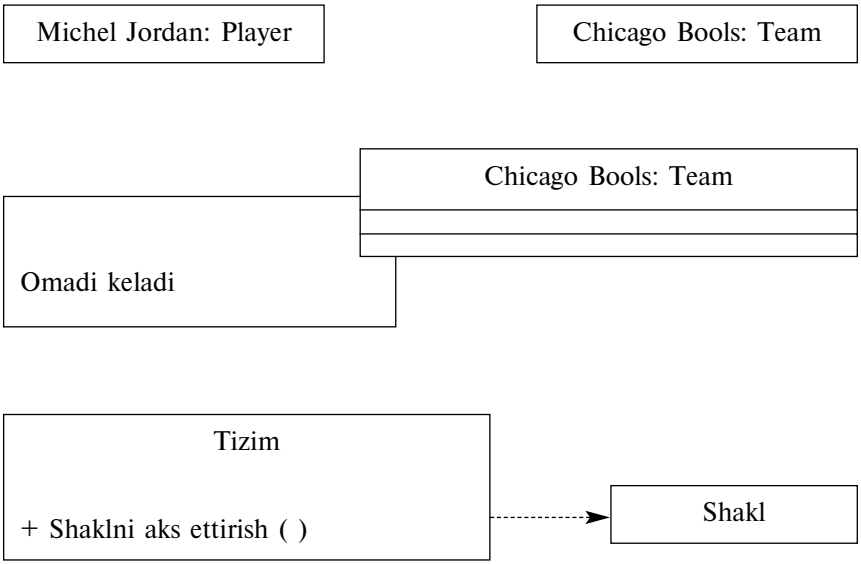
Assotsiatsiyalar

Agar sinflar konseptual jihatdan bir-biri bilan aloqa qilsa, bu aloqa assotsiatsiya deb ataladi. Assotsiatsiyalar cheklagichlarga (masalan{navbat bo'yicha}, {yoki}), kvalifikatorga («bittasi ko'pga» nisbatan qo'shimcha axborot), sinflarga, bo'lishlilikka ega, refleksiv bo'lishi mumkin(1.3-rasm).



1.3-rasm. Assotsiatsiyalar diagrammalari.

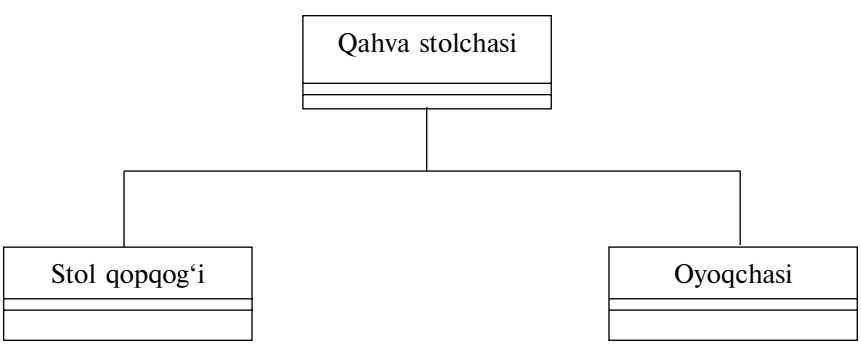
Vorislik, tobelik bo‘lishi mumkin(1.4-rasm).



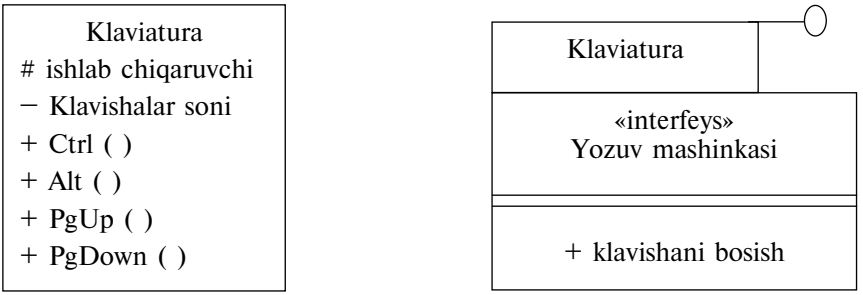
1.4-rasm. Vorislik va tobelik Assotsiatsiyalar diagrammalari.

Agregatlash, kompozit Obyektlar, interfeyslar va joriy qilish

Ba’zida sinf komponent-sinflardan tashkil topadi. Bu o‘zaro aloqaning agregatsiya deb ataluvchi alohida turi. Tanlovni ko‘rsatish maqsadida {yoki} dan foydalanish mumkin. Agregatsiya «qism-butun» munosabatlarini taqozo etadi (1.5-rasm).



1.5-rasm. Agregatsiya diagrammasi.



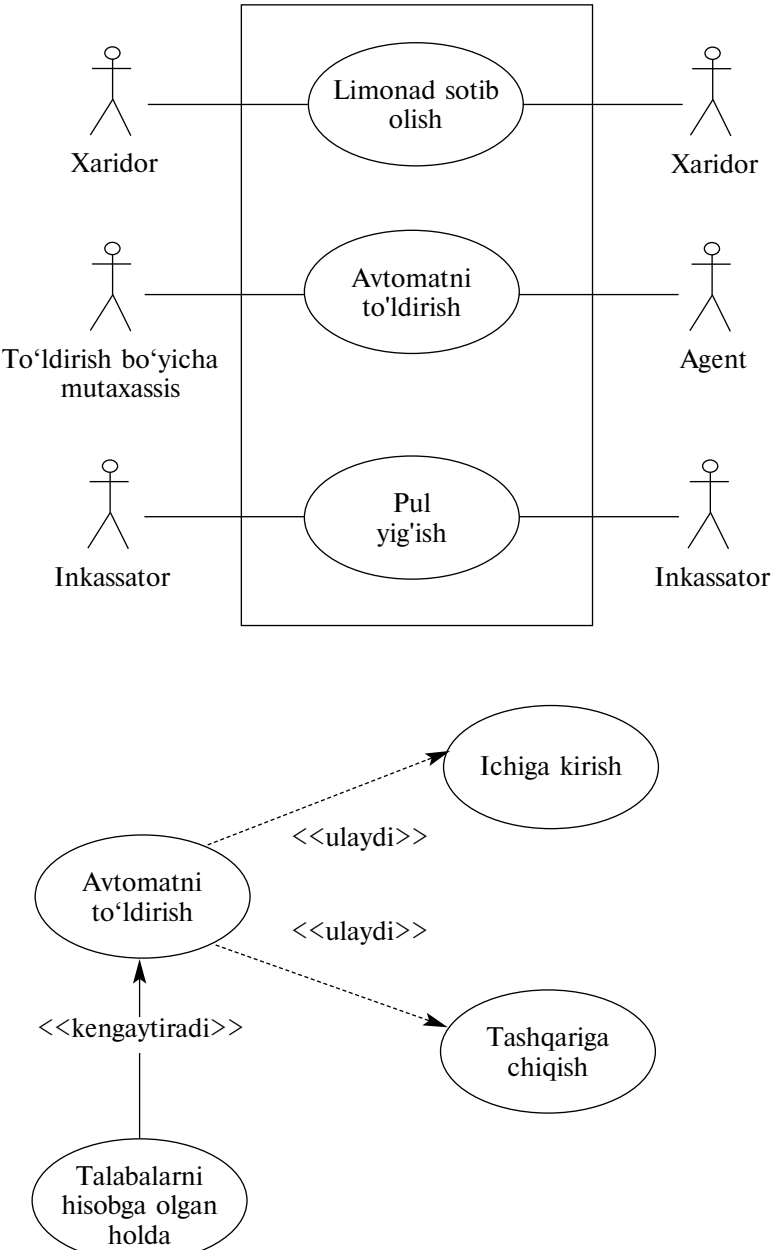
1.6-rasm. Kompozit diagrammasi.

Kompozit agregatsiyaning asl turi bo‘lib, har bir element faqat bitta butunga tegishli ekani bilan tavsiflanadi. Takroran qo‘llanadigan operatsiyalar to‘plamini ajratib olish va bir guruhga birlashtirish mumkin. Interfeysni belgilashning ikkita turi mavjud: doiracha va stereotip. «Ko‘rimlilik» atamasi atributlar va operatsiyalarga nisbatan qo‘llanadi hamda ulardan foydalanishi mumkin bo‘lgan boshqa sinflar turlarini keltirib chiqaradi(1.6-rasm).

- Ochiq soha «+» — hamma foydalanishi mumkin.
- Himoyalangan soha «#» — faqat merosxo‘rlar tomonidan qo‘llanadi.
- Yopiq soha «—» — faqat sinf kirish huquqiga ega.
- Joriy qilish interfeys operatsiyalarining ochiqligini ko‘zda tutadi.
- Statik atributlar va operatsiyalar (sinfga xos, ushbu sinfning barcha Obyektlari uchun yagona) ning tagiga chiziladi.

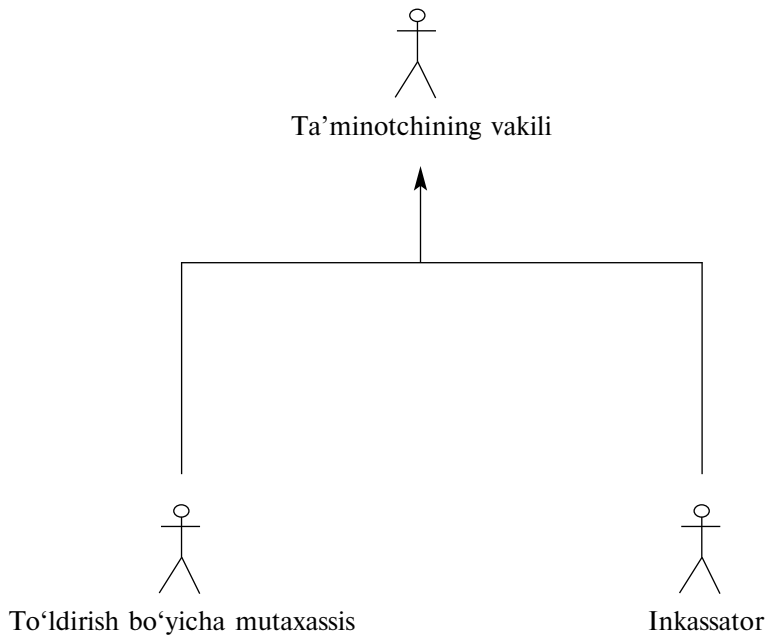
Pretsedentlar diagrammasi (Use case)

Pretsedent — tizimdan foydalanish senariylarining to‘plami. Har bir xatti-harakatlar ketma-ketligi boshqa tizim, foydalanuvchi va h.k. tomonidan vaqtning qandaydir daqiqasida nomlanadi. Mohiyatlar, nomlanuvchi senariylar bajaruvchi deb ataladi. Pretsedentlardan takroran foydalanish mumkin. Bitta usul ulash, ikkinchisi kengaytirish (1.7-rasm).



1.7-rasm. Pretsedentlat diagrammalari.

Pretsedentlarni sinflar kabi umumlashtirish (vorislik qilish) mumkin. Vorislik qilishda sho‘ba pretsedent ajdodining pretsedentiga o‘z qadamlarini qo‘shadi. Bajaruvchilar ham nasldan naslga o‘tishi mumkin. Boshida yuqori darajali pretsedentlar diagrammasini yaratish g‘oyat muhimdir (1.8-rasm).

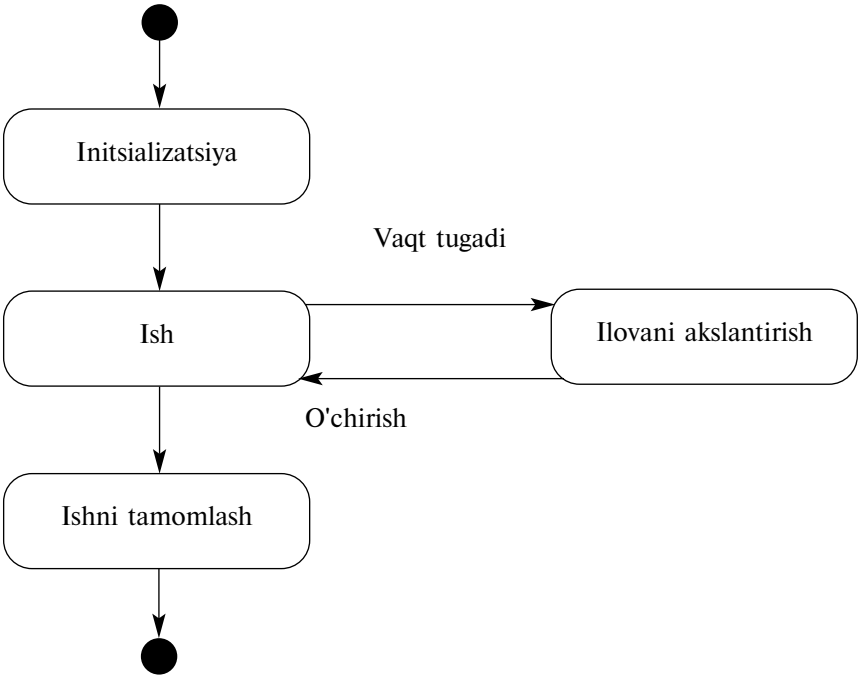


1.8-rasm. Yuqori darajali pretsedentlar diagrammasi.

Holatlar diagrammalari

Sodir bo‘layotgan voqealarga javoban hamda vaqt o‘tishi bilan Obyektlar o‘z holatlarini o‘zgartiradi. Holatlar diagrammasi Obyekt holatlari hamda ular o‘rtasidagi o‘tishlarni, shuningdek Obyektning dastlabki va oxirgi holatini ko‘rsatadi.

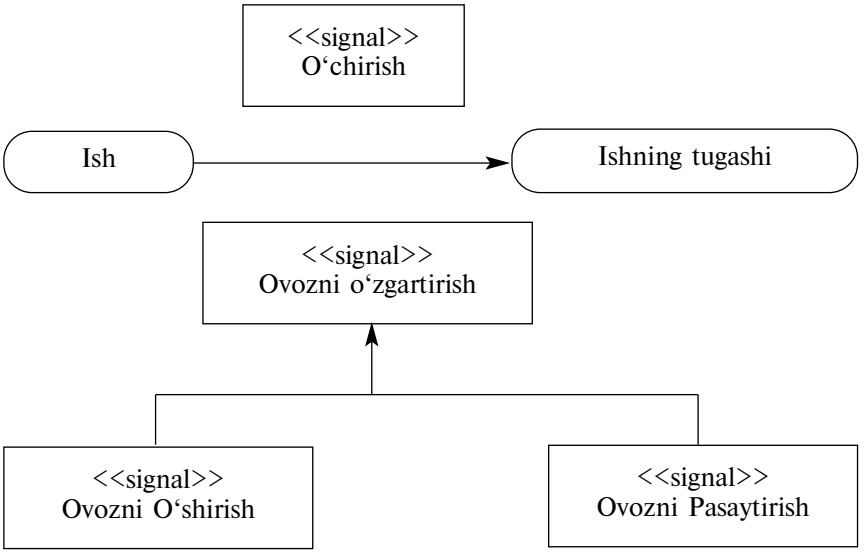
Taymer va schetchiklar kabi holatlar o‘zgaruvchilari ba’zida g‘oyat foydali bo‘lishi mumkin. Faoliyat turlari voqealar va xatti-harakatlardan iborat bo‘ladi. Kirish (tizim holatga kirganda nima sodir bo‘lishi?), chiqish (tizim holatdan chiqqanda nima sodir bo‘lishi?), bajarish (tizim holatda turganda, nima sodir bo‘ladi?) faoliyatning standart turlariga kiradi (1.9.1-rasm).



1.9.1-rasm. Holatlar diagrammasi

O‘tish liniyalariga qo‘shimcha detallarni qo‘shish mumkin. Masalan, o‘tishni keltirib chiqargan voqea ko‘rsatiladi. Holatlar murakkab bo‘lishi mumkin. Ular o‘z ichida tarmoq tizimlar (xatto tarmoq tizimlar guruhi)ga ega bo‘lishi mumkin. Bunday holatlar kompozit holatlar deb ataladi. Bu doiracha ichiga yozilgan «H» (History) harfi bilan belgilanadi. Barcha tobe holatlar xotirada saqlangan chuqur tarix uchun «H*» belgisi qo‘yiladi.

Holatlar diagrammasida qabul qiluvchi Obyektning o‘tishini ta’minlovchi voqea signal deb ataladi. Boshqa har qanday sinflar kabi signallarni ham meros olish mumkin. O‘tish shartli (voqea bo‘yicha) yoki shartsiz (barcha xatti-harakatlarning bajarilishi bo‘yicha) bo‘lishi mumkin (1.9.2-rasm).



1.9.2-rasm. Holatlar diagrammasida signallar.

Holatlar diagrammasida tizimdagi bitta Obyekt holatlari o‘rtasidagi barcha o‘tishlar o‘z aksini topadi. Diagrammalar analitiklar (tahlilchilar) va developerlarga xohishdagi xulq-atvor haqida to‘liq axborot beradi.

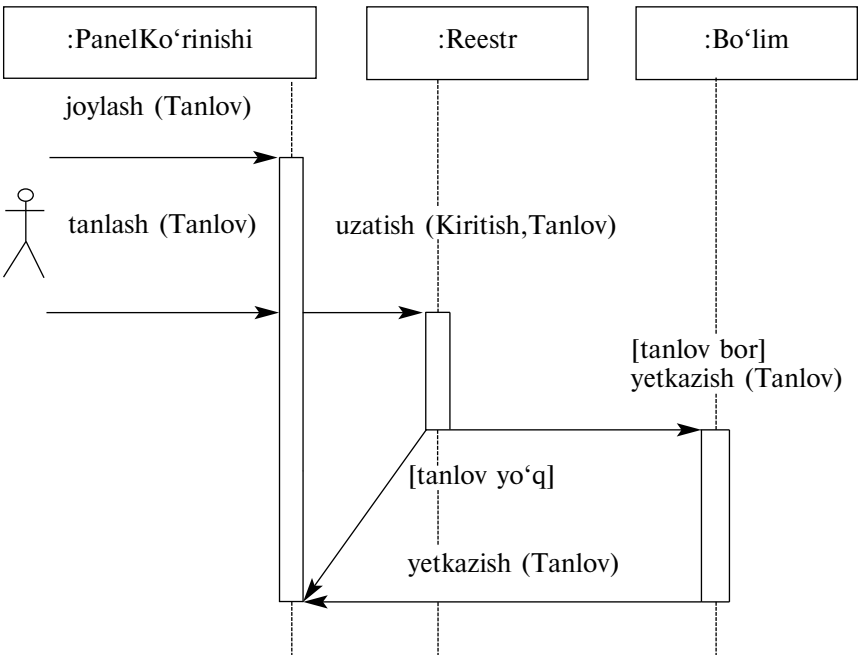
Ketma-ketliklar diagrammalari

Ketma-ketliklar diagrammasi oddiy Obyektlardan, xabarlardan (strelkalar ko‘rinishida), shuningdek vertikal vaqt o‘qidan iborat. Xabar oddiy (boshqaruvni uzatish), sinxron (javob kutadi), nosinxron (javob kutmaydi, amal qilishda davom etadi) bo‘lishi mumkin. Misol: 1. Xaridor tangani avtomatning old panelidagi teshikka joylaydi. 2. Xaridor limonad navini tanlaydi. 3. Tanga reestrga kelib tushadi. 4. Ko‘rib chiqilayotgan asosiy senariy uchun hisoblaymizki, limonadning kerakli navi mavjud hamda reestr limonadni avtomatning old paneliga yetkazib berish uchun komanda beradi(1.10-rasm).

Ketma-ketliklar diagrammasida quyidagi holatlar sodir bo‘lishi mumkin: [shart bo‘yicha o‘tish], *[hozircha sikli], yaratish (yaratmoq(), ‘<<’ yaratmoq ‘>>’), Obyektni bartaraf etish (X).

UML ning ketma-ketliklar diagrammasi Obyektlarning

o‘zaro aloqasiga vaqt o‘zgarishini ham qo‘shadi. Grafikda u uzunchoq tor to‘g‘ri to‘rtburchak ko‘rinishida ifodalanadi va aktivatsiya nuqtasi (operatsiyalardan birining bajarilishi)ni ko‘rsatadi.

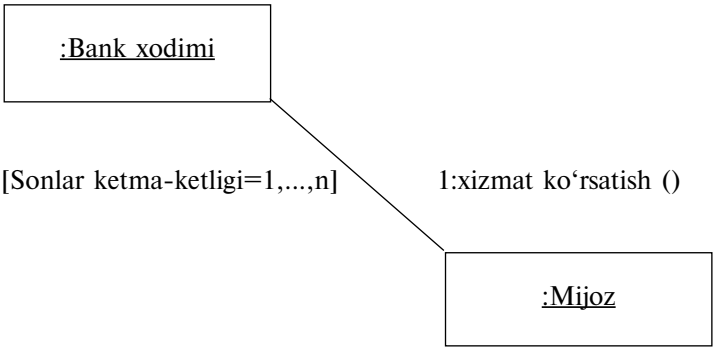


1.10-rasm. Ketma-ketliklar diagrammasi.

Kooperatsiya diagrammalari

Avval ketma-ketliklar diagrammasida tasvirlangan axborotni kooperatsiya diagrammasida ham ifodalash mumkin. Diagrammalarning bu ikkata turi semantik (ma’no) jihatdan ekvivalentdir. Ketma-ketliklar diagrammasi vaqtga muvofiq tartiblashtirilgan bo‘lsa, kooperatsiya diagrammasi Obyektlarning fazodagi joylashuviga muvofiq tartiblashtirilgan.

Kooperatsiya digrammasida Obyektlar o‘rtasidagi assotsiatsiya bio Obyekt ikkinchisiga uzatadigan xabar ko‘rinishida ifodalangan. Xabarning ifodalanishi: strelka, tartib, raqam, nom. Shartlar, avval bo‘ganidek, kvadrat qavslarda aks ettiriladi. sikl (davr)ga tavsif berish uchun, shart oldidan yulduzcha qo‘yish kerak (1.11-rasm).



1.11-rasm. Kooperatsiya diagrammalari.

Ayrim operatsiyalar boshqalariga nisbatan sho‘ba operatsiyalari hisoblanadi. Xabarlarni raqamlash tizimida ularning raqami ajdod nomini bildiruvchi o‘nlik sondan keyin qo‘yilgan nuqtadan keyin yoziladi. Kooperatsiya diagrammasi qabul qiluvchilarning ko‘plab Obyektlarini modellashtirish imkonini beradi. Xabarlar oqimini boshqaruvchi faol Obyektlarni, shuningdek sinxronlashtirilgan xabarlarni ham tasvirlash mumkin.

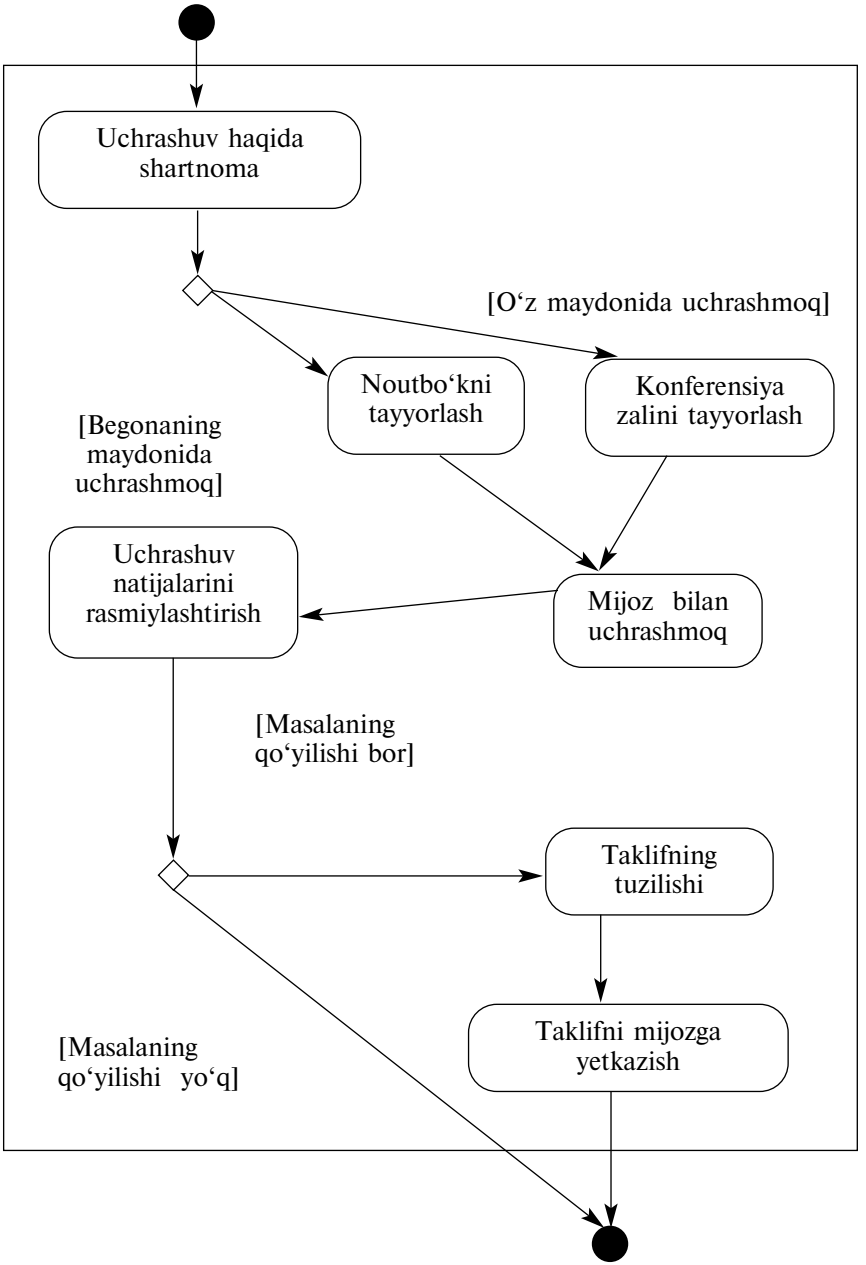
Faoliyat turlarining diagrammalari

Bu diagrammalar blok-sxemalarga g‘oyat o‘xshash. Bunda qarorlarni qabul qilish nuqtalari, barerlar va parallel oqimlar, signallarni uzatish mavjud. Kerakli faoliyat turi qaysi Obyektga tegishli ekanini spetsifikatsiyalash imkoniyati ham bor (1.12-rasm).

Nazorat savollari

Nazariy savollar.

1. Obyektga mo‘ljallangan dasturlash. Obyektga mo‘ljallangan yondashuv. Obyektga mo‘ljallangan yondashuv tarixi. Obyektga mo‘ljallangan yondashuv afzalliklari va maqsadlari
2. Obyektga mo‘ljallangan yondashuvning uchta tamoyili: Inkapsulyatsiya, Vorislik, Polimorfizm. Inkapsulyatsiya tushunchasi va vazifalari. Abstraksiya. Vorislik tushunchasi. Vorislik turlari. Polimorfizm tushunchasi.
3. Obyektga mo‘ljallangan tahlil va loyihalash. Unifikatsiyalangan modellash tili — UML asoslari. UML tili vazifalari va tarkibi.



1.12-rasm. Faoliyat turlarining diagrammalasi.

Obyektlar orasidagi munosabatlar. Tobelik munosabatlari. Assotsiatsiya munosabatlari. Assotsiatsiyalar sinflari. Agregatlash.

4. UML diagrammalari. Sinflar va Obyektlar diagrammasi. Pretsedentlar diagrammasi. Holatlar, Ketma-ketliklar va Kooperatsiya diagrammalari. Faoliyat turlarining diagrammalari. Komponentlar diagrammasi. Yoyib yuborish diagrammalari.

O‘z-o‘zini sinash uchun savollar.

1. Sinf va Obyekt tushunchasiga ta‘rif bering.
2. Inkapsulyatsiya nima uchun kerak?
3. Abstraksiya nima uchun kerak?
4. IS-A va HAS-A munosabatlari farqi nimadan iborat?
5. Qo‘shilish va Parametrik polimorfizm farqi nimadan iborat?
6. Ortiqcha yuklanish afzalliklarini ko‘rsating.
7. Assotsiatsiyalar sinflariga ta‘rif bering.
8. Agregatsiya qanday munosabatlarni taqozo etadi?
9. Pretsedent deb nimaga aytiladi?
10. Holatlar diagrammalari vazifalari va tarkibi nimadan iborat?

2. C++DA OBYEKTGA MO‘LJALLANGAN DASTURLASH

2.1. Sinflarni ishlab chiqish

Sinflar ma’lum maqsadlarga erishish uchun ishlab chiqiladi. Odatda dasturchi mavhum g‘oyadan boshlaydi va asta-sekin loyihani ishlab chiqish jarayonida turli detallar bilan to‘ldirib boriladi. Ba’zida bir-biriga g‘oyat o‘xshash bo‘lgan bir necha sinfni ishlab chiqish bilan ish tugallanadi. Sinflarda kodlarni bu kabi takrorlash (dubllashtirish) dan qochish uchun bu sinflarni ikki qismga bo‘lish kerak, ya‘ni umumiy qismni ota sinfida aniqlab, farqlanadiganlarini hosila sinfda qoldirish kerak.

Sinfdan foydalanishdan oldin u e’lon qilinishi kerak. Odatda, amaliyotchi dasturchi tayyor bazaviy sinflardan foydalanadi, bundan tashqari u barcha spetsifikatsiyalarni va ichki ishlash yo‘llarini bilishi mutlaqo shart emas. Biroq, C++bazaviy sinfidan foydalanishingiz uchun qanday ma’lumotlar a’zolari va metodlarga kira olishingiz mumkinligini (C++Builder komponentasi qo‘llansa, taqdim etilayotgan xususiyatlar va voqealarni ham) albatta bilishingiz lozim.

Bazaviy sinfni e’lon qilish

C++ sizga o‘z xususiyatlari, ma’lumotlari, metodlari va voqealari nomlarini inkapsulalaydigan bazaviy sinfni e‘lon qilish imkonini beradi. O‘zlarining bevosita vazifalarini bajarishdan tashqari, Obyektli metodlar sinf xususiyatlari va ma’lumotlari qiymatlariga kirish uchun ma’lum imtiyozlarga ham ega bo‘ladilar.

Sinf ichidagi har bir e‘lon qilish, sinf nomi qaysi seksiyada paydo bo‘lishiga qarab, bu nomlarga kirish imtiyozlarini aniqlaydi. Har bir seksiya quyidagi kalit-so‘zlarning biridan boshlanadi: **private, protected, public.** Bazaviy sinfni e‘lon qilishning umumlashma sintaksisi quyidagi ko‘rinishga ega:

class className

private:

<privat ma’lumotlar a’zolari> <privat konstruktorlar> <privat metodlar>

protected:

<himoyalangan ma'lumotlar a'zolari> <himoyalangan konstruktorlar> <himoyalangan metodlar>

public:

<ommaviy xususiyatlar> <ommaviy ma'lumotlar a'zolari> <ommaviy konstruktorlar> <ommaviy destruktor> <ommaviy metodlar>

Shunday qilib, C++da bazaviy sinfni e'lon qilish quyidagicha kirish huquqlari va tegishli ko'rishlik sohasini taqdim etadi:

- Privat **private** nomlar faqat ushbu sinf metodlariga ruxsat etilgan eng cheklangan kirish huquqiga ega. Hosila sinflar uchun bazaviy sinflarning privat metodlariga kirish taqiqlangan.
- Himoyalangan **protected** nomlar ushbu sinf va undan hosil bo'lgan sinflar metodlariga ruxsat etilgan kirish huquqiga ega.
- Ommaviy **public** nomlar barcha sinflar va ularning Obyektlari metodlariga ruxsat etilgan cheksiz kirish huquqiga ega.

Quyidagi qoidalar sinf e'lon qilinishining turli seksiyalarining hosil bo'lishida qo'llanadi:

1. Seksiyalar har qanday tartibda ham paydo bo'lishi, ularning nomlari esa takroran uchrayverishi mumkin.

2. Agar sektsiya nomlanmagan bo'lsa, sinf nomlarining keyingi e'lonlarini kompilyator privat deb hisoblaydi. Bu yerda sinf va tuzilmaning e'lon qilinishida farq yuzaga kelayapti: tuzilma yashirin holda ommaviy deb olib qaraladi.

3. Agar siz haqiqatan ham ma'lumotlar a'zolariga har qayerdan kirishni ruxsat etmoqchi bo'lmasangiz, imkon darajasida ularni ommaviy seksiyaga joylashtirmang. Faqat hosila sinflar metodlariga kirish huquqini berish uchun ularni odatda himoyalangan deb e'lon qiladilar.

4. Metodlardan ma'lumotlar xususiyatlari va a'zolarini tanlash, tekshirish va qiymatlarini o'rnatish uchun foydalaning.

5. Konstruktorlar va destruktorlar maxsus funksiyalar bo'lib, qiymatni qaytarmaydilar va o'z sinfining nomiga ega bo'ladilar. Konstruktor berilgan sinf Obyektini quradi, destruktor esa uni olib tashlaydi.

6. C++ning bittadan ortiq yo'riqnomasiga ega bo'lgan metodlarni (konstruktorlar va destruktorlar kabi) sinfdan tashqarida deb e'lon qilish tavsiya etiladi.

Sinf metodlarini sinfdan tashqarida aniqlashga misol:

Funksiya prototipi ichida joylashtirilgan employee sinfini ko‘rib chiqamiz. Funksiyaning o‘zi esa sinfdan tashqarida aniqlangan.

Navbatdagi CLASSFUN.CPP dasturi show_employee funksiyasining ta’rifini sinfdan tashqarida joylashtiradi va bunda sinf nomini ko‘rsatish uchun global ruxsat operatoridan foydalanadi:

```
#include <iostream.h>
#include <string.h>
class employee
{
public:
char name [64];
long employee_id;
float salary;
void show_employee(void);
};
void employee::show_employee(void)
{
cout << "Nom: " << name << endl;
cout << "Nimar: " << employee_id << endl;
cout << "Iklad: " << salary << endl;
};
void main(void)
{
employee worker, boss;
strcpy(worker.name, "John Doe");
worker.employee_id = 12345;
worker.salary = 25000;
strcpy(boss.name, "Happy Jamsa");
boss.employee_id = 101;
boss.salary = 1011012.00;
worker.show_employee();
boss.show_employee();
}
```

Konstruktorlar va destruktorlar

Nomlaridan ko‘rinib turganidek, konstruktor — bu metod bo‘lib, u o‘z xotirasida ushbu sinf Obyektini quradi, destruktor esa — bu Obyektni olib tashlaydigan metod. Konstruktorlar va destruktorlar boshqa Obyektli metodlardan quyidagi xususiyatlariga ko‘ra farqlanadi:

- O‘z sinfi nomi bilan bir xil bo‘lgan nomga ega.
- Qaytariladigan qiymatga ega emas.
- Garchi hosila sinf bazaviy sinflarning konstruktorlari va destruktorlarini chaqira olsa-da, konstruktor va destruktorlarning o‘zlari vorislik qilolmaydi.
- Agar boshqacha e‘lon qilinmagan bo‘lsa, kompilyator tomonidan avtomatik tarzda public sifatida generatsiya qilinadi.
- Sinf Obyektlarining yaratilishi va yo‘q qilinishini tegishli tarzda kafolatlash uchun kompiyaltor tomonidan chaqirib olinadi.
- Agar Obyekt dinamik xotiraning ajratilishi va yo‘q qilinishini talab qilsa, new va delete operatorlariga noaniq murojaatga ega bo‘lishi mumkin.

Quyida konstruktorlar va destruktorlar e‘lonining umumlashma sintaksisini namoyish qiluvchi misol keltiramiz:

```
class className
{public:
//className ma’lmotlarining boshqa a’zolari;
className(); //yashirin konstruktor
className (<parametrlar ro‘yxati>);//Dalillarga ega konstruktor
className(const className&);//Nusha ko‘chirish konstruktori
//Boshqa konstruktorlar
~className();//Destruktor
//Boshqa metodlar};
```

Sinf soni cheklanmagan konstruktorlarga ega bo‘lishi, shu jumladan, konstruktorlarga umuman ega bo‘lmasligi mumkin. Konstruktorlar virtual deb e‘lon qilinishi mumkin emas. Hamma konstruktorlarni himoyalangan sektsiyaga joylashtirmang hamda, yashirin argumentlar qiymatidan foydalanib, ularning sonini kamaytirishga intiling.

Ko‘p hollarda sinf Obyektlarini initsializatsiya qilish (nomlash) ning bir nechta usullariga ega bo‘lish yaxshi natija beradi. Bunga bir nechta konstruktor vositasida erishish mumkin. Masalan:

```
class date {
int month, day, year;
public:
```

```
// ...
date(int, int, int);          //yilning oyi, kuni
date(char*);                  //satr ko‘rinishidagi sana
date(int);                    //bugungi kun, oy, yil
date();                       //yashirin sana:bugungisi
};
```

Ortiqcha yuklangan funksiyalar qanday qoidalarga amal qilsa, konstruktorlar ham parametrlar turlariga nisbatan xuddi shunday qoidalarga amal qiladilar. Agar konstruktorlar o‘z paramterlari turlari bo‘yicha ancha-muncha farq qilsa, kompilyator har gal foydalanganda to‘g‘ri parametrni tanlab olishi mumkin:

```
date today(4);
date 4-july("1983 yil, 4 iyul");
date guy("5-Noy")
date now;              //yashirish initsiallashadi (nomlanadi.)
```

Konstruktorlarning uch turi mavjud.

- *Yashirin konstruktor* parametrlarga ega emas. Agar sinf bitta ham konstruktorga ega bo‘lmasa, kompilyator avtomatik tarzda bitta yashirin konstruktor yaratadiki, u o‘z sinfiga mansub Obyektni yaratishda xotirani shunchaki ajratib beradi.

Date holatida har bir parametr uchun "yashirin qabul qilish: today" (bugun) sifatida talqin qilinadigan yashiringan qiymatni berish mumkin.

```
class date {
int month, day, year;
public:
// ...
date(int d =0, int m =0, int y =0);
date(char*);              // satr vositasida berilgan sana
};
date::date(int d, int m, int y)
{
day = d ? d : today.day;
month = m ? m : today.month;
year = y ? y : today.year;
// yo‘l qo‘yiladigan sana ekanini tekshirish
// ...
}
```

- *Dalillarga ega konstruktor* Obyektning yaratilish paytida uni initsiallash (nomlash) ga, ya‘ni turli funksiyalarni chaqirib olishga, dinamik xotirani ajratish, o‘zgaruvchilarga dastlabki qiymatlarni berish va h.k. ga imkon beradi.
- *Nusha ko‘chirish konstruktori* berilgan sinf Obyektlarini yaratish uchun ma’lumotlarni ushbu sinfning mavjud bo‘lgan boshqa Obyektidan nusxa ko‘chirish yo‘lidan borishga mo‘ljallangan. Bunday konstruktorlar ma’lumotlarning dinamik tuzilmalarini modellashtiradigan Obyektlar nusxasini yaratishda ayniqsa maqsadga muvofiqdir. Biroq yashirin holda kompilyator yuzaki nusxalash konstruktorlari (shallow copy constructors) deb ataluvchi faqat ma’lumotlar a’zolaridan nusxa oladigan konstruktorlarni yaratadi. Shuning uchun, agar biron-bir ma’lumotlar a’zolari ko‘rsatkichlarga ega bo‘lsa, ma’lumotlarning o‘zidan nusxa ko‘chirilmaydi. «Chuqur» nusxalashni konstruktor kodiga joriy qilish uchun tegishli yo‘riqnomalarni kiritish kerak.

```
Misol:
class string
{
char*Str;
intsize;
public:
string(string&);// nusxa ko‘chirish konstruktorlari
}
string::string(string& right)//dinamik o‘zgaruvchilar va resurslar
{
// nusxalarini yaratadi
s = new char[right->size];
strcpy(Str,right->Str);
}
```

Sinf faqat bitta ommaviy destruktorni e‘lon qilishi mumkin. Uning o‘z sinfi bilan bir xil bo‘lgan nomi oldidan ~ (tilda) belgisi turishi kerak. Destruktor parametrlarga ega emas hamda virtual deb e‘lon qilinishi mumkin. Agar sinf destruktor e‘loniga ega bo‘lmasa, kompilyator avtomatik ravishda uni yaratadi.

Odatda tegishli konstruktorlar bajargan operatsiyalarga teskari operatsiyalarni destruktorlar bajaradi. Agar siz fayl sinfi Obyektini yaratgan bo‘lsangiz, bu holda destruktorda bu faylning

yopilishi ehtimoldan xoli emas. Agar sinf konstruktori ma'lumotlar massivi uchun dinamik xotirani ajratsa (new operatori yordamida), bu holda destruktor ajratilgan xotirani bo'shatishi (delete operatori yordamida) ehtimoldan xoli emas va h.k.

date Obyekti konstruktori va destruktoriga misol:

```
class date { int *day, *month, *year
public:
date(int d, int m, int y)
{
day=new int;
month=new int;
year=new int;
*day= d ? d : 1;
*month = m ? m : 1;
*year = y ? y : 1;
}
...
~date()
{
delete day;
delete month;
delete year;
} };
```

this ko'rsatkichi

Obyekt uchun chaqirilgan funksiya (a'zo) da ma'lumotlar (obyekt a'zolari) ga bevosita iqtibos qilish mumkin. Masalan:

```
class x {
int m;
public:
int readm() { return m; }
};
x aa;
x bb;
void f()
{
int a = aa.readm();
int b = bb.readm();
```

```
// ...
}
```

readm() a'zosining birinchi chaqirilishida m aa.m ga tegishli bo'ladi, ikkinchi chaqirilishida esa bb.m ga tegishli bo'ladi.

Obyekt ko'rsatkichi (a'zo funksiyasi mana shu obyekt uchun chaqirilgan) funksiyaning yashirin parametridir. Mana shu yashirin parametrga, xuddi this da bo'lganidek, ochiq iqtibos qilish mumkin. X sinfining har bir funksiyasida this ko'rsatkichi x* this;

sifatida yashirin tavsiflangan hamda shunday nomlangan (initsiatsiya qilingan) ki, obyektni ko'rsatadi (funktsiya-a'zo mana shu obyekt uchun chaqirilgan). X sinfini ekvivalent ravishda quyidagicha tavsiflash mumkin:

```
class x {
int m;
public:
int readm() {return this->m;}
};
```

A'zolarga iqtibos qilganda this dan foydalanish ortiqcha. This asosan funksiya-a'zolarni yozishda qo'llanadi (bu funksiya-a'zolar bevosita ko'rsatkichlar ustida ish olib boradi).

Konstantali funksiya-a'zolar va konstantali obyektlar

Biron-bir funktsiya yoki sinfga sinf obyektining shaxsiy qismiga kirish ruxsati talab qilinsa, ba'zida kirish huquqi qoidalaridan istisno tariqasida chetlanish talab qilinadi. Bu esa sinfning o'zi o'z obyektlariga «chetdan» kirish huquqlarini belgilaydi degan tamoyilga mos keladi. Kirishni nazorat qilish vositalariga funksiya-a'zolarning *konstantali (const)* deb e'lon qilinishi kiradi. Bu holda ular o'zlari birga chaqirilayotgan joriy obyekt qiymatlarini o'zgartirish huquqiga ega emaslar. Shunda bunday funksiyaning sarlavhasi quyidagi ko'rinishga ega bo'ladi:

```
void dat::put() const
{...}
```

Konstanta obyektlarini ham xuddi shunday aniqlash mumkin:

```
const class a{...} value:
```

2.2. Statik elementlar va funksiyalar

Ma'lumotlar elementidan birgalikda foydalanish

Odatda, ma'lum sinf obyektlari yaratilayotganda har bir obyekt o'z-o'zining ma'lumotlar elementlari to'plamini oladi. Biroq shunday hollar ham yuzaga keladiki, unda bir xil sinflar obyektlariga bir yoki bir nechta ma'lumotlar elementlaridan (statik ma'lumotlar elementlaridan) birgalikda foydalanish kerak bo'lib qoladi. Bunday hollarda ma'lumotlar elementlari umumiy yoki juz'iy deb e'lon qilinadi, keyin esa tur oldidan quyida ko'rsatilganidek, static kalit-so'z keladi:

```
private;
static int shared_value;
```

Sinf e'lon qilingach, elementni sinfdan tashqaridagi global o'zgaruvchi sifatida e'lon qilish kerak. Bu quyida shunday ko'rsatilgan:

```
int class_name::shared_value;
```

Navbatdagi SHARE_IT.CPP dasturi book_series sinfini aniqlaydi. Bu sinf (seriya)ning barcha obyektlari (kitoblari) uchun bir xilda bo'lgan page_count elementidan birgalikda foydalanadi. Agar dastur ushbu element qiymatini o'zgartirsa, bu o'zgarish shu ondayoq barcha sinf obyektlarida o'z aksini topadi:

```
#include <iostream.h>
#include <string.h>
class book_series
{
public:
book_series (char *, char *, float);
void show_book(void);
void set_pages(int) ;
private:
static int page_count;
char title[64];
char author[64];
float price;
};
int book_series::page__count;
```

```
void book_series::set_pages(int pages)
{
page_count = pages;
}
book_series::book_series(char *title, char *author, float price)
{
strcpy(book_series::title, title);
strcpy(book_series::author, author);
book_series::price = price;
}
void book_series::show_book (void)
{
cout << «Eslatma: « << title << endl;
cout << «Muallif: « << author << endl;
cout << «Narx: « << price << endl;
cout << «Betlar: « << page_count << endl;
}
void main(void)
{
book_series programming («C++da dasturlashni o‘rganayapmiz», «Jamsa», 213.95);
book_series word(«Windows uchun Word bilan ishlashni o‘rganyapmiz», «Wyatt», 19.95);
word.set_pages(256);
programming.show_book ();
word.show_book() ;
cout << endl << «page_count ning o‘zgarishi» << endl;
programming.set_pages(512);
programming.show_book();
word.show_book();
}
```

Ko‘rinib turganidek, sinf *page_count* ni *static int* sifatida e’lon qiladi. Sinfni aniqlagandan so‘ng, dastur shu vaqtning o‘zida page_count elementini global o‘zgaruvchi sifatida e’lon qiladi. Dastur page_count elementini o‘zgartirganda o‘zgarish shu vaqtning o‘zidayoq book_series sinfining barcha obyekt-larida namoyon bo‘ladi.

Agar obyektlar mavjud bo‘lmasa, public static atributli elementlardan foydalanish

Sinf elementini static kabi e’lon qilishda bu element ushbu sinfning barcha obyektlari tomonidan birgalikda qo‘llanadi. Biroq shunday vaziyatlar yuz berishi mumkinki, dastur hali obyektni yaratganicha yo‘q, ammo u elementdan foydalanish kerak. Elementdan foydalanish uchun dastur uni public va static sifatida e’lon qilishi kerak. Masalan, USE_MBR.CPP dasturi, xatto agar book_series sinfidagi obyektlar mavjud bo‘lmasa ham, bu sinfning page_count elementidan foydalanadi:

```
#include <iostream.h>
#include <string.h>
class book_series
{
public:
static int page_count;
private:
char title [64];
char author[64];
float price;
};
int book_series::page_count;
void main(void)
{
book_series::page_count = 256;
cout << «page_count ning joriy qiymati» <<
book_series::page_count << endl ga teng;
}
```

Bu o‘rinda, sinf page_count sinfi elementini public sifatida e’lon qilgani uchun, xatto agar book_series sinfidagi obyektlar mavjud bo‘lmasa ham dastur sinfning ushbu elementiga murojaat qilishi mumkin.

Statik funksiya-elementlaridan foydalanish

Avvalgi dastur ma’lumotlar *statik* elementlarining qo‘llanishini ko‘rsatib bergan edi. C++ xuddi shunday usul bilan *statik* funksiya-elementlar (metodlar)ni aniqlash imkonini beradi. Agar *statik* metod yaratilayotgan bo‘lsa, dastur bunday metodni, xatto uning obyektlari yaratilmagan holda ham chaqirib olishi mumkin. Masalan, agar sinf sinfdan tashqari

ma'lumotlar uchun qo'llanishi mumkin bo'lgan metodga ega bo'lsa, siz bu usulni statik qila olishingiz mumkin bo'lardi. Quyida displey ekranini tozalash uchun ANSI drayverining esc-ketma-ketligidan foydalanadigan menu sinfi keltirilgan. Agar tizimda ANSI.SYS drayveri o'rnatilgan bo'lsa, ekranni tozalash uchun clear_screen metodidan foydalanish mumkin. Bu metod *statik* deb e'lon qilingani tufayli, xatto agar menu turidagi obyekt mavjud bo'lmasa ham dastur undan foydalanishi mumkin. Keyingi CLR_SCR.CPP dasturi displey ekranini tozalash uchun clear_screen metodidan foydalanadi:

```
#include <iostream.h>
class menu
{
public:
static void clear_screen(void);
// Bu erda boshqa metodlar bo'lishi kerak
private:
int number_of_menu_options;
};
void menu::clear_screen(void)
{
cout << '\033' << «[2J»;
}
void main(void)
{
menu::clear_screen();
}
```

Dastur clear_screen elementini *statik* sifatida e'lon qilar ekan, xatto agar menu turidagi obyekt mavjud bo'lmasa ham dastur bu funksiyadan ekranni tozalash uchun foydalanishi mumkin. clear_screen funksiyasi ANSI Esc[2] esc-ketma-ketligidan ekranni tozalash uchun foydalanishi mumkin.

2.3. Hosilaviy sinflarni e'lon qilish

C++ o'zining barcha ajdodlarining xususiyatlari, ma'lumotlari, metodlari va voqealarini meros qilib oladigan hosila sinfini e'lon qilish imkoniyatini beradi, shuningdek yangi tavsiflarni e'lon qilishi hamda meros sifatida olinayotgan ayrim funksiyalarni ortiqcha yuklashi mumkin. Bazaviy sinfning ko'rsatib o'tilgan tavsiflarini meros qilib olib, yangi tug'ilgan sinfni ushbu

tavsiflarni kengaytirish, toraytirish, o‘zgartirish, yo‘q qilish yoki o‘zgarishsiz qoldirishga majburlash mumkin.

Vorislik bazaviy sinf kodidan hosila sinf nusxalarida takroran foydalanish imkonini beradi. *Takroran qo‘llash* konsepsiyasi jonli tabiatda o‘z paralleliga ega: DNK ni bazaviy material sifatida olib qarash mumkinki, u har bir yaratilgan mavjudotdan o‘zining shaxsiy turini qayta ishlab chiqish uchun takroran foydalanadi.

Hosila sinfni e’lon qilishning umumlashgan sintaksisini ko‘rib chiqamiz. Seksiyalarni sanab o‘tish tartibi himoya imtiyozlarini eng cheklanganlaridan, to eng ommaviylariga qarab kengayib borishiga mos keladi:

class className: [<kirish huquqini beruvchi spetsifikator>] parent Class {
<Do‘stonasinflarni e’lon qilish>
private:
<privat ma’lumotlar a’zolari>
<privat konstruktorlar>
<privat metodlar> **protected:**
<himoyalangan ma’lumotlar a’zolari>
<himoyalangan konstruktorlar>
<himoyalangan metodlar> **public:**
<ommaviy xususitlar>
<ommaviy ma’lumotlar a’zolari>
<ommaviy konstruktorlar>
<ommaviy destruktor>

Sinf o‘zining bazaviy sinfidan yuzaga kelayotganida, uning barcha nomlari hosila sinfda avtomatik tarzda yashirin privat bo‘lib qoladi. Ammo uni, bazaviy sinfning quyidagi kirish spsifikatorlarini ko‘rsatgan holda osongina o‘zgartirish mumkin:

- private. Bazaviy sinfning meros bo‘lib o‘tayotgan (ya’ni himoyalangan va ommaviy) nomlari hosila sinf nusxalarida kirib bo‘lmaydigan bo‘lib qoladi.
- public. Bazaviy sinf va uning ajdodlarining nomlari hosila sinf nusxalarida kirib bo‘ladigan bo‘ladi, barcha himoyalangan nomlar esa himoyalangan bo‘lib qolaveradi.

Bazaviy sinf imkoniyatlarini *kengaytiradigan* sinflarni yuzaga keltirish mumkin: bu yo‘l siz uchun g‘oyat qulay, ammo ozgina shlashni talab qilgan funksiyaga ega. Hosila sinfda kerakli funktsiyani yangidan yaratish vaqtni bekorga sarflash bilan barobar.

Buning o‘rniga bazaviy sinfda koddan takroran foydalanish kerak: bunda u talab qilingan darajada kengaytirilishi mumkin. Hosila sinfda sizni qoniqtirmaydigan bazaviy sinf funksiyasini qayta aniqlang, xolos.

Xuddi shunday yo‘l bilan bazaviy sinf imkoniyatlarini *cheklaydigan* sinflarni yuzaga kelitirish mumkin: Bu yo‘l siz uchun g‘oyat qulay, ammo nimanidir ortiqcha qiladi.

Tugmacha obyekti turlaridan birining yaratilishi misolida tavsiflarni kengaytirish va cheklash metodikalarining qo‘llanishini ko‘rib chiqamiz. Har xil turdagi tugmachalar sizning dasturlaringiz grafik interfeysining dialogli darchalarida tez-tez paydo bo‘lib turadi.

TButtonControl bazaviy sinfi Draw ota usuli yordamida tugmachani biri ikkinchisining ichiga joylashtirilgan ikkita to‘g‘ri to‘rtburchak ko‘rinishida aks ettirish imkoniga ega: tashqi ramka va ichki bo‘yalgan soha.

Ramkasiz oddiy tugmachani yaratish uchun, ota sinf sifatida TButtonControl dan foydalanib, SimpleButton hosila sinfini qurish hamda Draw metodini, uning funktsionalligini cheklagan holda qayta yuklash lozim:

```
class SimpleButton: public : TButtonControl {public:
SimpleButton(int x, int y) ;
void Draw() ;
~SimpleButton() { }
};
SimpleButton::SimpleButton(int x, int y) :
TButtonControl(x, y) { }
void SimpleButton::Draw()
{ outline->Draw();
}
```

SimpleButton uchun konstruktor obyektining yagona vazifasi — ikki parametrli bazaviy sinfni chaqirib olish. Aynan SimpleButton::Draw () metodini qayta yuklash tugmacha tashqi ramkasining ota sinfida sodir bo‘ladigan chiqarilishiga yo‘l qo‘ymaydi. Tabiiyki, metod kodini o‘zgartirish uchun uni TButtonControl bazaviy komponentasining dastlabki matni bo‘yicha o‘rganib chiqish lozim.

Endi izohlovchi nomga ega tugmachani yaratamiz. Buning uchun TButtonControl bazaviy sinfidan TextButton hosila sinfini qurish hamda Draw usulini, uningfunksionalligini oshirgan holda qayta yuklash lozim.

Quyida Text sinfining title nomli obyekti TextButton konstruktori tomonidan yaratilishi, SimpleButton:-.Draw () metodi esa uni aks ettirilishi ko‘rsatilgan:

```
class Text { public:
Text(int x, int y, char* string) { };
void Draw() { }
};
class TextButton: public : TButtonControl {
Text* title;
public:
TextButton(int x, int y, char* title);
void Draw();
-TextButton() { }
};
TextButton::TextButton(int x, int y, char* caption):
TButtonControl(x, y) {
title = new Text(x, y, caption);
};
void TextButton::Draw () {
TextButton::Draw() ;
title->Draw() ;
}
```

Bo‘lim yakunida bazaviy va hosila sinflarini yaratish metodikasini bayon qilish bilan birga, dasturning C++ fragmenti keltiriladi. Bu fragmentda ikkita oddiy geometrik obyekt — doira va silindrning sinflar tabaqalanishi e’lon qilingan.

Dastur shunday tuzilganki, bunda doiraning r — radiusi va silindrning h — balandligi o‘zgaruvchilarining ichki qiymatlari yaratilayotgan obyektlar parametrlarini aniqlashi kerak. Circle bazaviy sinfi doirani modellashtiradi, Cylinder hosila sinfi esa silindrni modellashtiradi.

```
const double pi = 4 * atan(l);
class Circle { protected:
double r ;
public:
Circle (double rVal =0) : r(rVal) {}
void setRadius(double rVal) { r = rVal; }
double getRadiusO { return r; };
.double Area() { return pi*r*r; };
```

```
void showData() ;
};
class Cylinder : public Circle {
protected:
double h;
public:
Cylinder(double hVal = 0, double rVal = 0)
: getHeight(hVal), Circle(rVal) { };
void setHeight(double hVal) { h = hVal; }
double getHeight() { return h; };
double Area() { return 2*Circle::Area()+2*pi*r*h; };
 void showData() ;
void Circle::showData() {
cout «Doira radiusi = «getRadius() endl
«Aylana maydoni =» Area O «endl» endl;
};
void Cylinder::showData()
{
cout << «Asos radiusi = «<< getRadius()<<endl;
cout<< «Silindr balandligi =» << getHeight()<< endl;
cout<<«Yuza maydoni =» <<Area ()<< endl;
void main()
{
Circle circle(2) ;
Cylinder cylinder(10, 1);
circle.showData () ;
cylinder.showData() ;
```

Circle sinfining e'loni r ma'lumotlarining yagona a'zosi, konstruktor va qator metodlardan iborat. Obyektni yaratishda konstruktor r ma'lumotlar a'zosini doira radiusining boshlang'ich qiymati bilan nomlaydi (initsiallashtiradi). Konstruktorning yangi sintaksisini ko'rsatib o'tamiz: chaqirishda u bazaviy konstruktor sinfiga, shuningdek ikki nuqtadan keyin ko'rsatilgan har qanday ma'lumotlar a'zosiga murojaat qilishi mumkin. Bizning misolimizda r ma'lumotlari a'zosi unga rVal parametri bilan murojaat qilish orqali «yaratiladi» va nolli qiymat bilan nomlanadi (inetsiallashtiriladi).

setRadius usuli g ma'lumotlar a'zosi qiymatini belgilaydi, getRadius usuli esa uni qaytaradi. Area metodi doira maydonini qaytaradi. show Data metodi aylana radiusi va doira maydonining qiymatlarini chiqarib beradi.

Circle sinfining hosilasi deb e'lon qilingan Cylinder sinfi h — yagona ma'lumotlar a'zosi, konstruktor va qator metodlardan iborat. Bu sinf r ma'lumotlar a'zosini silindr asosi radiusini saqlash uchun, hamda setRadius va getRadius metodlarini meros qilib oladi. obyektni yaratishda konstruktor r va h ma'lumotlar a'zolarini boshlang'ich qiymatlar bilan nomlaydi. Konstruktorning yangi sintaksisini ko'rsatib o'tamiz: bizning holatda h ma'lumotlar a'zosi hVal argumentining qiymati bilan nomlanadi (initsiallashtiriladi), r ma'lumotlar a'zosi esa rVal argumentiga ega bo'lgan bazaviy sinf konstruktorini chaqirish bilan nomlanadi.

setHeight funksiyasi h ma'lumotlar a'zosi qiymatini belgilaydi, getHeight esa qaytaradi. Circle::Area fuknsiyasi bazaviy sinfdan meros olingan funksiyani, silindr yuzasi maydonini qaytarish uchun, ortiqcha yuklaydi. ShowDate funksiyasi esa silindr asosining radiusi, balandligi va yuzasining maydoni qiymatlarini chiqarib beradi.

main funksiyasi Circle sinfiga mansub 2 radiusli circle aylanasini hamda balandligi 10, asosining radiusi 1 bo'lgan Sylinder sinfiga mansub sylinder silindrni yaratadi, keyin yaratilgan obyektlarning parametrlarini chiqarish uchun showData ga murojaat qiladi.

Aylana radiusi=2, doira maydoni=12.566

Asos radiusi=1, silindr balandligi=10, yuzasining maydoni=69.115.

2.4. Polimorfizm

Polimorfizm yunoncha so'z bo'lib, ikkita o'zakdan - poly (ko'p) va morphos (shakl) dan iborat hamda ko'p shakllilikni bildiradi. Polimorfizm — bu turdosh obyektlar (ya'ni bitta ajdod hosilasi bo'lgan sinflarga mansub obyektlar) ning dastur bajarilish vaqtida vaziyatga qarab. o'zlarini turlicha tuta olish xususiyati. OMD doirasida dasturchi obyekt xulq-atvoriga faqat bilvosita ta'sir ko'rsatishi, ya'ni dasturga kiritilayotgan metodlarni o'zgartirishi hamda avlodlarga o'z ajdodlarida yo'q bo'lgan o'ziga xos xususiyatlarni baxsh etishi mumkin.

Metodni o'zgartirish uchun uni avlodda ortiqcha yuklash kerak, ya'ni avlodda bitta nomdagi metodni e'lon qilish va unda kerakli xatti-harakatlarni ishga solish kerak. Natijada ajdod-obyekt va avlod-obyektda bitta nomdagi ikkita metod amal qila-

di. Bunda ushbu metodlarning kodlari turlicha ishga tushiriladi va, demakki, obyektlarga turlicha xatti-harakat baxsh etadi. Masalan, geometrik shakllar turdosh sinflarining tabaqalanishida (nuqta, to‘g‘ri chiziq, kvadrat, to‘g‘ri to‘rtburchak, doira, ellips va h.k.) har bir sinf Draw metodiga ega bo‘lib, u ushbu shaklni chizib berish talabi qo‘yilgan voqea-hodisaga tegishli javob berilishi uchun mas’uldir.

Polimorfizm tufayli avlodlar bitta voqeaga o‘ziga xos tarzda munosabat bildirish uchun o‘z ajdodlarining umumiy metodlarini ortiqcha yuklashlari mumkin.

Virtual funksiyalar

OMD da polimorfizmga nafaqat yuqorida tavsifi berilgan vorislik va ajdod metodini ortiqcha yuklatish mexanizmi vositasida erishiladi, balki *virtuallash* vositasida ham shunday erishiladiki, u ajdod funksiyalarga o‘z avlodlari funksiyalariga murojaat qilish imkonini beradi. Polimorfizm sinf arxitekturasi orqali ishga tushiriladi, biroq faqat a’zo-funksiyalar polimorf bo‘lishlari mumkin.

C++da polimorf funksiya bitta nomdagi ehtimoliy funksiyalardan biriga faqat bajarilish paytida, ya’ni unga sinfning konkret obyekti uzatilayotgan paytda bog‘lab qo‘yiladi. Boshqacha qilib aytganda, dastlabki dastur matnida funksiyaning chaqirilishi faqat tahminan belgilanadi, aynan qanday funksiya chaqirilayotgani aniq ko‘rsatilmaydi. Bu jarayon *kechikkan bog‘lanish* deb nom olgan. Navbatdagi misol oddiy a’zo-funktsiyalarning polimorf bo‘lmagan xulq-atvori nimaga olib kelishi mumkinligini ko‘rsatadi.

```
class Parent { public:
double F1(double x) { return x*x; };
double F2(double x) { return Fl(x)/2; };
 class Child : public Parent { public:
double F1(double x) { return x*x*x; }
};
void main() {
Child child;
cout << child.F2(3)<<endl;
}
```

Parent sinfi F1 va F2 a’zo-funksiyalarga ega, bunda F1 ni F2 chaqiradi. Parent sinfining hosilasi bo‘lgan Child sinfi F2

funksiyasiga vorislik qiladi, biroq F1 funksiyasini oldindan belgilaydi. Kutilayotgan 13,5 natijasi o‘rniga dastur 4,5 qiymatini chiqarib beradi. Gap shundaki, kompilyator child.F2(3) ifodasini meros qilib olingan Parent::F2 funksiyasi murojaatiga translyatsiya qilib yuboradi, bu funksiya esa o‘z navbatida Parent::F1 ni chaqiradi, Child::F1 ni emas. SHunday bo‘lganda edi, polimorf xulq-atvor qo‘llab-qo‘vvatlangan bo‘lar edi.

C++ kechikkan bog‘lanishni funksiya bajarilish paytida aniqlaydi hamda funksiyalarni *virtuallash* vositasida ularning polimorf xulq-atvorini ta’minlaydi. Bazaviy va hosila sinflarda virtual funksiyalarni e’lon qilish sintaksisini umumlashtiradigan misolni ko‘rib chiqamiz:

```
class className1 {
//Boshqa a’zo-funksiyalar
virtual return Type functionName(<parametrlar ro‘yxati>);
}
class className2 : public className1 {
//Boshqa a’zo-funksiyalar
virtual return Type functionName(<>);
}
```

Parent va Child sinflari obyektlarida F1 funksiyasining polimorf xulq-atvorini ta’minlash uchun uni virtual deb e’lon qilish zarur. Quyida dasturning modifikatsiyalangan matni keltiriladi:

```
class Parent {
public:
virtual double F1(double x) { return x*x; }
double F2(double x) { return Fl(x)/2; }
};
class Child : public Parent { public:
virtual double F1(double x) { return x*x*x; }
);
void main() {
Child child;
cout «child.F2(3)» endl;
}
```

Mana endi dastur kutilayotgan 13,5 natijasini chiqarib bera-

di. Kompilyator shald.F2(3) ifodasini meros qilib olingan Parent::F2 funksyasi murojaatiga translyatsiya qilib yuboradi, bu funksiya esa, o‘z navbatida, Child::F1 avlodining qayta aniqlangan virtual funksiyasini chaqirib oladi.

Agar funksiya bazaviy sinfda virtual deb e’lon qilingan bo‘lsa, uni faqat hosila sinflarda qayta aniqlash mumkin, bunda parametrlar ro‘yxati avvalgidek qolishi zarur. Agar hosila sinfning virtual funksiyasi parametrlar ro‘yxatini o‘zgartirgan bo‘lsa, bu holda uning bazaviy sinfdagi (hamda uning barcha ajdodlaridagi) versiyasi kirib bo‘lmas bo‘lib qoladi. Boshida bunday vaziyat boshi berk ko‘chaga kirib qolgandek ko‘rinishi mumkin, — amalda ortiqcha yuklanish mexanizmini qo‘llab-quvvatlamaydigan OMD tillarida shunday bo‘ladi ham. C++ bu muammoni virtual funksiyalardan emas, balki xuddi shu nomli, faqat boshqa parametr ro‘yxatiga ega bo‘lgan ortiqcha yuklangan funksiyalardan foydalangan holda xal qiladi.

Virtual deb e’lon qilingan funksiya hosila sinflarda **virtual** kalit-so‘z bilan e’lon qilingani yoki qilinmaganidan qat’i nazar, barcha hosila sinflarda virtual hisoblanadi.

Virutal funktsiyalardan berilgan sinf obyektlarining o‘ziga xos xulq-atvorini ishga solish uchun foydalaning. Barcha metodlaringizni virtual deb e’lon qilmang, bu ularni chaqirishda qo‘shimcha hisoblash sarflariga olib keladi. Hamma vaqt destruktorlarni virtual deb e’lon qiling. Bu sinflar tabaqalanishida obyektlarni yo‘q qilishda polimorf xulq-atvorni ta’minlaydi.

Polimorf obyekt-telefonning yaratilishi

Aytaylik, sizning boshliqlaringiz sizga obyekt-telefoningiz diskli, tugmachali yoki to‘lovli telefonlardan birini tanlab olib, emulyatsiya qila olishi kerak dedi. Boshqacha qilib aytganda, obyekt-telefon bitta qo‘ng‘iroq uchun tugmachali apparat sifatida, boshqa qo‘ng‘iroq uchun to‘lovli telefon sifatida va h.k. ishlashi mumkin edi. Ya’ni bir qo‘ng‘iroqdan ikkinchisiga sizning obyekt-telefoningiz o‘z shaklini o‘zgartirishi lozim bo‘ladi.

Turli sinflarga mansub bu telefonlarda faqat bitta farqlanuvchi funksiya mavjud — bu dial metodi. Polimorf obyektni yaratish uchun siz avval bazaviy sinf funksiyalarini, ularning prototiplari oldidan virtual kalit-so‘zni qo‘ygan holda aniqlaysiz. Bu bazaviy sinf funksiyalari hosila sinflar funksiyalaridan shuning bilan farqlanadiki, ular *virtual*dirlar.

Keyin dasturda bazaviy sinf obyektiga ko‘rsatkich tuziladi. Sizning telefonga tuzilayotgan dasturingiz uchun siz *phone* bazaviy sinfiga ko‘rsatkich tuzasiz:

```
phone *poly_phone;
```

Obyekt shaklini o‘zgartirish uchun siz quyida ko‘rsatilganidek, ushbu ko‘rsatkichga hosila sinf obyektining manzilini berib qo‘ya qolasiz:

```
poly_phone=(phone*)&home_phone;
```

Qiymat berish operatoridan keyin keladigan (phone*) belgilari turlarga keltirish operatori bo‘lib, bu operator C++ning kompilyatoriga hamma narsa joyida, bir turdagi o‘zgaruvchi (touch_tone) manzilini boshqa turdagi o‘zgaruvchi (phone)ga berish zarurligini ma’lum qiladi. Dastur poly_phone obyekti ko‘rsatkichiga turli obyektlar manzilini taqdim qilishi mumkin ekan, u holda bu obyekt ham o‘z shaklini o‘zgartirishi, demakki, polimorf bo‘lishi mumkin. Navbatdagi POLYMORPH.CPP dasturi bu metoddan obyekt-telefon yaratish uchun foydalanadi. Dastur ishga tushirilgach, poly_phone obyekti o‘z shaklini diskli telefondan tugmachalisiga, keyin esa to‘lovlisiga o‘zgartiradi:

```
#include <iostream.h>
#include <string.h>
class phone
{
public:
virtual void dial(char *number) { cout << «Raqam to‘plami» << number << endl; }
void answer(void) { cout << «Javobni kutish» << endl; }
void hangup(void) { cout << «Qo‘ng‘iroq bajarildi-trubkani qo‘yish» << endl; }
void ring(void) { cout << «Qo‘ng‘iroq, qo‘ng‘iroq, qo‘ng‘iroq» << endl; }
phone(char *number) { strcpy(phone::number, number); };
protected:
char number[13] ;
};
class touch_tone : phone
{
public:
void dial(char * number) { cout << «Pik Pik Raqam to‘plami « << number << endl; }
touch_tone(char *number) : phone(number) { }
};
```

```
class pay_phone: phone
{
public:
void dial(char *number) { cout << «Iltimos to‘lang» << amount << « sentov» << endl; cout << «Raqam to‘plami» << number << endl; }
pay_phone(char *number, int amount) : phone(number) { pay_phone::amount = amount; }
private:
int amount;
};
void main(void)
{
pay_phone city_phone(«702-555-1212», 25);
touch_tone home_phone(«555-1212»);
phone rotary(«201-555-1212») ;
// Obyekt diskli telefonga aylantirilsin
phone *poly_phone = &rotary;
poly_phone->dial(«818-555-1212»);
// obyekt shakli tugmachali telefonga o‘zgartirilsin
poly_phone = (phone *) &home_phone;
poly_phone->dial(«303-555-1212»);
// obyekt shakli to‘lovli telefonga o‘zgartirilsin
poly_phone = (phone *) &city_phone;
poly_phone->dial(«212-555-1212»);
}
```

Agar ushbu dastur kompilyatsiya qilinib ishga tushirilsa, displey ekranida quyidagi yozuv paydo bo‘ladi:

```
S:\> POLYMORPH <ENTER>
Raqam to‘plami 818-555-1212
Pik Pik Raqam to‘plami 303-555-1212
Iltimos 25 sent to‘lang
Raqam to‘plami 212-555-1212
```

Roly_phone obyekti dastur bajarilishi davomida o‘z shaklini o‘zgartirib turar ekan, u polimorf bo‘ladi.

Do‘stona funksiyalar

Do‘stona funksiyalar, garchi ular biron-bir sinfga mansub bo‘lmasalar-da, biroq tashqi sinf ma’lumotlarining barcha privat

va himoyalangan a'zolariga kirishi huquqiga ega bo'ladilar. Do'stona funksiyalarning e'lon qilinish sintaksisini qaytarilayotgan tur ko'rsatkichi oldidan turgan friend kalit-so'z yordamida ko'rib chiqamiz:

```
class className
{
public:
className(); //Yashirin konstruktor//friend ning boshqa konstruktorlari
returnType friendFunction (<parametrlar ro'yxati>)
};
```

Agar oddiy a'zo-funksiyalar, sinf nusxasiga yashirish parametr — this ko'rsatkichini uzatish hisobiga, o'z sinfining barcha ma'lumotlariga avtomatik tarzda kirish huquqiga ega bo'lsa, do'stona funksiyalar ushbu parmetrning ochiq-oydin spetsifikatsiyasini talab qiladi.

Darhaqiqat, X sinfida e'lon qilingan F do'stona funksiyasi bu sinfga mansub emas, demakki, x.F va xptr>F (bu erda x —X sinfining nusxasi, xptr — uning ko'rsatkichi) operatorlari tomonidan chaqirib olinolmaydi. Bu o'rinda F(&x) yoki F(xptr) murojaatlari sintaktik jihatdan korrekt (to'g'ri) bo'ladi.

Shunday qilib, do'stona funksiyalar sinfning a'zo-funksiyalari vositasida ishga tushirilishi noqulay, qiyin va xatto mumkin bo'lmagan masalalarni ham hal qilishlari mumkin.

2.5. Shablonlar

Shablonlar (parametrlangan turlar) bog'langan funksiyalar yoki sinflar oilasini tuzish imkonini beradi. Shablonni aniqlashning umumiy sintaksisi quyidagi ko'rinishga ega:

temlate <shablonli turlar ro'yxati>{e'lon qilish};

Funksiyalar shablonlari va sinflar shablonlari farqlanadi.

Funksiya shabloni ortiqcha yuklanayotgan funksiyalarni aniqlash namunasini beradi. Tartiblashtirishga yo'l qo'yadigan har qanday turdagi ikkita argumentdan kattarog'ini qaytaradigan Tax(x, y) funksiyasi shablonini ko'rib chiqamiz:

template <class T>T max(Tx, Ty){**return**(x>y)? x:y;}:

Bunda <class T> shablonining argumenti tomonidan taqdim etilgan ma'lumotlar turi har qanday bo'lishi mumkin. Undan dasturda foydalanishda kompilyator tax funksiyasi kodini bu

funksiyaga uzatilayotgan parametrlarning faktik turiga muvofiq generatsiya qiladi:

```
int i;
Myclass a,b;
int i=max(i, 0);//argumentlar turi int
myclass m=max(a, b);// argumentlar turi myclass
```

Faktik turlar kompilyatsiya paytida ma'lum bo'lishlari kerak. Shablonlarsiz max funksiyasini ko'p martalab ortiqcha yuklashga to'g'ri kelar edi, ya'ni, garchi barcha funksiya versiyalarining kodlari bir xil bo'lsa ham har bir qo'llanayotgan tur uchun alohida ortiqcha yuklash kerak bo'lar edi. C++ standarti bu maqsad uchun **#define max(x, y) ((x>y)? x:y)** markosidan foydalanishni qat'iyan tavsiya etmaydi, chunki C++ tiliga oddiy S tili ustidan shunday afzalliklarni beradigan turlarni tekshirish mexanizmi blokirovka qilingan bo'lishi mumkin. Shu narsa ayonki, max(x, u) funksiyasining vazifasi — bir-biriga mos turlarni qiyoslash. Afsuski, makrosni qo'llash bir-biriga mos kelmaydigan turlarning (masalan, int va struct) qiyoslanishiga yo'l qo'yadi.

```
//ikkita o'zgaruvchining o'rnini o'zgartiradigan funksiya shabloni:
template<class T>//T — parametrlanayotgan tur nomi
void chage(T*x, T*y)
{Tz=*x; *x=*y;*y=z;}
Bu funksiya quyidagicha chaqirilishi mumkin:
long k=10,I=5;
chang(&k, &I);
Kompilyator:
void change(long*x, long*y){long z=*x; *x=*y; *y=z;}
```

ta'rifini ifodalab beradi.

Funksiyalar shablonlari parametrlarining asosiy xususiyatlari:

1. Parametrlar nomlari shablonning butun ta'rifi bo'ylab unikal bo'lmog'i lozim.

2. Shablon parametrlarining ro'yxati bo'sh bo'la olmaydi.

3. Shablon parametrlari ro'yxatida har biri class so'zidan boshlanadigan bir nechta parametr bo'lishi mumkin.

Sinflar shabloni sinflar oilasini aniqlash namunasini beradi. Quyida bir o'lchamli ma'lumotlar massivi sinflarining generatori bo'lgan Vector shabloniga misol keltirilgan:

template <class T>class Vector

Ushbu sinfning turlashtirilgan elementlari ustida elementlarning konkret turidan qat'i nazar, bir xil bazaviy operatsiyalar (kiritilsin, o'chirilsin, indekslansin va h.k.) bajariladi. Agar turga parametrdek muomala qilinsa, bu holda kompilyator berilgan tur elementlariga ega bo'lgan vektorlar sinflarini generatsiya qiladi.

Funktsiyalar shablonlarida bo'lganidek,

class Vector<char*.(...}; sinflari shablonlarini ochiq-oydin qayta aniqlashga ruxsat etiladi, bunda Vector belgisi hamma vaqt burchakli qavslar ichiga olingan ma'lumotlar turi bilan birgalikda kelishi kerak.

Navbatdagi dastur ikkita sinfni yaratishda arrey sinf shablonidan foydalanadi. Bu sinflarning biri int turining qiymatlari bilan, ikkinchisi float turining qiymatlari bilan ish ko'radi.

```
#include <iostream.h>
#include <stdlib.h>
template<class T, class T1> class array
{
public:
array (int size);
T1 sum(void);
T average_value(void);
void show_array(void);
int add_value(T);
private:
T *data;
int size;
int index;
};
template<class T, class T1> array<T, t1>::array(int size)
{
data = new T[size];
if (data == NULL)
{
cerr << «Xotira etarli emas − dastur tugayapti» << endl;
exit(l);
}
array::size = size;
```

```
array::index = 0;
}
template<class T, class T1> Tl array<T, Tl>::sum(void)
{
T1 sum = 0;
for (int i = 0; i < index; i++) sum += data[i];
return(sum);
}
template<class T, class T1> T array<T,
T1>::average_value(void)
{
Tl sum =0;
for (int i = 0; i < index; i++) sum += data[i];
return (sum / index);
}
template<class T, class T1> void array<T,
T1>::show_array(void)
{
for (int i = 0; i < index; i++) cout << data[i] << ' ';
cout << endl;
}
template<class T, class T1> int array<T, T1>::add_value(T
value)
{
if (index == size)
return(-1); // Massiv to'la
else
{
data[index] = value;
index++;
return(0); // Omadli
}
}
void main(void)
{
// 100 ta elementdan iborat massiv
array<int, long> numbers(100);
// 200 ta elementdan iborat massiv
array<float, float> values(200);
int i;
for (i = 0; i < 50; i++) numbers.add_value(i);
```

```
numbers.show_array();
cout << «Sonlar summasi teng «<< numbers.sum () << endl;
cout << «O‘rtacha qiymat teng «<< numbers.average_value() << endl;
for (i = 0; i < 100; i++) values.add_value(i * 100);
values.show_array();
cout << « Sonlar summasi teng.» << values.sum() << endl;
cout << « O‘rtacha qiymat teng « << values.average_value() << endl;
}
```

2.6. Fayllar bilan ishlash

Oldindan belgilangan obyektlar va oqimlar

C++da kiritish-chiqarish oqimlarining sinflari mavjud bo‘lib, ular kiritish-chiqarish standart kutubxonasi (stdio.h)ning obyektga mo‘ljallangan ekvivalenti (stream.h)dir. Ular quyidagicha:

Ios	bazaviy oqimli sinf
streambuf	oqimlarning buferlanishi
istream	kiritish oqimlari
ostream	chiqarish oqimlari
iostream	ikkiga yo‘naltirilgan oqimlar
iostream_withassign	noaniq o‘zlashtirish operatsiyali oqim
istrstream	starli kiritish oqimlari
ostrstream	starli chiqarish oqimlari
strstream	ikkiga yo‘naltirilgan satrli oqimlar
ifstream	faylli kiritish oqimlari
ofstream	faylli chiqarish oqimlari
fstream	ikkiga yo‘naltirilgan faylli oqimlar

Standart oqimlar (istream, ostream, iostream) terminal bilan ishlash uchun xizmat qiladi.

Satrli oqimlar (istrstream, ostrstream, strstream) xotirada joylashtirilgan satrli buferlardan kiritish-chiqarish uchun xizmat qiladi.

Faylli oqimlar (ifstream, ofstream, fstream) fayllar bilan ishlash uchun xizmat qiladi.

Quyidagi obyekt-oqimlar dasturda main funksiyasini chaqirish oldidan avvaldan aniqlangan va ochilgan bo‘ladi:

extern istream cin; //Klaviaturadan kiritish standart oqimi

extern ostream cout; //Ekranga chiqarish standart oqimi

extern ostream cerr; //Xatolar haqidagi xabarlarni chiqarish standart oqimi (ekran)

extern ostream cerr: //Xatolar haqidagi xabarlarni chiqarishning buferlashtirilgan standart oqimi.

C++da fayllar bilan ishlash sinflari

C++da fayllar bilan ishlash fstream kutubxonasidagi biron-bir sinflar yordamida amalga oshiriladi.

fstream kutubxonasi fayllarni o‘qib olish uchun javob beradigan ifstream sinfiga hamda faylga axbortning yozib olinishiga javob beradigan ofstream sinfiga ega.

Biron-bir faylni yozish yoki o‘qish uchun, ochish uchun ofstream turdagi yoki mos holda ifstream turdagi o‘zgaruvchini yaratish kerak. Bunday o‘zgaruvchini initsiallashda fayl nomi o‘zgaruvchi nomidan keyin qavs ichida berilgan belgilar massivi ko‘rinishida uzatiladi.

Masalan, S diskida joylashgan ‘text.txt’ faylini ochish kerak. Buning uchun kodning quyidagi fragmenti qo‘llanadi:

```
ifstream ifI (''C:\text.txt'');        //o‘qish uchun
ofstream ofI(''C:\text.txt''):         //yozish uchun
```

Bu yerda ifI va ofI — o‘zgaruvchilar nomi bo‘lib, ular orqali fayl bilan ma’lumotlarni ayirboshlash amalga oshiriladi. Agar fayl ham dasturning bajarilayotgan fayli joylashtirilgan papkada bo‘lsa, u holda faylning nomi to‘liq ko‘rsatilmasligi mumkin (faqat fayl nomi, unga borish yo‘lisiz). Bundan tashqari fayl nomini to‘g‘ridan to‘g‘ri ko‘rsatish o‘rniga uning nomidan iborat belgilar massivlarini ko‘rsatish mumkin.

Masalan:

```
char s[20]=''C:\text.txt'';
ifstream ifI (s);
```

Axborotni faylga yozish uchun put komandasidan foydalanish mumkin. Bu komanda orqali standart turdagi yakka o‘zgaruvchi yoki biron-bir belgilar massivi uzatiladi. Belgilar

massivi uzatilgan holda ham massivdagi belgilar sonini uzatish kerak.

```
ofstream ofI («C:\text.txt»);
char s[9]=«The text»;
ofI.put(s,9);
int i=100;
ofI.put(i);
```

put funksiyasini chaqirish o‘rniga «<<» operatoridan foydalanish mumkin.

```
ofstream ofI («C:\text.txt»);
ofI<<«The text»;
int i=100;
ofI<<i;
```

Bu operatordan kodning bitta satrida turli turdagi qiymatlarni uzatgan holda ko‘p martalab foydalanish mumkin.

```
ofstream ofI («C:\text.txt»);
char s[9]=«The text»;
int i=100;
ofI<<«The text»<<i<<s<<200;
```

Satr haqida gap ketganda, chiqarish satr oxiri belgisi, ya’ni ‘\n’ paydo bo‘lishidan oldin amalga oshiriladi. Belgisiz turga ega bo‘lgan barcha o‘zgaruvchilar oldin belgilarga o‘zgartirib olinadi.

Axborotni fayldan o‘qib olish uchun ‘’>>’’ operatoriga ekvivalent bo‘lgan get funksiyasi qo‘llanadi. put funksiyasi kabi, get funksiyasi ham har qanday o‘zgaruvchilarning standart turlari yoki/va belgilar massivlari bilan ishlay oladi. Shuningdek get ga har jihatdan ekvivalent bo‘lgan getline funksiyasi mavjud: farqi faqat shundaki, getline funksiyasi satr oxiridagi oxirgi belgini qaytarmaydi.

Fayl oxirini aniqlash

Fayl ichidagisini, fayl oxiri uchramaguncha o‘qish dasturidagi oddiy fayl operatsiyasi hisoblanadi. Fayl oxirini aniqlash uchun dasturlar oqim obyektining eof funksiyasidan foydalanishlari mumkin. Agar fayl oxiri hali uchramagan bo‘lsa, bu

funksiya 0 qiymatini qaytarib beradi, agar fayl oxiri uchrasa, 1 qimatini qaytaradi. While siklidan foydalanib, dasturlar fayl oxirini topmagunlaricha quyida ko‘rsatilganidek, uning ichidagilarini uzluksiz o‘qishlari mumkin:

```
while (! Input_file.eof())
{
//Operatorlar
}
```

Ushbu holda dastur, eof funksiyasi yolg‘on (0) ni qaytarguncha, siklni bajarishda davom etadi. Navbatdagi TEST_EOF.CPP dasturi BOOKINFO.DAT fayli ichidagisini, fayl oxiriga etmaguncha, o‘qish uchun eof funksiyasidan foydalanadi.

```
#include <iostream.h>
#include <fstream.h>
void main (void)
{
ifstream input_file(«BOOKINFO.DAT»);
char line[64];
while (! input_file.eof())
{
input_file.getline(line, sizeof(line));
cout << line << endl;
}
}
```

Xuddi shunday, keyingi dastur — WORD_EOF.CPP fayl ichidagisini bitta so‘z bo‘yicha bir martada fayl oxiri uchramaguncha, o‘qiydi:

```
#include <iostream.h>
#include <fstream.h>
void main(void)
{
ifstream input_file(«BOOKINFO.DAT»);
char word[64] ;
while (! input_file.eof())
{
input_file >> word;
cout << word << endl;
}
}
```

Va, nihoyat, keyingi dastur — CHAR_EOF.CPP fayl ichidagisini bitta belgi bo‘yicha bir martada get funktsiyasidan foydalanib, fayl oxiri uchramaguncha o‘qiydi:

```
#include <iostream.h>
#include <fstream.h>
void main(void)
{
ifstream input_file(«BOOKINFO.DAT»);
char letter;
while (! input_file.eof())
{
letter = input_file.get();
cout << letter;
}
}
```

Fayl operatsiyalarini bajarishda xatolarni tekshirish

Hozirgacha taqdim etilgan dasturlarda ko‘zlanganidek, V/V fayl operatsiyalarini bajarishda xatolar sodir bo‘lmaydi. Afsuski, bunga hamma vaqt ham erishib bo‘lmaydi. Masalan, agar siz kiritish uchun fayl ochayotgan bo‘lsangiz, dasturlar ushbu fayl mavjudligini tekshirib ko‘rishi kerak. Xuddi shunday, agar dastur ma’lumotlarni faylga yozayotgan bo‘lsa, operatsiya muvaffaqiyatli o‘tganiga ishonch hosil qilish kerak (masalan, diskda bo‘sh joyning yo‘qligi ma’lumotlarning yozib olinishiga to‘sqinlik qiladi). Xatolarni kuzatib borishda dasturlarga yordam berish uchun fayl obyektining fail funksiyasidan foydalanish mumkin. Agar fayl operatsiyasi jarayonida xatolar bo‘lmagan bo‘lsa, funksiya yolg‘on (0)ni qaytaradi. Biroq, agar xato uchrasa, fail funksiyasi haqiqatni qaytaradi. Masalan, agar dastur fayl ochadigan bo‘lsa, u xatoga yo‘l qo‘yilganini aniqlash uchun fail funksiyasidan foydalanishi kerak. Bu quyida shunday ko‘rsatilgan:

```
ifstream input_file(«FILENAME.DAT»);
if (input_file.fail())
{
cerr << «Ochilish xatosi FILENAME.EXT» << endl;
exit(1);
}
```

Shunday qilib, dasturlar o‘qish va yozish operatsiyalari

muvaffaqiyatli kechganiga ishonch hosil qilishlari kerak. TEST_ALL.CPP dasturi turli xato vaziyatlarni tekshirish uchun fail funksiyasidan foydalanadi:

```
#include <iostream.h>
#include <fstream.h>
void main(void)
{
char line[256] ;
ifstream input_file(«BOOKINFO.DAT») ;
if (input_file.fail()) cerr << «Ochilish xatosi BOOKINFO.DAT» << endl;
else
{
while ((! input_file.eof()) && (! input_file.fail()))
{
input_file.getline(line, sizeof(line)) ;
if (! input_file.fail()) cout << line << endl;
}
}
}
```

Faylning kerak bo‘lmay qolganda berkitilishi

Dasturni tugallash uchun operatsiya tizimi o‘zi ochgan fayllarni berkitadi. Biroq, odatga ko‘ra, agar dasturga fayl kerak bo‘lmay qolsa, uni berkitishi kerak. Faylni berkitish uchun dastur quyida ko‘rsatilganidek, dastur close funksiyasidan foydalanishi kerak:

```
input_file.close ();
```

Faylni yopayotganingizda dastur ushbu faylga yozib olgan barcha ma’lumotlar diskka tashlanadi va ushbu fayl uchun katalogdagi yozuv yangilanadi.

O‘qish va yozish operatsiyalarining bajarilishi

Hozirga qadar gap borayotgan dasturlar belgili satrlar ustida operatsiyalar bajarar edi. Dasturlaringiz murakkablashgan sari, ehtimol, sizga massivlar va tuzilmalarni o‘qish va yozish kerak bo‘lib qolar. Buning uchun dasturlar read va write funksiyalaridan foydalanishlari mumkin. read va write funksiyalaridan foydalanishda ma’lumotlar o‘qiladigan yoki yozib olinadigan ma’lumotlar buferini, shuningdek buferning baytlarda

o‘lchanadigan uzunligini ko‘rsatish lozim. Bu quyida ko‘rsatilganidek amalga oshiriladi:

```
input_file.read(buffer, sizeof(buffer));
output_file write(buffer, sizeof(buffer));
```

Masalan, STRU_OUT.CPP dasturi tuzilma ichidagisini EMPLOYEE.DAT fayliga chiqarish uchun write funksiyasidan foydalanadi:

```
#include <iostream.h>
#include <fstream.h>
void main(void)
{
struct employee
{
char name[64];
int age;
float salary;
} worker = { «Djon Doy», 33, 25000.0 };
ofstream emp_file(«EMPLOYEE.DAT») ;
emp_file.write((char *) &worker, sizeof(employee));
}
```

Odatda write funksiyasi belgilar satriga ko‘rsatkich oladi. (char*) belgilari turlarga keltirish operatori bo‘lib, bu operator siz ko‘rsatkichni boshqa turga uzatayotganingiz haqida kompilyatorga axborot beradi. Xuddi shunday tarzda STRU_IN.CPP dasturi read metodidan xizmatchi haqidagi axborotni fayldan o‘qib olish uchun foydalanadi:

```
#include <iostream.h>
#include <fstream.h>
void main(void)
{
struct employee
{
char name [6 4] ;
int age;
float salary;
} worker = { «Djon Doy», 33, 25000.0 };
ifstream emp_file(«EMPLOYEE.DAT»);
emp_file.read((char *) &worker, sizeof(employee));
```

```
cout << worker.name << endl;
cout << worker.age << endl;
cout << worker.salary << endl;
}
```

2.7. Istisnolar

C++ tili OMD doirasida istisnolarga xizmat ko‘rsatish standartini belgilab beradi. Dastur o‘zining ishlab chiqilishida ko‘zda tutilmagan normal bo‘lmagan vaziyatga duch kelganda, boshqaruvni ushbu muammoni hal qilishga qodir bo‘lgan dasturning boshqa qismiga berish mumkin hamda yo dasturni bajarishni davom ettirish yoki ishni tugallash kerak. *Istisnolarni joydan joyga tashlab berish (excpletion throwing)* dasturning normal bajarilishiga to‘sqinlik qiladigan sabablarning tashxisi uchun foydali bo‘lishi mumkin bo‘lgan axborotni tashlab berish nuqtasida to‘plash imkonini beradi. Siz dastur tugallanishi oldidan zarur xatti-harakatlarni bajaradigan *istisnolarga ishlov bergich (exception handler)*ni aniqlashingiz mumkin. Dastur ichida yuzaga keladigan sinxron istisnolar deb nomlanuvchi istisnolarga xizmat ko‘rsatiladi. Ctrl+C klavishalarini bosish kabi tashqi holatlar istisno hisoblanmaydi.

Istisnolarni generatsiya qila oladigan kod bloki try kalit-so‘z bilan boshlanadi va shakldor qavslar ichiga olinadi. Agar try bloki ushbu blok ichida istisnoni topib olsa, dasturiy uzilish sodir bo‘ladi hamda quyidagi xatti-harakatlar ketma-ketligi bajariladi:

1. Dastur istisnoga ishlov bergichning to‘g‘ri keladiganini qidiradi.

2. Agar ishlov bergich topilsa, stek tozalanadi va boshqaruv istisnolarga ishlov bergichga uzatiladi.

3. Agar ishlov bergich topilmagan bo‘lsa, ilovani tugatish uchun terminate funksiyasi chaqiriladi.

Yuzaga kelgan istisnoga ishlov beruvchi kod bloki catch kalit-so‘z bilan boshlanadi va shakldor qavs ichiga olinadi. Istisnoga ishlov bergichning kamida bitta kod bloki bevosita try blokining ortidan kelishi kerak. Dastur generatsiya qilishi mumkin bo‘lgan har bir istisno uchun o‘z ishlov bergichi ko‘zda tutilgan bo‘lishi kerak. Istisnolarga ishlov bergichlar navbatma-navbat ko‘rib chiqiladi hamda turi bo‘yicha catch operatoridagi argument (dalil) turiga to‘g‘ri keladigan istisnoga ishlov bergich tanlab olinadi. Ishlov bergich tanasida goto operatorlari bo‘l-

magan taqdirda berilgan try bloki istisnolariga ishlov bergichning oxirgisidan keyin kelgan nuqtadan boshlab dasturning bajarilishi yana davom etadi. Istisnolarga ishlov berishning umumlashma sxemasi quyidagi ko‘rinishga ega:

```
Try{
//Istisnoni generatsiya qilishi mumkin bo‘lgan har qanday kod
}
catch(T X) |
{
//T turdagi X istisnolarga ishlov bergich, istisno |
//avval kelgan try blokining ichida avval generatsiya qilingan bo‘lishi mumkin // Avval kelgan try catch (...) blokining boshqa istisnolariga ishlov bergichlar
//Avval kelgan try blokining har qanday istisnosiga ishlov bergichlar
}
```

Istisno yuzaga kelganda, throw operatoridagi <ifoda> nom berish ifodasi muvaqqat obyektni nomlaydi (initsiallashtiradi). Bunda muvaqqat obyektning turi ifoda argumenti (dalili) ning turiga mos keladi. Ushbu obyektning boshqa nusxalari, masalan, istisno obyektidan nusxa ko‘chirish konstruktori yordamida generatsiya qilinishi mumkin.

Garchi operatsiya tizimining o‘zi bajarilayotgan dasturni ko‘zda tutilmagan holatlardan «chiqarib olish»ga harakat qilsa hamki, xatolarga ishlov berishning qabul qilingan standart metodikasiga amal qilish — ishonchli ilovalarni qurishning kafolatlaridan biridir.

Misol:

Ikkita suzuvchi sonlarning bo‘linishidan hosil bo‘lgan bo‘linmaning interaktiv hisoblagichidan iborat dasturni ko‘rib chiqamiz. Bu dastur cheksiz sikl (davr) da bo‘linuvchi va bo‘luvchi qiymatlarini so‘rab oladi, favqulotda vaziyat yuzaga kelganda esa istisnoni keltirib chiqaradi. Istisnoni taqdim etayotgan MyDivideByZeroError sinfi istisnolarni boshqarish tizimi bilan samarali o‘zaro aloqa uchun barcha zarur elementlar to‘plamiga ega.

```
#include <iostream.h>
#include <string.h>
```

```
#define YESMESS «Biz davom etamiz.»
#define NOMESS  «Biz tugallaymiz.»
class MyDivideByZeroError
{
char *MyErrorMessage;
public:
char ContinueKey;
MyDivideByZeroError(): MyErrorMessage(NULL)
{
char YesKey;
cout << «Nolga bo‘lish qayd etildi» << endl;
cout << «Tezkor choralar ko‘rilsinmi? (Y/N)»;
cin >> YesKey;
if ( YesKey == ‘Y’ || YesKey == ‘y’ )
{
ContinueKey = 1;
MyErrorMessage = strdup(YESMESS);
}
else
{
ContinueKey = 0;
MyErrorMessage = strdup(NOMESS);
}
}
MyDivideByZeroError(const MyDivideByZeroError&
CopyVal)
{
ContinueKey = CopyVal.ContinueKey;
MyErrorMessage = strdup(CopyVal.MyErrorMessage);
}
~MyDivideByZeroError()
{
  if (MyErrorMessage) delete(MyErrorMessage);
}
void PrintMessage()
{
 cout << MyErrorMessage << endl;
}
};
float Dividor(float, float) throw(MyDivideByZeroError);
void main()
```

```
    {
    float MyVal1, MyVal2;
    for (;;)
    {
    //__ Nazorat ostidagi blokning boshi _________
    try
    {
    cout << «============================» <<
endl;
    cout << «MyVal1 >>»;
    cin >> MyVal1;
    cout << «MyVal2 >>»;
    cin >> MyVal2;
    cout << «Hisoblaymiz...» << Dividor(MyVal1, MyVal2) <<
endl;
    cout << «Uddaladik!»;
    }
    catch (MyDivideByZeroError MyExcept)
    {
    MyExcept.PrintMessage();
    if (MyExcept.ContinueKey = 0)
    {
    cout << «Xatolar bilan kurashish jonga tegdi! Ketdik.» <<
endl;
    break;
    }
    }
    //__ Nazoratdagi blokdan tashqarida _____________
    cout << «Blokdan tashqaridamiz. Davom etamiz...» <<
endl;
    }
    }
    float Dividor(float Val1, float Val2) throw
(MyDivideByZeroError)
    {
    if (Val2 == 0.0) throw MyDivideByZeroError();
    return Val1/Val2;
    }
```

Nazorat savollari

Nazariy savollar.

1. Sinflar. Ma'lumotlar va metodlar. Konstruktorlar va destruktorlar. Kirish huquqlari: public, protected va private. Konstantali funksiya-a'zolar va konstantali obyektlar. Statik elementlar va funksiyalar.
2. Vorislik tushunchasi. Vorislik turlari. Polimorfizm. Virtual funksiyalar.
3. Shablonlar. Funksiyalar shablonlari va sinflar shablonlari.
4. Oldindan belgilangan obyektlar va oqimlar. Oqimlar turlari. C++da fayllar bilan ishlash sinflari.
5. Xatolarni qayta ishlash. Istisno tushunchasi. Istisnolardan foydalanish. Try...Throw...Catch.

O'z-o'zini sinash uchun savollar.

1. C++ tilida class va struct orasida qanday farq mavjud?
2. Hamma elementlari private bo'lgan sinf yaratish mumkinmi?
3. Bo'sh sinf yaratish mumkinmi? Masalan: class Empty {}.
4. Sinf elementlari const atributga ega bo'lishi mumkinmi?
5. Konstruktorlar va destruktorlar vazifalari qanday?
6. Destruktorlar qanday chaqiriladi?
7. Sinf do'stlari nima uchun ishlatiladi?
8. Bazaviy va hosilaviy sinflar konstruktorlari qanday tartibda chaqiriladi?
9. Bazaviy va hosilaviy sinflar destruktorlari qanday tartibda chaqiriladi?

3.1. DHTMLga kirish

Gipermatnli ma'lumotlar tizimi ma'lumotlar bo'g'inlari to'plamlari, shu bo'g'inlarda aniqlangan gipermatnli aloqalar to'plamlari va bo'g'inlar va aloqalarni boshqarish instrumentlaridan tashkil topgan. World Wide Web texnologiyasi bu —gipermatnli taqsimlangan sistemalarni Internetga kiritish texnologiyasi va shundan kelib chiqib, u bunday tizimlarning umumiy ta'rifiga mos kelishi kerak. Bu shuni bildiradiki, gipermatnli tizimlarning yuqorida keltirilgan barcha komponentalari Webda ham bo'lishi kerak.

Webda gipermatnli tizimga ikki xil nuqtayi nazardan qarash mumkin. Birinchidan, o'zaro gipermatnli o'tishlar (ANCHOR konteyneri) vositasida bog'langan holda tasvirlanishi kerak bo'lgan sahifalar to'plami sifatida. Ikkinchidan, tasvirlanayotgan sahifalar (matn, grafika, uyali kod va hokazolar)ni tashkil qiluvchi elementlar ma'lumot obyektlarining to'plami sifatida. So'nggi holatda sahifaning gipermatnli o'tishlar to'plami − bu matnga ichki qo'yilgan rasm kabi ma'lumot bo'lagi.

Ikkinchi yondashuvda gipermatnli tizim elementari ma'lumot obyektlari to'plami uchun gipermatnli aloqalar o'rnini o'ynovchi HTML-sahifalarning o'zi tomonidan aniqlanadi. Bu yondashuv tayyor komponentalardan tasvirlanayotgan sahifalarni qurish nuqtayi nazardan ancha serhosilroqdir.

*Web*da sahifalarni yaratishda "klient-server" arxitekturasi bilan bog'liq muammo yuzaga chiqadi. Sahifalarni ham klient tomonida, ham server tomonida yaratish mumkin. 1995-yilda Netscape kompaniyasi mutaxassislari *JavaScript* dasturlash tilini ishlab chiqib, sahifalarni klient tomonida boshqarish mexanizmini yaratishdi.

Shunday qilib, *JavaScript* − bu Webni gipermatnli sahifalarini klient tomonida ko'rish senariyalarini boshqarish tili. Yanada aniqroq aytiladigan bo'lsa, *JavaScript* − bu nafaqat klient tomonidagi dasturlash tili. Liveware *JavaScript* tilining avlodi bo'lib, Netscape serveri tomonida ishlovchi vosita bo'ladi. Ammo *JavaScript* tilini mashhur qilgan narsa bu klient tomonida dasturlashdir.

*JavaScript*ning asosiy vazifasi — *HTML-konteynerlar* atributlarining qiymatlarini va ko‘rsatuvchi muhitining *xossalirini* HTML-sarlavhalarini ko‘rish jarayonida foydalanuvchi tomonidan o‘zgartirish imkoniyatlari, boshqacha aytganda ularni dinamik sarlavhalar qilish (DHTML)dir. Yana shuni aytish joizki, sarlavhalar qayta yuklanmaydi. Amalda buni, masalan, quydagicha ifodalash mumkin: sarlavhaning fonini, rangini yoki hujjatdagi rasmni o‘zgartirish, yangi oyna ochish yoki ogohlantirish oynasini chiqarish va h.k.

“*JavaScript*” nomi Netscape kompaniyasining xususiy mahsuloti. Microsoft tomonidan amalga oshirilgan til rasman *Jscript* deb nomlanadi. *Jscript* versiyalari *JavaScript*ning mos versiyalari bilan mos keladi (aniqroq qilib aytganda oxirigacha emas).

JavaScript — YeCMA (European Computer Manufacturers Association — Evropa Kompyuter Ishlab Chiqaruvchilar Assotsiyatsiyasi) tomonidan standartlashtirilgan. Mos standartlar quyidagicha nomlanadi: ECMA-262 va ISO-16262. Ushbu standartlar bilan *JavaScript* 1.1ga taqriban ekvivalent ECMAScript tili aniqlanadi. Eslatish joizki, bugungi kunda *JavaScript* ning hamma versiyalari ham ECMA standartlariga mos kelavermaydi. Mazkur kurs yoki qo‘llanmada barcha hollarda biz *JavaScript* nomidan foydalanamiz.

JavaScriptning asosiy xususiyatlari. JavaScript — bu Internet uchun katta bo‘lmagan klient va server ilovalarni yaratishga mo‘ljallangan nisbatan oddiy obyektga yo‘naltirilgan til. *JavaScript* tilida tuzilgan dasturlar HTML-hujjatning ichiga joylashtirilib ular bilan birga uzatiladi. Ko‘rish dasturlari (*brauzerlar — browser* ingliz so‘zidan) Netscape Navigator va Microsoft Internet Explorer hujjat matniga joylashtirilgan dasturlarni (*Scriptkod*) uzatishadi va bajarishadi.

Shunday qilib, *JavaScript* — interpritatorli dasturlash tili hisoblanadi. *JavaScript*da tuzilgan dasturlarga foydalanuvchi tomonidan kiritilayotgan ma’lumotlarni tekshirayotgan yoki hujjatni ochganda yoki yopganda biror bir amallarni bajaruvchi dasturlar misol bo‘lishi mumkin.

JavaScriptda yaratilgan dasturlarga misol sifatida foydalanuvchi tomonidan kiritilgan ma’lumotlarni tekshiruvchi, hujjatni ochish yoki yopish vaqtida qandaydir amallarni bajaruvchi dasturlarni keltirish mumkin. Bunday dasturlar foydalanuvchi tomonidan berilgan ko‘rsatmalarga — sichqoncha tugmasini bosilishiga, ma’lumotlarni ekran orqali kiritishiga yoki sichqon-

chani sahifa bo‘ylab siljitilishiga ko‘ra ish bajaradi. Bundan tashqari JavaScript dagi dasturlar brauzerning o‘zini va hujjatning atributlarini ham boshqarishi mumkin.

JavaScript dasturlash tili sintaktik jihatdan Java dasturlash tiliga, obyektli modellashni istisno qilgan holda, o‘xshab ketsada, lekin ma’lumotlarni statik tiplari va qat’iy tiplashtirish kabi xususiyatlarga ega bo‘lmaydi. JavaScriptda Java dasturlash tilidan farq qilib, sinf (klass) tushunchasi bu tilning asosiy sintaktik qurilmasi hisoblanmaydi. Bunday asos sifatida foydalanilayotgan tizim tomonidan qo‘llab-quvvatlanayotgan, oldindan aniqlangan ma’lumot tiplari: sonli, mantiqiy va satrli; mustaqil ham bulishi, obyektning metodi (JavaScripda metod tushunchasi funksiya/qism-dasturning uzi) sifatida ham ishlatilishi mumkin bo‘lgan funktsiyalar; katta sondagi o‘z xossalariga va metodlariga ega bo‘lgan oldindan aniqlangan obyektlardan iborat obyektli model va yana dastur ichida foydalanuvchi tomonidan yangi obyektlarni berish qoidalari hisoblanadi.

JavaScriptda dasturlar yaratish uchun hech qanday qo‘shimcha vositalar kerak bo‘lmaydi — faqatgina tegishli versiyadagi JavaScript qo‘llanishi mumkin bo‘lgan brauzer va DHTML-hujjatlarni yaratishga imkon beruvchi matn muharriri kerak bo‘ladi. JavaScriptdagi dastur bevosita HTML−hujjatlarni ichiga joylashtirilganligi uchun dastur natijasini — hujjatni brauzer yordamida ko‘rish orqali tekshirish mumkin va kerakli holda o‘zgartirishlar kiritilishi mumkin.

JavaScript dasturlash tilining imkoniyatlari. Uning yordamida HTML — hujjatlarning ko‘rinishi va tuzilishini dinamik ravishda boshqarish mumkin. Ekranda tasvirlanayotgan hujjatga brauzer tomonidan yuklangan hujjatning sintaktik tahlil qilish jarayonida istalgan HTML-kodlarni joylashtirish mumkin. “Document” obyekti yordamida foydalanuvchining oldingi bajargan amallari yoki boshqa bir faktorlarga ko‘ra yangi hujjatlarni avtomatik hosil qilish mumkin.

JavaScript yordamida brauzer ishini boshqarish mumkin. Masalan, *“window”* obyekti suzib yuruvchi oynalarni ekranga chiqarish, brauzerning yangi oynalarini yaratish, ochish va yopish, oynalarning yugurdagi va o‘lchamlarining rejimlarini o‘rnatish va hokazolarning imkoniyatini beruvchi metodlarga ega.

JavaScript hujjatdagi ma’lumotlar bilan bog‘lanish imkoniyatini beradi. Document obyekti va undagi mavjud obyektlar dasturlarga HTML-hujjatlarning qismlarini o‘qish va ba’zida

ular bilan bog‘lanish imkoniyatini beradi. Matnning o‘zini o‘qish mumkin emas, lekin masalan berilgan hujjatdagi gipermatnli o‘tishlar ro‘yhatini olish mumkin. Hozirgi vaqtda Form obyekti va undagi mavjud bo‘lishi mumkin bo‘lgan obyektlar: Button, Checkbox, Hidden, Password, Radio, Reset, Select, Submit, Text va Textarealar hujjatdagi ma’lumotlar bilan bog‘lanish uchun keng imkoniyatlar beradi.

JavaScript foydalanuvchi bilan aloqa qilishga imkon beradi. Bu tilning eng muhim xususiyati unda amalga oshirilgan hodisalarni qayta ishlashni aniqlash imkoniyati — ma’lum bir hodisaning (odatda foydalanuvchi tomonidan bajarilgan amal) ro‘y berish vaqtida bajariladigan dastur kodining ixtiyoriy qismi hisoblanadi. JavaScript hodisalarni qayta ishlovchi sifatida ixtiyoriy yangi oldindan berilgan funksiyalardan foydalanish imkoniyatini beradi. Masalan, foydalanuvchi sichqoncha ko‘rsatkichini gipermatnli o‘tishlar ustiga keltirsa, holatlar satrida mahsus xabarni chiqaruvchi, yoki ma’lum bir amalni bajarishni tasdiqlashni so‘rovchi dialogli oynani ekranga chiqaruvchi, yoki foydalanuvchi tomonidan kiritilgan qiymatlarni tekshiruvchi va xatolik yuz bergan holda kerakli ko‘rsatmalarni berib, to‘g‘ri qiymatni kiritishni so‘rovchi dasturlarni yaratish mum-kin.

JavaScript ixtiyoriy matematik hisoblashlarni bajarish imkoniyatini beradi. Bundan tashqari, bu tilda vaqt va sanalarning qiymatlari bilan ishlovchi yuqori darajada rivojlangan vositalar mavjud. JavaScript CGI-dasturlarga, Perl dasturlash tiliga va to‘ldiruvchi sifatida, ayrim hollarda Java tiliga muqobil til sifatida yaratilgan.

Har bir boshlovchi dasturchining asosiy savoli: “Dasturlar qanday tuziladi va bajariladi?”. Bu savolga iloji boricha soddaroq, lekin *JavaScript*-kodlarini qo‘llanilishining barcha usullarini unutmagan holda javob berishga harakat qilamiz.

Birinchidan, *JavaScript*-kodlari brauzer tomonidan bajariladi. Unda maxsus *JavaScript* interpritatori mavjud. Unga ko‘ra programmaning bajarilishi interpretator tomonidan boshqaruvni qachon va qay tarzda olishiga bog‘liq bo‘ladi. Bu esa o‘z navbatida kodning funksiyaviy qo‘llanilishiga bo‘g‘liq bo‘ladi. Umuman olganda *JavaScript* ning funksional qo‘llanishining 4 xil usulini ajratib ko‘rsatish mumkin:

1) gipermatnli o‘tish (URL sxemasi);

2) hodisalarni qayta ishlash (handler);

3) o‘rniga qo’yish(entity)

*4) qo‘yish (*SCRIPT *konteyneri).*

JavaScript bo‘yicha o‘quv qo‘llanmalarida *JavaScript* ni qo‘llashning bayoni, odatda, SCRIPT konteyneridan boshlanadi. Lekin dasturlash nuqtayi nazaridan bu unchalik ham to‘g‘ri emas, chunki bunday tartib asosiy savol: “*JavaScript*-kodi boshqaruvni qanday oladi?” ga javob bermaydi. Ya’ni *JavaScript* da yozilgan va HTML-hujjatning ichiga joylashtirilgan dastur qanday tarzda chaqiriladi va bajariladi.

HTML-sahifa muallifining kasbi va uning dasturlash asoslaridan xabardorligining darajasiga qarab, *JavaScript* ni o‘zlashtirishga kirishishni bir necha xil variantlari mavjud. Agar siz klassik tillar (C, Fortran, Pascal va h.) bo‘yicha dasturlovchi bo‘lsangiz, u holda hujjat matni ichida dasturlashdan boshlagan ma’qul, agar siz Windows sistemasida dasturlashga o‘rgangan bo‘lsangiz, u holda hodisalarni qayta ishlashni dasturlashdan boshlaganingiz ma’qul, agar siz faqat HTML bo‘yichagina tajribaga ega bo‘lsangiz yoki anchadan beri dasturlash bilan shug‘ullanmayotgan bo‘lsangiz, u holda gipermatnli o‘tishlarni dasturlashdan boshlaganingiz ma’qul.

URL-sxemali JavaScript. URL sxemasi (Uniform Resource Locator) — bu Web-texhologiyalarning asosiy elementlaridan biri. Webdagi har bir axborot resursi o‘zining o‘ziga xos URLiga ega bo‘ladi. URL A konteynerining HREF atributida, IMG konteynerining SRC atributida, FORM konteynerining ACTION atributida va hokozallarda ko‘rsatiladi. Barcha URL lar resursga ruxsatning protokoliga bog‘liq bo‘lgan ruxsat sxemalariga bo‘linadi, masalan, FTP-arxiviga kirish uchun ftp sxemasi, Gopher-arxiviga kirish uchun gopher sxemasi, elektron maktublarni jo‘natish uchun smtp sxemasi qo‘llaniladi. Sxemaning tipi URLning birinchi komponentasiga ko‘ra aniqlanadi. Bu holatda URL http bilan boshlanayapti — mana shu kirish sxemasini aniqlashdir (http sxemasi).

Gipermatnli sistemalar uchun dasturlash tillarining asosiy vazifasi gipermatnli o‘tishlarni dasturlashdir. Bu shuni bildiradiki, u yoki bu gipermatnli o‘tishlarni tanlashda gipermatnli o‘tishni amalga oshiruvchi dastur chaqiriladi. Web-texnologiyalarida standart dastur sifatida sahifani yuklash dasturi hisoblanadi. HTTP protokoli bo‘yicha standart o‘tishni *JavaScript* da dasturlangan o‘tishdan farq qilish uchun yaratuvchilar yangi URL sxemasi — *JavaScript* ni kiritishdi:

<A HREF=«JavaScript:JavaScript_kodi»>...</A>
<IMG SRC=«JavaScript:JavaScript_kod»>

Bu holatda «JavaScript_kod» matni birinchi holatda gipermatnli o‘tishni tanlanganda chaqiriladigan va ikkinchi holatda rasmni yuklashda chaqiriladigan *JavaScript* da yaratilgan dasturiy-qayta ishlovchilarni bildiradi.

Masalan, Diqqat!!! gipermatnli o‘tishga keltirilgan holda ogohlantiruvchi oynani chiqarish mumkin:

<A HREF=»JavaScript: alert(‘Diqqat!!!’);»> Diqqat!!!</A>

Formadagi submit tipidagi tugmani bosish orqali shu formadagi matnli maydonni to‘ldirish mumkin:

```
<FORM NAME=f METHOD=post
ACTION=«JavaScript: window.document.f.i.VALUE=
=’Tugma bosildi Click’;void(0);»>
<TABLE BORDER=0>
<TR>
<TD><INPUT NAME=i></TD>
<TD><INPUT TYPE=submit VALUE=Click></TD>
<TD><INPUT TYPE=reset VALUE=Reset></TD>
</TABLE>
</FORM>
```

URLda murakkab dasturlar va funksiya chaqirilishlarini joylashtirish mumkin. Faqatgina shuni yodda tutish kerakki, *JavaScript* sxemasi hamma brauzerlarda ham ishlamaydi, faqatgina Netscape Navigator va Internet Explorer larning to‘rtinchi versiyalaridan boshlab ishlaydi.

Shunday qilib, gipermatnli o‘tishlarni dasturlashda interpretator boshqaruvni foydalanuvchi sichqoncha tugmasini gipermatnli o‘tishga “bosgandan” keyingina oladi.

Hodisalarni qayta ishlovchilar. Hodisalarni qayta ishlovchi dasturlar (handler) shu hodisa bog‘liq bo‘lgan konteynerlarning atributlarida ko‘rsatiladi. Masalan, tugmani bosishda *onClick* hodisasi ro‘y beradi:

<FORM><INPUT TYPE=button VALUE=«Tugma» onClick=«window.alert(‘Diqqat!!!’);»></FORM>

O‘rniga qo’yish. O‘rniga qo’yish (entity) Web-sahifalarda anchagina kam uchraydi. Shunga qaramay u brauzer tomonidan HTML-sahifani hosil qilish uchun ishlatiladigan kerakli darajada kuchli vosita hisoblanadi. O‘rniga qo’yishlar HTML-konteynerlarning atributlari qiymatlari sifatida ishlatiladi. Masalan, foydalanuvchining shaxsiy sahifasini aniqlovchi forma maydonining boshlang‘ich qiymati sifatida joriy sahifaning URLi ko‘rsatiladi:

```
<SCRIPT>
function l()
{
str = window.location.href;
return(str.length);
}
</SCRIPT>
<FORM><INPUT VALUE=«&{window.location.href};»
SIZE=«&{l()};»>
</FORM>
<SCRIPT>
<!— Bu sharx ...JavaScript-kod...// —>->
</SCRIPT>
<BODY>
Asosiy hujjat ...
</BODY>
</HTML>
```

HTML-sharxlar bu yerda sahifaning berilgan bo‘lagini eski brauzerlardagi HTML-parserlar tomonidan interpretatsiya qilinishidan saqlanish uchun qo‘yilgan (yuqori boshqaruvdagilarda hali ham uchraydi). O‘z navbatida HTML-sharxining oxiri JavaScript interpretatori tomonidan interpretatsiya qilinishidan himoya qilingan (satr boshidagi // belgisi). Bundan tashqari konteyner boshlanishidagi tegning LANGUAGE atributining

qiymati sifatida *"JavaScript"* ko‘rsatilgan. *JavaScript* ga muqobil sifatida ko‘riluvchi VBScript keng qo‘llaniluvchi tildan ko‘ra ko‘proq ekzotika hisoblanadi, shuning uchun bu atributni tushirib qoldirish mumkin – *"JavaScript"* qiymati o‘z-o‘zidan qabul qilinadi.

O‘z-o‘zidan ayonki, hujjat sarlavhasida matnni hosil qilishni joylashtirish mantiqsizdir, u brauzer tomonidan namoyish qilinmaydi. Shuning uchun sarlavhada keyinchalik hujjat ichida foydalaniladigan umumiy o‘zgaruvchilar va funksiyalar e’lon qilinadi. Bunda Netscape Navigatorda Internet Explorerga nisbatan talab qat’iyroqdir. Agar funksiya sarlavhada e’lon qilinmagan bo‘lsa, hujjat ichida bu funksiya chaqirilsa, bu funksiya aniqlanmaganligi to‘g‘risidaga xabarni olish mumkin.

Funksiyani joylashtirish va ishlatishga oid misolni ko‘ramiz:

```
<HTML>
<HEAD>
<SCRIPT>
function time_scroll()
{
d = new Date();
window.status = d.getHours()+«:»+d.getMinutes()+«:»+d.
getSeconds();
setTimeout(‘time_scroll();’,500);
}
</SCRIPT>
</HEAD>
<BODY onLoad=time_scroll()>
<CENTER>
<H1>Holat satrida soat!</H1>
</CENTER>
</BODY>
</HTML>
```

Shuni ta’kidlab o‘tish kerakki, o‘rniga qo’yishlar Internet Explorer 4.0 da ishlamaydi, shuning uchun ulardan foydalanishda ehtiyotkorroq bo‘lish kerak. Brauzerga o‘rniga qo‘yish bo‘lgan sahifani berishdan avval shu brauzer tipini tekshirib olish kerak.

O‘rniga qo’yishlar bo‘lgan holda brauzer (parser komponent) tomonidan HTML-hujjatni tahlil qilish vaqtida interpre-

tator boshqaruvni o‘z qo‘liga oladi. Parser konteyner atributida &{..} konstruktsiyasini uchratishi bilan boshqaruvni *JavaScript* interpretatoriga beradi, u esa o‘z navbatida bu kodni bajargandan so‘ng boshqaruvni yana parserga qaytaradi. Shunday qilib, bu operatsiya HTML-sahifaga grafikani yuklashga o‘xshab ketadi.

Qo‘yish(SCRIPT konteyneri-interpretatorni majburiy chaqirish). SCRIPT konteyneri bu o‘rniga qo’yishlarni *JavaScript*-kod tomonidan hujjat matnini hosil qilish imkoniyati darajasigacha rivojlantirilishidir. Bu ma’noda SCRIPTni qo‘llanilishi Server Side Includesga o‘xshab ketadi, ya’ni server tomonidan hujjatlarning sahifalarini hosil qilishga. Lekin bu yerda biz sal ilgarilab ketdik. Hujjatni tahlil qilish vaqtida HTML-parser SCRIPT konteyneri boshlanishi tegini uchratgandan so‘ng boshqaruvni interpretatorga beradi. Interpretator SCRIPT konteynerining ichidagi barcha kod bo‘lagini bajaradi va SCRIPT konteyneri oxirini ko‘rsatuvchi tegdan so‘ng sahifa matnini qayta ishlash uchun boshqaruvni HTML-parserga qaytaradi.

SCRIPT konteyneri quyidagi 2 ta asosiy funksiyani bajaradi:

1) HTML-hujjat ichiga *kodni joylashtirish*;
2) brauzer tomonidan HTML-belgilarni *shartli hosil qilish*.

Birinchi funksiya keyinchalik o‘tishlar dasturi, hodisalarni qayta ishlovchilar va almashtirishlar sifatida ishlatish uchun o‘zgaruvchilar va funksiyalarni e’lon qilinishiga o‘xshab ketadi. Ikkinchisi bu hujjatni yuklash yoki qayta yuklash paytida JavaScript-kod bajarilishi natijalarini o‘rniga qo‘yishdir.

HTML-hujjat ichiga kodni joylashtirish. Xususan olganda bu yerda hech qanday ajralib turadigan har xillik yo‘q. Kodni hujjat sarlavhasida, HEAD konteynerining ichida yoki BODY ning ichiga joylashtirish mumkin. So‘nggi usul va uning xususiyatlari “Brauzer tomonidan HTML bo‘limlarini shartli hosil qilish” bo‘limida ko‘rib chiqiladi. Shuning uchun diqqatimizni hujjat sarlavhasiga qaratamiz.

Sarlavhada SCRIPT kodi konteynerining ichiga joylashtiriladi:

```
<HTML>
<HEAD>
<SCRIPT>
```

```
function time_scroll()
{
d = new Date();
window.status = d.getHours()+«:»+d.getMinutes()+«:»+d.
getSeconds();
setTimeout(‘time_scroll();’,500);
}
</SCRIPT>
</HEAD>
<BODY onLoad=time_scroll()>
<CENTER>
<H1> Holat satrida soat!</H1>
<FORM>
<INPUT TYPE=button VALUE=«Oynani yopish»
onClick=window.close()>
</FORM>
</CENTER>
</BODY>
</HTML>
```

Bu misolda biz time_scroll() funksiyasini hujjat sarlavhasida e’lon qildik, keyinchalik esa uni BODY konteyneri boshining tagida hodisalarni qayta ishlovchi load sifatida chaqirdik (onLoad=time_scroll()).

O‘zgaruvchilarni e’lon qilishga misol sifatida avvalgi-oyna tomonidan keyingi-oynaning statusini o‘zgartirishni ko‘ramiz.

Quyidagi funksiya yordamida hosilaviy oynani e’lon qilish va keyin chaqirish orqali yaratamiz:

```
function sel()
{
id = window.open(«», «example», «width=500,
height=200,status,menu»);
id.focus();
id.document.open();
id.document.write(«<HTML><HEAD>»);
id.document.write(«<BODY>»);
id.document.write(«<CENTER>»);
id.document.write(«<H1> Hosilaviy oynada matnni o‘zgar-
tirish.</H1>»);
id.document.write(«<FORM NAME=f>»);
id.document.write(«<INPUT TYPE=text NAME=t
```

```
SIZE=20 MAXLENGTH=20 VALUE='Bu test'>»);
    id.document.write(«<INPUT TYPE=button VALUE=
='Oynani yopish' onClick=window. close()></FORM>»);
    id.document.write(«</CENTER>»);
    id.document.write(«</BODY></HTML>»);
    id.document.close();
    }
    <INPUT TYPE=button VALUE=«Misol oynasida holat
satrini o'zgartirish»
    onClick=«id.defaultStatus='Salom'; id.focus();»>
```

Keyingi oynani ochishda id o'zgaruvchiga oyna *obyekt*iga bo'lgan ko'rsatkich id=window.open() ni joylashtirdik. Endi biz uni Window sinfining *o'byekt*i identifikatori sifatida ishlatishimiz mumkin. Bizning holatda id.focus() dan foydalanish shart. "Misoldagi oynadagi status maydonini o'zgartirish" tugmasini bosish orqali e'tiborni asosiy oynaga o'tkaziladi. U ekran o'lchamida bo'lishi mumkin. Bunda asosiy oyna tomonidan berkitib turilgan ikkinchi oynada o'zgarishlar ro'y beradi. O'zgarishlarni ko'rish uchun diqqatni yana ikkinchi oynaga berish kerak. id o'zgaruvchi qandaydir funktsiya tashqarisida aniqlangan bo'lishi kerak. Bu holatda u oynaning *xususiyati* ga aylanadi. Agar biz uni keyingi oynani ochish funktsiyasi ichiga joylashtirsak, u holda hodisalarni qayta ishlovchi click orqali unga murojaat qila olmaymiz.

Brauzer tomonidan HTML — bo'limlarini shartli hosil qilish. Serverdan o'zimizning brauzer imkoniyatlariga yoki foydalanuvchiga moslashtirilgan sahifalarni olish doimo yoqimli. Bunday sahifalarni hosil qilishning faqatgina 2 imkoniyati bor: server tomonidan yoki bevosita klient tomonidan. *JavaScript* — kod klient tomonida bajariladi (aslida Netscape kompaniyasi serverlari *JavaScript*-kodni server tomonida ham bajarishi mumkin, lekin bu holda u LiveWire-kod nomini oladi; LiveConnect bilan adashtirilmasin), shuning uchun faqatgina klient tomonida hosil qilinishini ko'rib chiqamiz.

HTML-bo'limlarini hosil qilish uchun SCRIPT konteyneri hujjat asosiy qismining ichiga joylashtiriladi. Oddiy misol-mahalliy vaqtni sahifaga moslashtirish:

```
<BODY>
...
<SCRIPT>
```

```
d = new Date();
document.write(«<BR>»);
document.write(«Sarlavhani yuklanish vaqti: «+d.getHours()+»:«+d.getMinutes()+«:»+d.getSeconds());
document.write(«<BR>»);
</SCRIPT>
...
</BODY>
```

JavaScriptda operatorlar, ifodalar, funktsiyalar

Operatorlar: arifmetik amallar, o‘zlashtirish, orttirish, kamaytirish.

Shartli ifodalar:

Qo‘shish “+”, ayirish “−”, ko‘paytirsh “*”, bo‘lish “/”, qoldiqli bo‘lish “%”.

Bu ifodalar har qanday sonli ifodalarda uchrashi mumkin.

Shuni ko‘rish mumkinki, JavaScriptdagi o‘zlashtirish operatorlari C va Java dagilar bilan bir xil: “=”, “+=”, “ −=”, “*=”, “/=”, “%=”.

x=y ifoda boshqa ko‘pgina dasturlash tillaridagi kabi “x” o‘zgaruvchiga “y” qiymatni berishni bildiradi.

Quyidagi operatorlar C dagi mos operatorlar bilan bir xil sintaktik qoidalarga ega:

y+=x bilan y=y+x ekvivalent
y−=x bilan y=y−x ekvivalent
y*=x bilan y=y*x ekvivalent
y/=x bilan y=y/x ekvivalent
y%=x bilan y=y%x — y ni x ga butun bo‘lgandagi qoldiq-ekvivalent.

Shartli ifodalar quyidagi ko‘rinishga ega;

(shart)?qiymat1:qiymat2

Agar *shart* ning qiymatu true bo‘lsa, shartli ifodaning qiymati *qiymat1* ga teng bo‘ladi, aks holda *qiymat2* ga teng bo‘ladi. Shartli ifodalarni oddiy ifodalarni qo‘llash mumkin bo‘lgan hamma joyda qo‘llash mumkin.

Misol:

```
a=(b<1)?0:(x−1)+c
```

Orttirish va kamaytirish operatorlari ham C dagi kabi sintaksisga ega: «x++», «++x», «x—», «—x».

Ifodalar:
y=x++ ikki o‘zlashtirishga ekvivalent: y=x; y=y+1,
y=++x ikki o‘zlashtirishga ekvivalent: x=x+1; y=x,
y=x— ikki o‘zlashtirishga ekvivalent: y=x; x=x−1,
y=—x ikki o‘zlashtirishga ekvivalent: x=x−1; y=x.

Satrli operatorlar:

Satrlar bilan ishlash uchun bir nechta operatorlar mavjud:

«+» — satrlarni qo‘shish s1+s2 (konkatenatsiya) s1 satrning simvollari ketma-ketligidan keyin s2 satrning simvollari ketma-ketligi kelgan satrni beradi.

eval(s) — JavaScriptda qurilgan ichki funksiya. U berilgan argument — bir yoki bir nechta JavaScript operatorlarini (nuqtali vergul bilan ajratilgan holda) o‘z ichiga olgan s satr bilan berilgan kodni bajaradi. Berilgan funksiyani nafaqat operatorni bajarish uchun balki ifodalarni hisoblash uchun ham ishlatish mumkin. U berilgan koddagi oxirgi hisoblangan ifodaning qiymatini qaytaradi. *eval(s)* funksiaysi foydalanuvchi tomonidan kiritish punktida kiritilgan qiymatni hisoblash va yana JavaScript dasturda bajarilayotgan kodni dinamik ravishda moslashtirish imkoniyatini beradi. U *parseInt* va *parseFloat* funksiyalariga nisbatan umumiyroqdir.

parseFloat(s) — JavaScriptda ichki qurilgan funksiyadir. U s satrdan birinchi belgidan son bo‘lmagan dastlabki belgigacha oraliqdan sonni qidiradi (Float tipidagi). Agar son topilmasa NaN (“Not a Number”) qiymatini qaytaradi.

parseInt(s) — butun sonlar uchun huddi o‘sha amalni bajaradi (Integer). Bunda avtomatik ravishda asos topiladi.

parseInt(s,n) — n asos bo‘yicha huddi o‘sha amal bajariladi (n 2 dan 36 gacha). n=0 bo‘lgan hol — bu *parseInt(s)* ni beradi. Bunda avtomatik ravishda asos topiladi.

Bitli o‘zlashtirish operatorlari:

Bitli o‘zlashtirishning bir nechta operatorlari mavjud:

x<<=n x=(x<<n) ga ekvivalent — ikkilik ko‘rinishdagi x sonda chapga n bit siljish;

x>>=n x=(x>>n) ga ekvivalent— ikkilik ko‘rinishdagi x sonda ishora bitini saqlagan holda o‘ngga n bit siljish (manfiy sonlarning qo‘shimcha kodida birinchi bit 1 ga teng bo‘ladi. Siljishdan keyin birinchi bitning o‘rniga 1 yoziladi va son manfiyligicha qoladi);

x>>>=n x=x>>>n ga ekvivalent — ikkilik ko‘rinishdagi x sonda birinchi bitiga 0 ni qo‘ygan holda o‘ngga n bit siljish (Siljishdan keyin birinchi bitning o‘rniga 0 yoziladi va son musbat songa aylanadi).

Chat tomondagi barcha ifodalar (bizda u — «x») 32-bitli butun sonlarga aylantirib olinadi, keyin siljish bajariladi, keyin hosil bo‘lgan son ifodaning — natijaning (bizda u yana “x”) tipiga aylantiriladi va o‘zlashtirish bajariladi.

Misollar:

1) 9<<2 ifoda 36 ni beradi, chunki 1001 (9 sonining 2 lik ko‘rinishdagi tasviri) da chapga 2 bit siljish 100100ni , ya’ni 36 ni beradi. Yetishmayotgan bitlar 0 lar bilan to‘ldiriladi.

9>>2 ifoda 2 ni beradi, chunki 1001 (9 sonining 2 lik ko‘rinishdagi tasviri)da o‘ngga 2 bit siljish 10 ni, ya’ni 2 ni beradi. Yetishmayotgan bitlar 0 lar bilan to‘ldiriladi.

«&» — bitli AND — «VA»;

«|» — bitli OR — «YOKI»;

«^» — bitli XOR — «INKOR QILUVCHI YOKI».

Barcha operatsiyalar sonlarning 2 lik ko‘rinishida bajariladi, lekin natija oddiy o‘nli ko‘rinishda qaytariladi.

Misollar:

15&9 ifoda 9 ni beradi, ya’ni (1111) AND (1001) ifoda 1001 ga teng;

15|9 ifoda 15 ni beradi, ya’ni (1111) OR (1001) ifoda 1111 ga teng;

15^9 ifoda 6 ni beradi, ya’ni (1111) XOR (1001) ifoda 0110 ga teng.

Mantiqiy ifodalar.

«&&» — mantiqiy AND — «VA»;

«||» — mantiqiy OR — «YOKI»;

«!» — mantiqiy NOT — «YO‘Q».

Misol:

(a>b)&&(b<=10)||(a>10)

JavaScriptda mantiqiy ifodalarni qisqartirilgan tekshirilishi deb ataluvchi amal doimo qo‘llaniladi: B1&&B2 operandda B1=false bo‘lgan holda B2 ni baholash bajarilmaydi va false qiymati qaytariladi. Shunga o‘xshab B1||B2 B1=true holatda true deb baholanadi. Bunda mantiqiy ifodalarning tahlili chapdan o‘ngga qarab olib boriladi va faqatgina to‘la ifoda baholanib bo‘lishi bilan natija qaytariladi. Shuning uchun agar B2 sifatida funksiya bo‘lsa, u chaqirilmaydi, va agar u aks ta’sirga ega bo‘lsa, bu xatolikka olib kelishi mumkin.

Taqqoslash operatorlari:

«= =» — «teng»;

«>» — «katta»;

«<» — «kichik»;

«>=» — «katta yoki teng»;

«<=» — «kichik yoki teng»;

«!=» — «teng emas».

Taqqoslash operatorlari nafaqat sonli ifodalarga, balki satrli ifodalarga ham qo‘llanilishi mumkin. Bunda satrlar teng hisoblanadi, qachonki ulardagi barcha simvollar ustma-ust tushsa va bir xil tartibda kelsa (bo‘sh belgi ham simvol deb qaraladi). Agar satrlar turli xil uzunliklarga ega bo‘lsa, u holda uzunroq satr katta hisoblanadi. Agar ularning uzunliklari teng bo‘lsa, u holda chapdan o‘ngga borgan sari kattaroq nomerdagi simvolga ega bo‘lgan satr katta hisoblanadi.

(a < b < c < ... < z < A < ... < Z).

Satrlarni qo‘shish mumkin, agar S1=«bu», S2=«mening satrim» bo‘lsa, u holda S!+S2 «bu mening satrim» ni beradi.

Operatorlarning bajarilish tartibi (kichigidan boshlab; bir satrdagilarning tartibi bir xil):

«+=», «−=», «*=», «/=», «%=», «<<=», «>>=», «>>>=», «&=», «^=», «|=».

Operatorlarning tartibi. Quyida operatorlarning bajarilish tartibi o‘sish tartibida berilgan:

shartli operator: «?:»;

mantiqiy «YOKI»: «||»;

mantiqiy «VA»: «&&»;

bitli «YOKI»: «|»;

bitli «XOR»: «^»;

bitli «VA»: «&»;

tenglikka solishtirish: «= =», «!=»;

solishtirish: «<», «<=», «>», «>=»;

bitli siljish: «<<», «>>», «>>>»;

qo‘shish, ayirish: «+», «—»;

ko‘paytirish, bo‘lish: «*», «/», «%» ;

inkor qilish, orttirish, kamaytirish: «!», «~», «++», «—»;

qavslar: «( )», «[ ]».

Funksiyalar:

JavaScriptda huddi C yoki Java dagi kabi protseduralar va protsedura-funksiyalar funktsiya deb ataladi. Funksiyalarni e’lon qilish quyidagilarni o‘z ichiga oladi:

Qabul qilingan *function* so‘zi;

Funksiayning nomi;

Yumaloq qavs ichiga olingan va vergul bilan ajratilgan funksiya argumentlari;

Figuraviy qavs ichiga olingan funksiyaning asosiy qismi.

Ya’ni:

```
function myFunction(arg1, arg2, ...)
{
...
Operatorlar ketma-ketligi
...
}
```

Bu yerda myFunction — funksiyaning nomi, arg1, arg2 — formal parametrlar ro‘yxati

Misol:

```
function Factorial(n)
{
if((n<0)||(round(n)!=n))
{
alert(«Factorial quydagi argumentda aniqlanmagan «+n);
return NaN;
}
else
{
result=(n*Factorial(n-1));
return result;
}
}
```

Funksiya qabul qilingan return so‘zi yordamida qiymatni qaytarmasligi mumkin.

Misol:

```
function Greeting(s)
{document.write(«Hello,»+s+»!»);
}
```

Funksiyani chaqirish aniq bir qiymatlar orqali bajariladi.

Misol:

Factorial(2+1);— 6 ni qaytaradi,

Greeting(«world»); — ekranga «Hello, world!» satrini chiqaradi.

Har bir funksiya, masalan, myFunction funksiyasi argumentlari arguments deb nomlangan massiv ko‘rinishida saqlanuvchi myFunction deb nomlanuvchi obyekt hisoblanadi, bunda uning argumentlariga quyidagicha murojaat qilinadi:

myFunction.arguments[i],

bu yerda i — argument nomeri (nomerlash 0 dan boshlanadi).

Funksiayning sonli argument qiymatlari soni funksiya e’lon qilingandagi formal parametrlari soniga teng yoki undan ortiq bo‘lishi kerak. Bunda funksiyani chaqirishdagi argumentlar soni myFunction.arguments.length maydonida saqlanadi va bu maydonning qiymatini qaytadan berish orqali dinamik ravishda o‘zgartirish mumkin.

Misol: Hujjatda HTML formatida ro‘yxatni chiqarish.

Bu yerda birinchi argument (ListType) tartiblangan ro‘yxat uchun «o» yoki «O» qiymatni olish va tartiblanmagan ro‘yxat uchun «u» yoki «U» qiymatni olishi mumkin. Keyin ro‘yxatning elementlari keladi.

```
function myList(ListType)
{
document.write(«<»+ListType+«L»);
for(var i=1; i < myList.arguments.length; i=i+1)
{document.write(«<LI>»+myList.arguments[i]);
}
document.write(«</»+ListType+»L>»);
}
```

HTML-hujjatda bu funksiyani chaqirish:

```
<script>
myList(«0», «bir», 2, «3»)
</script>
```

Quyidagi matnni chiqaradi:

bir
2
3

Shartli (nisbiy) o‘tishli funksiyalar

Shartli o‘tish operatori.

if operatori sintaksisining birinchi varianti:

```
if(shart)
{
tasdiq
}
```

Bu yerda tasdiq — operator yoki operatorlar ketma-ketligi.

Bu holatda shartli o‘tish operatori quyidagicha ishlaydi: avval shart tekshiriladi va agar uning qiymati true ko‘rinishida bo‘lsa, *tasdiq* bajariladi. Aks holda if dan keyin keluvchi operator bajariladi.

if operatori sintaksisining ikkinchi varianti:

```
if(shart)
{
tasdiq1
}
else
{
tasdiq2
}
```

Bu holda avval *shart* tekshiriladi va agar uning qiymati “true” ko‘rinishda bo‘lsa, *tasdiq1* bajariladi, aks holda, ya’ni “false” bo‘lsa, *tasdiq2* bajariladi.

Shartli o‘tish operatorining ishlatilishiga misol:

```
function checkData()
{
if (document.form1.threeChar.value.length==3)
{return true;
}
else
{alert('Roppa rosa 3 belgi kiriting');
return false;
}
}
```

s*witch* tanlash operatori:

```
switch (ifoda)
{
qiymat1: operator1 break;
qiymat2: operator2 break;
//...
default: operatorN break;
}
```

Tanlash operatori quyidagi tartibda ishlaydi: avval *ifoda* ning qiymati hisoblanadi, keyin uning *qiymat1* bilan tengligi tekshiriladi va agar u teng bo‘lsa, *operator1* bajariladi, keyin *ifoda* qiymatining *qiymat2* bilan tengligi tekshiriladi va agar u teng bo‘lsa,

operator2 bajariladi va hokazo. Agar *ifoda* qiymati hech bir qiymat: *qiymat1, qiymat2*, va hokazolarga teng bo‘lmasa, u holda o‘z-o‘zidan *operatorN* bajariladi.

Siklli (takrorlanuvchi) funksiyalar

for(boshlang‘ich qiymat sektsiyasi; shart sektsiyasi; hisoblagich o‘zgarishi sektsiyasi)
{
tasdiq
}

Bu seksiyalardan har biri ham bo‘sh bo‘lishi mumkin. Boshlang‘ich qiymat berish va hisoblagich o‘zgarishi seksiyalarida ifodalar ketma-ketligini vergul bilan ajratgan holda yozish mumkin. Siklni bajarilishi quyidagi tartibda bo‘ladi. Birinchi boshlang‘ich qiymat seksiyasi bajariladi. Keyin shart tekshiriladi. Agar shartning qiymati true bo‘lsa, u holda siklning asosiy qismi (*tasdiq*) bajariladi, keyin hisoblagich o‘zgartirgich seksiyasi bajariladi. Agar shartning qiymati false bo‘lsa, sikldan chiqiladi.

Misol:

```
function HowMany(SelectObject)
{
var numberSelected=0
for (i=0; i< SelectObject.options.leght; i++)
{
if (SelectObject.options[i].selected==true) number Selected++;
}
return numberSelected;
}
```

for operatori obyektdagi barcha maydonlarni ko‘rib chiqish uchun ishlatilishi mumkin (keyingi obyektli model haqidagi bo‘limni qarang).

Sintaksis:

```
for(o‘zgaruvchi  in obyekt)
{
ifoda
}
```

Bunda obyektdagi ko‘rsatilgan o‘zgaruvchining barcha mumkin bo‘lgan qiymatlari hosil qilinadi va ularning har biri uchun tasdiq bajariladi.

Misol: student sinfini va shu sinfning obyekti (ekzemplyar) Helen ni yaratamiz.

```
function student(name, age, group)
{
this.name=name;
this.age=age;
this.group=group;
}
function for_test(myObject)
{
for(var i in myObject)
{
document.write(«i=»+i+« => »+myObject[i]+«/n»);
}
};
Helen=new student(«Helen K.», 21, 409);
for_test(Helen);
```

Ekranga chiqarish:

```
i=0 => Helen K.
i=1 => 21
i=2 => 409
```

while tsikli:

```
while(shart)
{
ifoda
}
```

while siklining bajarilishi shartni tekshirishdan boshlanadi. Agar uning qiymati true ga teng bo‘lsa, sikl bajariladi, aks holda boshqaruv sikldan keyingi operatorga beriladi.

while operatorining ishlatilishiga misol:

```
n1=10
n=0
x=0
wxile(n<n1)
{
n=n+1;
```

```
x=x+n;
}
```

Sikllarni bajarilishini to‘xtatib qo‘yuvchi operatorlar *for* va *while* sikllarining joriy bajarilishlarini to‘xtatish uchun *break* operatori ishlatiladi.

break operatorining ishlatilishiga misol:

```
function test(x)
{
var j=0;
var sum=0;
while(n<n1)
{
if(n==x)
{ sum=x;
break;
}
sum=sum=n;
n=n+1;
}
return sum;
}
```

continue operatori for va while larning ichida joriy iteratsiyani bajarilishini to‘xtatadi va keyingi iteratsiyaga o‘tishni ta’minlaydi.

continue operatorining ishlatilishiga misol:

```
function test1(x)
{
var j=0;
while(n<n1)
{ if(n==x)
{ sum=x;
continue;
}
sum=sum=n;
n=n+1;
}
return sum;
}
```

Bu yerda biz JavaScript tilining asoslari, ma’lumotlarning

tiplari, ma'lumotlar ustida bajariladigan asosiy operatorlar va dasturning bajarilishini boshqarish kabi tushunchalarni hamda JavaScript-kodlarni HTML-hujjat ichiga joylashtirish usullarini ko'rib chiqdik.

3.2. HTML obyektlari

Bu yerda biz Javascriptda obyektlarga asoslangan dasturlash va hujjat (Web-sahifa)ning obyektli modelini ko'rib chiqamiz.

Obyektlarga asoslangan dasturlash tillarida sinflar obyektlarining ierarxiyasi mavjudligi faraz qilinadi. JavaScriptda bunday ierarxiya Window sinfining obyektlaridan boshlanadi, ya'ni har bir obyekt u yoki bu oynaga biriktirilgan bo'ladi. Har bir obyektga yoki uning xossasiga murojaat qilish uchun bu obyektning yoki uning xossasining shu obyekt kirgan ierarxiyadagi eng yuqori turgan obyektdan boshlab to'liq yoki qisman nomi ko'rsatiladi.

window
location
history
anchors[]
links[]
images[]
applets[]
embeds[]
frames[]
navigator
plugins[]
mimetipes[]
document
forms[]
elements[]
button
chekbox
hidden
password
radio
reset
Select(options[])
text

textarea
submit

Shuni aytishimiz mumkinki, obyektli modelning keltirilgan sxemasi Netscape Navigator ning 4 va undan yuqoki versiyalari hamda Microsoft Internet Explorer ning 4- va undan yuqoki versiyalari uchun to‘g‘ri. Yana bir marta ta’kidlab o‘tamizki, Internet Explorer va Netscape Navigator larda obyektli modellar turlichadir, keltirilgan sxema esa ularning umumiy qismlari asosida tuzilgan.

Umuman olganda *JavaScript* klassik obyektlarga asoslangan til hisoblanmaydi (uni yana yengillashtirilgan obyektli til deb ham atashadi). Unda meros qilib olish va polimorfizm yo‘q. Dasturchi o‘zining *obyektlar* sinfini function operatori yordamida aniqlab olishi mumkin, lekin ko‘pincha standart *obyektlar*, ularning konstruktorlaridan foydalanadi va sinflarning destruktorlaridan umuman foydalanmaydi. Bu shu bilan tushintiriladiki, *JavaScript*-dasturlarining bajarilish doirasi joriy oynadan tashqariga chiqmaydi.

Ba’zida *JavaScript* dagi turli obyektlarda bir xil nomdagi *xossalar* aniqlangan bo‘ladi. Bu holatda dasturchi qaysi *obyektning xossasidan* foydalanmoqchi ekanligini aniq ko‘rsatishi kerak. Masalan, Window va Document lar location *xossasiga* ega bo‘ladi. Faqat Window uchun bu Location sinfining obyekti, Document uchun esa qiymat sifatida yuklanayotgan hujjatning URLini oluvchi satrli literaldir.

Yana shuni hisobga olish kerakki, ko‘pgina obyektlar uchun obyektlar xossalarining qiymatlarini oddiy o‘zgaruvchilarga aylantirish uchun standart metodlar mavjud. Masalan, barcha obyektlar uchun belgilar satriga aylantirish uchun metod aniqlangan: toString().location bilan misolda agar satrli kontekstda window.location ga murojaat qilinsa, aylantirish o‘z-o‘zidan bajariladi va dasturchi buni payqamaydi:

```
<SCRIPT>
document.write(window.location);
document.write(«<BR>»);
document.write(document.location);
</SCRIPT>
```

Lekin baribir farq bor va anchagina. Xuddi shu misolda satrli konstantaning uzunligini olamiz:

```
<SCRIPT>
w=toString(window.location);
d=toString(document.location);
h=window.location.href;
document.write(w.length);
document.write(d.length);
document.write(h.length);
</SCRIPT>
```

Shunga osongina ishonch hosil qilish mumkinki, URL tipidagi obyektning xossasiga murojaat qilinsa, location xossasi aynan shu tipdagi obyekt hisoblanadi, almashtirishdan keyin simvollar satri uzunligi boshqacha bo‘ladi.

Barcha obyektlar uchun standartlar: xossa va hodisalar metodlari

Klient tomonida sahifalar ustidan boshqaruv mexanizmini yaratish uchun hujjatning obyektli modelidan foydalanish taklif qilingan. Modelning asosiy ma’nosi shundaki, har bir HTML-konteyner — bu quyidagi uchlik bilan harakterlanuvchi obyekt:

1) xossalar;
2) metodlar;
3) hodisalar.

Obyektli modelni sahifalar va brauzer orasidagi bog‘lanish usuli sifatida tasavvur qilish mumkin. Obyektli model — bu brauzerning dasturiy ta’minotida ishtirok etuvchi va ro‘y beruvchi, ular yordamida HTML kodi va sahifadagi senariy matnlari bilan ishlash qulay bo‘lgan ko‘rinishdagi obyektlar, metodlar va hodisalarning tasvirlanishidir. Biz uning yordamida brauzerga nima xohlashimizni bildirishimiz va unga mos ravishda ekrandagi sahifani o‘zgartirishimiz mumkin.

Bir xil xossa, metodlar va hodisalar to‘plamlariga ega bo‘lgan obyektlar bir xil tipli obyektlar sinflariga birlashtiriladi. Sinflar — bu bo‘lishi mumkin bo‘lgan obyektlarning ta’riflanishidir. *Obyektlarning* o‘zlari esa faqatgina brauzer tomonidan hujjatni yuklagandan so‘ng yoki dastur ishi natijasi sifatida paydo bo‘ladi. Buni yo‘q obyektga murojaat qilmaslik uchun doimo esda tutish kerak.

Xossalar.

Ko‘pgina HTML-konteynerlar atributlarga ega bo‘ladi. Masalan, yakor konteyneri <A ...>...</A> uni gipermatnli o‘tishga aylantiruvchi HREF atributga ega bo‘ladi:

<A HREF=tuit.htm>tuit</A>

Agar yakor konteyneri <A ...>...</A> ni obyekt sifatida ko‘radigan bo‘lsak, u holda HREF atributi “yakor” obyektining *xossasini* beradi. Dasturchi *atribut* qiymatini, shu bilan *obyektning xossasini* o‘zgartirishi mumkin:

document.links[0].href=«tuit.html»;

Barcha atributlarda ham qiymatlarni o‘zgartirish mumkin emas. Masalan, grafik rasmning balandligi va bo‘yi sahifada rasmni tasvirlashda dastlabki yuklanganiga ko‘ra aniqlanadi. Barcha keyingi rasmlar avvalgisiga ko‘ra masshtablashtiriladi. Shuni ta’kidlab o‘tish kerakki, Microsoft Internet Explorerda rasmning o‘lchamlari o‘zgartirilishi mumkin.

Tasvirning umumiyligi uchun JavaScriptda xossalar bilan HTML-bo‘laklarda muqobili bo‘lmagan obyektlar beriladi. Masalan, Navigator obyekti yoki brauzer oynasi deb ataluvchi, *JavaScript*dagi umuman eng yuqoкi obyekt hisoblanuvchi bajarish muhiti.

Metodlar.

JavaScript terminologiyasida *obyekt metodlari* deganda uning *xossalarini* o‘zgartiruvchi funksiyalar tushiniladi. Masalan, “document” obyekti bilan open(), write(), close() metodlari bog‘langan. Bu metodlar hujjatni hosil qilish yoki uning mazmunini o‘zgartirish imkoniyatini beradi. Quyidagi oddiy misolni ko‘ramiz:

```
function hello()
{              id=window.open(«»,»example»,»width=400,
height=150»);
id.focus(); id.document.open();
id.document.write(«<H1>Salom!</H1>»);
id.document.write(«<HR><FORM>»);
id.document.write(«<INPUT TYPE=button VALUE=
='Oynani yopish' «);
id.document.write(«onClick='window.opener.focus();win-
dow.close();'>»);
id.document.close();
}
```

Bu misolda open() metodi hujjatga yozish oqimini ochadi, write() metodi bu yozishni amalga oshiradi, close() metodi hujjatga yozish oqimini yopadi. Bularning barchasi oddiy faylga yozishdagi kabi ro‘y beradi. Agar oynada status maydonchasi bo‘lsa (odatda unda hujjatni yuklash darajasi tasvirlanadi), yopil-

magan hujjatga yozish oqimi bo‘lgan holda unda huddi hujjatni yuklashdagi kabi yozishni davom etayotganligini ko‘rsatuvchi to‘rtburchak tasvirlanadi.

Hodisalar.

Metodlar va *xossalardan* tashqari obyektlar hodisalar bilan ham xarakterlanadi. Xususan, *JavaScript*da dasturlashning mazmuni ana shu hodisalarni qayta ishlovchilarni yozishdan iboratdir. Masalan, button tipidagi obyekt (button tipidagi INPUT konteyneri − “Tugma”) bilan click hodisasi bo‘lishi mumkin, ya’ni foydalanuvchi tugmani bosishi mumkin. Buning uchun INPUT konteynerining atributlari click hodisasini qayta ishlovchi atribut — onClick bilan kengaytirilgan. Bu atributning qiymati sifatida HTML-hujjatning muallifi *JavaScript*da yozadigan hodisani qayta ishlash dasturi ko‘rsatiladi:

<INPUT TYPE=button VALUE=«Bosing» onClick=«window.alert(‘Iltimos, yana bir marta bosing’);»>

Hodisalarni qayta ishlovchilar shu hodisalar bog‘langan konteynerlarda ko‘rsatiladi. Masalan, BODY konteyneri butun hujjatni xossalarini aniqlaydi, shuning uchun butun hujjatni yuklanib bo‘lish hodisasini qayta ishlovchi bu konteynerda onLoad atributining qiymati sifatida ko‘rsatiladi.

Qat’iy qilib aytganda har qanday brauzer, xoh u Internet Explorer yoki Netscape Navigator, va yoki Opera bo‘lsin, o‘zining obyektli modeliga ega bo‘ladi. Turli xil brauzerlar (va xatto bitta brauzerning turli xil versiyalari)ning obyektli modellari bir-birlaridan farq qiladi, lekin prinsipial jihatdan bir xil strukturaga ega bo‘ladi. Shuning uchun ularning har biriga alohida to‘xtalib o‘tishdan ma’no yo‘q. Biz barcha brauzerlarga qo‘llash mumkin bo‘lgan umumiy yondashuvlarni va ayrim hollarda ular orasidagi farqlarni ko‘rib chiqamiz.

Window (oyna) obyekti

Window obyektlar sinfi — bu JavaScriptdagi obyektlar ierarxiyasidagi eng yuqoкi sinf. Unga Window obyekti va Frame obyekti kiradi. Window obyekti brauzer-dastur oynasi bilan, Frame obyekti esa HTML-sahifa muallifi tomonidan FRAMESET va FRAME konteynerlaridan foydalanganda brauzer oynasi tomonidan hosil qilinuvchi va uning ichida joylashgan oynalar bilan bog‘lanib ketadi.

JavaScriptda dasturlashda ko‘pincha Window tipidagi obyektlarning quyidagi xossalari va metodlari ishlatiladi:

Xossalar	Metodlar	Hodisalar
status	open()	Hodisalar yo‘q
location	close()	
history	focus()	
navigator		

Window obyekti faqatgina oynani ochish vaqtidagina yaratiladi. Sahifani oynaga yuklashda paydo bo‘ladigan barcha qolgan obyektlar Window obyektining xossalaridir. Shunday qilib turli xil sahifalarni yuklashda Windowning xossalari turlicha bo‘lishi mumkin.

Status maydoni (holatlar paneli). **Status maydoni** — bu HTML-sahifa mualliflari JavaScriptdan foydalanishni boshlagan birinchi narsadir. Kalkulyatorlar, o‘yinlar, matematik hisoblashlar va boshqa elementlar judayam sun’iy ko‘rindi. Buning natijasida status maydonidagi yugurib yuruvchi satr Web dan foydalanuvchilarning diqqatini haqiqatdan torta oladigan durdona bo‘lgan edi. Asta sekin uning ommaviyligi yo‘qola bordi. Yuguruvchi satrlar juda kam qo‘llaniladigan bo‘ldi, lekin status maydonchasini dasturlash ko‘pgina Web-bo‘g‘imlarida uchraydi.

Status maydoni (status bar) deb HTML-sahifani tasvirlovchi sohaning shundoq ostidagi brauser oynasining quyi qismidagi o‘rta maydonga aytiladi. Status maydonida brauzer holati (hujjatni yuklash, grafikani yuklash, yuklashni tugatish, appletni bajarish va h.k.) haqidagi ma’lumot tasvirlanadi. JavaScript dagi dastur bu maydon bilan oynaning o‘zgaruvchi xossasi kabi ishlash imkoniyatiga ega. Bunda amalda u bilan 2 ta turli xildagi xossalar bog‘langan bo‘ladi:

1) *window.status;*
2) *window.defaultStatus.*

Ular o‘rtasidagi farq shundaki, amalda brauzer ba’zi bir hodisalar bilan bog‘liq bo‘lgan bir nechta holatlarda bo‘ladi. Brauzer holati status maydonidagi xabarda aks etadi. Umuman olganda faqatgina 2 ta holatlar mavjud: hech qanday hodisalar yo‘q (defaultStatus) va qandaydir hodisalar ro‘y berayapti (status).

Statusni dasturlaymiz. Status xossasi sahifani oddiy yuklashdan farq qiluvchi hodisalar to‘g‘risidagi xabarlarni tasvirlash bilan bog‘liq. Masalan, sichqoncha kursori gipermatnli o‘tish ustidan o‘tayotganda HREF atributida ko‘rsatilgan URL status maydonchasida aks ettiriladi. Sichqoncha kursori gipermatnli o‘tishdan xoli bo‘lgan maydonga o‘tishi bilan status maydonchasida boshlang‘ich xabar tiklanadi (Document: Done). Bu texnika berilgan sahifada status va defaultStatus larni bayon qilishda amalga oshirilgan.

<A HREF=#status onMouseover=»window.status=’Jump to status description’;return true;»

onMouseout=»window.status=’Status bar programming’; return true;»>window.status</A>

JavaScriptning dokumentatsiyasida ko‘rsatilganki, mouseover va mouseout hodisalarni qayta ishlovchilar true qiymatini qaytarishi kerak. Bu brauzer boshlang‘ich harakatlarni bajarmasligi uchun kerak bo‘ladi. Tajriba shuni ko‘rsatadiki, Netscape Navigator 4.0 true qiymati holatida ham juda yaxshi ishlaydi.

defaultStatusni dasturlaymiz. defaultStatus xossasi hech qanday hodisa ro‘y bermayotgan vaqtda status maydonida aks ettirilayotgan matnni aniqlaydi. Bizning hujjatda biz uni hujjatni yuklash vaqtida aniqladik:

<BODY onLoad=»window.defaultStatus=’Status bar programming’;»>

Bu xabar sahifaning barcha komponentalari (matn, grafika, appletlar va h.k.) yuklab bo‘lingandan keyin paydo bo‘ladi. U hujjatni ko‘rishda ro‘y berishi mumkin bo‘lgan xohlagan hodisadan qaytgandan keyin status maydonchasida qayta tiklanadi. Qizig‘i shundaki, sichqonchani gipermatnli o‘tishlardan xoli bo‘lgan sohalar bo‘ylab harakatlantirilishi defaultStatus ni doimo aks ettirilishiga olib keladi.

location maydoni (manzil satri). Manzil maydonida yuklangan hujjatning URLi aks ettiriladi. Agar foydalanuvchi o‘zicha qaysidir sahifaga o‘tmoqchi bo‘lsa (uning URLini terib), u buni location maydonida amalga oshiradi. Maydon brauzer oynasining yuqoqi qismida, instrumentlar panelidan quyida, lekin shaxsiy tanlovlar panelidan yuqoqida joylashgan.

Umuman olganda Location — bu obyekt. JavaScript versiyalaridagi o‘zgarishlar tufayli Location quyi sinf sifatida Window sinfiga ham, Document sinfiga ham kiradi. Biz

Location ni faqatgina window.location sifatida ko‘rib chiqamiz. Bundan tashqari Location — bu Area va Link sinfining obyektlari kiruvchi URL sinfining quyi sinfi hamdir. Location URL ning barcha xossalarini meros qilib oladi va bu unga URL sxemasining istalgan qismiga kirishga imkon beradi.

Location obyektining xarakteristikalarini va ishlatish usullarini ko‘rib chiqamiz:

- *xossalar;*
- *metodlar;*
- *Locationni xarakterlovchi hodisalar yo‘q.*

Ko‘rib turibmizki, Location obyektining xarakteristikalari ro‘yhati to‘liq emas.

Xossalar. Aytaylik, brauzer quyida berilgan manzildagi sahifani aks ettirayotgan bo‘lsin: **http:// tuit.uz:80/r/dir/page? search#mark.**

U holda Location obyektining xossalari quyidagi qiymatlarni qabul qilishi mumkin:

```
window.location.href =
http://tuit.uz:80/r/dir/page?search#mark
window.location.protocol = http;
window.location.hostname = tuit.uz;
window.location.host = tuit.uz:80;
window.location.port = 80
window.location.pathname = /r/dir/;
window.location.search = search;
window.location.hash = mark;
```

Metodlar. Location metodlari sahifani yuklashni va qayta yuklashni boshqarish uchun mo‘ljallangan. Bu boshqaruv shuni bildiradiki, hujjatni qayta yuklash (reload) yoki yuklash (replace) mumkin. Bunda sahifalarni ko‘rish trassasiga (history) ma’lumot kiritilmaydi:

```
window.location.reload(true);
window.location.replace(‘#top’);
```

reload() metodi instrumentlar panelidagi Reload tugmasini bosgandagi brauzer harakatini to‘laligicha modellaydi. Agar bu metodni argumentsiz yoki unga true qiymat bergan holda chaqirilsa, brauzer hujjatning oxirgi o‘zgartirilgan vaqtini tekshiradi va uni yoki keshdan (agar hujjat o‘zgartirilmagan bo‘lsa) yoki serverdan yuklaydi. Bunday harakat Reload tugmasining

oddiygina bosish bilan mos keladi. Agar argument sifatida false ko‘rsatilsa, u holda brauzer hujjatni har qanday holatda ham serverdan yuklaydi. Bunday harakat Reload va Shift tugmalrini birgalikda bosish bilan mos keladi (Reload+Shift).

replace() metodi bir sahifani ikkinchisi bilan shunday almashtirishga imkon beradiki, bu almashtirish HTML-sahifalarni ko‘rish trassasi (history)da aks ettirilmaydi va instrumentlar panelidagi Back tugmasini bosish orqali foydalanuvchi doimo dastlabki oddiy usulda (gipermatnli o‘tish bo‘yicha) yuklangan sahifaga qaytadi. Eslatib o‘tamizki, Location xossasini o‘zgartirishda ham sahifani qayta yuklash ro‘y beradi, lekin bu holda o‘tish haqidagi ma’lumot history ga kiritiladi.

Kirishlar tarixi (History). World Wide Web sahifalariga kirishlar tarixi (trassa) foydalanuvchiga u bir necha minut (soat, kun) oldin ko‘rgan sahifaga qaytish imkoniyatini beradi. Kirishlar tarixi JavaScriptda history sinfining obyektiga aylantiriladi. Bu obyekt foydalanuvchi ko‘rgan va brauzer menyusidagi GO rejimini tanlagan holda olishi mumkin bo‘lgan URL-sahifalar massivini ko‘rsatadi. history obyekti metodlari shu massivdagi URLdan foydalangan holda sahifalarni yuklashga imkon beradi.

Brauzer havfsizligi bilan muammolar bo‘lmasligi uchun History bo‘yicha faqatgina URLning indeksi bo‘yicha sayr qilish mumkin. Bunda URL matnli qator sifatida dasturchiga berilmaydi. Ko‘pincha bu obyekt bir nechta turli xil sahifalarga o‘tishlar bo‘lgan misollar yoki sahifalarda misol yuklanadigan sahifaga qaytish mumkin deb faraz qilgan holda foydalaniladi:

```
<FORM><INPUT TYPE=button VALUE=»Orqaga» onClick=history.back()></FORM>
```

Berilgan kod bosish orqali oldingi sahifaga qaytishimiz mumkin bo‘lgan “Orqaga” tugmasini aks ettiradi.

Brauzer tipi (Navigator obyekti). Brauzerlar o‘rtasidagi “urush” tufayli (uni allaqachon Microsoft Internet Explorer foydasiga hal bo‘ldi deb hisoblash mumkin) sahifani aniq bir ko‘rish dasturiga moslashtirish muammosi vujudga keldi. Bunda ikki xil variant bo‘lishi mumkin: server tomonidagi brauzer tipini aniqlash va klient tomonidagi brauzer tipini aniqlash. Oxirgi variant uchun JavaScriptda Navigator obyekti mavjud. Bu obyekt — Window obyektining xossasi.

Ko‘rish dasturining tipini aniqlashga oddiy misolni ko‘ramiz:

<FORM><INPUT TYPE=button VALUE=«Navigator brauzer tipi»
onClick=»window.alert(window.navigator.userAgent);»></FORM>

Tugmani bosish bilan ogohlantirish oynasi aks ettiriladi. Unda tegishli brauzer HTML-sarlavhaga joylashtiradigan userAgent satri bo‘ladi.

Bu satrni komponentalar bo‘yicha bo‘laklash mumkin, masalan:

navigator.appName = Microsoft Internet Explorer
navigator.appCodeName = Mozilla
navigator.appVersion = 4.0 (compatible; MSIE 5.5; Windows 98)
navigator.userAgent = Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)

Navigator obyektini dasturlash tuqtayi nazaridan qiziqarli bo‘lgan bir necha xil qo‘llash usullari mavjud. Masalan, Javani qo‘llash mumkinligini tekshirish.

Bu imkoniyatni misolda ko‘rsatamiz:

```
<SCRIPT>
document.write(«<P ID=red>»);
if(navigator.javaEnabled()==true)
document.write(«Sizni dasturingiz Java-appletlarni ishlashi ta’minlaydi «);
if(navigator.javaEnabled()==false)
document.write(«<FONT COLOR=red> Sizni dasturingiz Java-appletlarni ishlashi ta’minlamaydi </FONT>»);
</SCRIPT>
</example>
```

Shunga o‘hshab sizning brauzeringizda ishlatsa bo‘ladigan grafik fayllar formatlarini ham tekshirish mumkin:

```
<SCRIPT>
if(navigator.mimeTypes[‘image/gif’]!=null)
document.write(«Sizni brauzeringiz GIF ishlashini ta’minlaydi <BR>»);
if(navigator.mimeTypes[‘image/tif’]==null)
document.write(«Sizni brauzeringiz TIFF ishlashini ta’minlamaydi «);
</SCRIPT>
```

Afsuski, bunday tekshiruv grafikani avtomatik yuklashni mavjudligini aniqlashga imkon bermaydi.

Oynalarni boshqarish. Oynalar bilan nimalar qilish mumkin? Ochish (yaratish), yopish (yo‘qotish), uni boshqa barcha ochiq oynalar ustiga joylashtirish (diqqatni berish). Bundan tashqari oynaning va unga bo‘ysungan obyektlar xossalarini boshqarish mumkin. Asosiy xossalarning bayoni “Brauzer oynasining xossalarini dasturlaymiz” bo‘limida berilgan, shuning uchun asosiy e’tiborni oddiy va ko‘proq ishlatiladigan oynalarni boshqarish metodlariga qaratamiz:

- *alert();*
- *confirm();*
- *prompt();*
- *open();*
- *close();*
- *focus();*
- *setTimeout();*
- *clearTimeout()*.

Bu yerda faqat ikki metod: scroll() и blur() lar ko‘rsatilmagan.

Birinchisi oynani berilgan pozitsiyaga joylashtirishga imkon beradi. Lekin uni oynaning koordinatalarini bilmagan holda ishlatish juda qiyin. Keyingisi esa oddiy ish hisoblanadi, agarki qatlamlamlarni dasturlash texnologiyalari yoki CSS (Cascading Style Sheets) lar ishlatilmayotgan bo‘lsa.

Ikkinchi metod diqqatni oynadan oladi. Bunda diqqatni qayerga qaratish umuman ma’lum bo‘lmaydi. Yaxshisi diqqatni yo‘qotgandan ko‘ra uni aniq maqsadli yo‘naltirgan ma’qul.

window.alert()

alert() metodi ogohlantirish *oynasini* chiqarishga imkon beradi:

<A HREF=»javascript:window.alert(‘Diqqat’)»>

So‘rovni qaytaring!</A>

Hammasi juda oddiy, lekin shuni nazarda tutish kerakki, xabarlar sistema fontlarida chiqariladi, shuning uchun rus tilidagi xabarlarni olish uchun OSning mahalliylashtirilgan varianti bo‘lishi kerak.

window.confirm()

confirm() metodi foydalanuvchiga u ma’qullashi yoki rad qilishi mumkin bo‘lgan savollarni berishga imkon beradi:

<FORM>

<INPUT TYPE=button VALUE=»Siz JavaScript ni bilasizmi?»

```
onClick=»if(window.confirm(‘Hammasini
bilaman’)==true)
{ document.forms[0].elements[1].value=’Ha’; }
else {
document.forms[0].elements[1].value=’Yo‘q’;
};»><BR>
</FORM>
```

Rus tilidagi xabarlar uchun alert() metodi uchun bayon qilingan barcha cheklashlar confirm() metodi uchun ham o‘rinli.

window.prompt()

prompt() metodi foydalanuvchidan informatsion oynaning kiritish maydoniga teriladigan qisqa matn satrini qabul qilishga imkon beradi:

```
<FORM>
<INPUT TYPE=button VALUE=»Kiritish oynasini
oching»
onClick=»document.forms[1].elements[1].value=window.pr
ompt(‘Xabarni kiriting’);»>
<INPUT SIZE=30>
</FORM>
```

Foydalanuvchi tomonidan kiritilgan satr istalgan o‘zgaruvchi tomonidan o‘zlashtirilishi mumkin va keyin JavaScript-dasturda ishlatish mumkin.

window.open()

Oynaning bu metodida atributlar boshqa obyektlarnikiga qaraganda ko‘proq. Open() metodi yangi oynalarni yaratishga mo‘ljallangan. Umumiy holda uning sintaksisi quyidagi ko‘rinishda bo‘ladi:

open(«URL»,»window_name»,»param,param,...», replace);

Bu yerda: URL — yangi oynaga yuklanadigan sahifa, window_name — A va FORM konteynerlaridagi TARGET atributida ishlatish mumkin bo‘lgan oyna nomi.

Parametrlar	Ishlatilishi
replace	Oynani ochish vaqtida History massiviga yozishni boshqarishga imkon beradi
param	Parametrlar ro'yxati
width	Oynaning piksellardagi kengligi

height	Oynaning piksellardagi balandligi
toolbar	Brauzerning sistema tugmalari bo‘lgan oynani yaratadi
location	Location maydonli oynani yaratadi
directories	Foydalanuvchining tanlovlari menyuli oynani yaratadi
status	Status maydonli oynani yaratadi
menubar	Menyuli oynani yaratadi
scrollbar	Yugurdak polosali oynani yaratadi
resizable	O‘lchamini o‘zgartirsa bo‘ladigan oynani yaratadi

Quyidagi misolni keltiramiz:

```
<FORM>
<INPUT TYPE=button VALUE=»Oddiy oyna» onClick=»window.open(‘about:blank’,’test1’,’directories=no,height=200,location=no,menubar=no,resizable=no,scrollbars=no,status=no,toolbar=no,width=200’);»>
<INPUT TYPE=button VALUE=»Murakkab oyna» onClick=»window.open(‘about:blank’,’test2’,’directories=yes,height=200,location=yes,menubar=yes,resizable=yes, scrollbars=yes,status=yes,toolbar=yes,width=200’);»>
</FORM>
```

“oddiy oyna” tugmasiga bosib, quyidagi parametrli *oynani* hosil qilamiz:

- directories=no — menyusiz *oyna*
- height=200 — balandligi 200 px
- location=no — location maydoni yo‘q
- menubar=no — menyusiz
- resizable=no — o‘lchamini o‘zgartirish mumkin emas
- scrollbars=no — yugurdak polosasi yo‘q
- status=no — status satri yo‘q
- toolbar=no — brauzerning sistema tugmalri yo‘q
- width=200 — kengligi 200

“murakkab oyna” tugmasini bosib, quyidagi *oynani* olamiz:

- directories=yes — menyuli oyna
- height=200 — balandlik 200 px
- location=yes — location maydoni mavjud
- menubar=yes — menyu mavjud
- resizable=yes — o‘lchamni o‘zgartirish mumkin
- scrollbars=yes — yugurdak polosasi mavjud
- status=yes — status satri mavjud
- toolbar=yes — brauzerning sistema tugmali mavjud
- width=200 — kenglik 200

window.close()

close() metodi — bu open() metodining aks tomonidir. U *oynani* yopish imkoniyatini beradi. Ko‘pincha aynan qaysi *oynani* yopish kerak degan savol tug‘iladi.

Agar joriy oynani yopish kerak bo‘lsa, u holda:

```
window.close();
self.close();
```

Agar bosh *oynani*, yani joriy oyna -ochilgan oynani yopish kerak bo‘lsa, u holda:

```
window.opener.close();
```

Agar ixtiyoriy *oynani* yopish kerak bo‘lsa, u holda avval uning identifikatorini olish kerak bo‘ladi:

```
id=window.open();
...
id.close();
```

Oxirgi misoldan ko‘rinib turibdiki, oynani nomi bo‘yicha emas (TARGET atributining qiymati bu yerda ahamiyatga ega emas), balki obyektning ko‘rsatkichi bo‘ycha yopiladi.

window.focus()

focus() metodi diqqatni u ishlatilgan *oynaga* qaratish uchun ishlatiladi. Diqqatni berish oynani qaysi vaqtda tanlash holatlarini eslamasdanoq, oynani ochishda ham, yopishda ham juda foydali. Misol ko‘ramiz.

Oynani ochamiz va uni yopmasdan turib yana shu nomdagi, lekin boshqacha matnli oynani ochamiz. Yangi oyna asosiy oynaning ustida paydo bo‘lmaydi, chunki diqqat unga berilmadi. Endi oyna ochilishini takrorlaymiz, faqat bu safar diqqatni berish orqali:

```
function myfocus(a)
{
```

```
id = window.open(«»,»example»,»scrollbars,width=300,
height=200»;
//oynani ochamiz va unga ko‘rsatkich bo‘lgan o‘zgaruvchi-
ni kiritamiz
//agar shu nomli oyna mavjud bo‘lsa, yangi oyna yaratil-
maydi,
//faqatgina shu oynaga yozish uchun oqim ochiladi
if(a==1)
{
id.document.open();
//yaratilgan oynaga yozish uchun oqim ochamiz
id.document.write(«<CENTER>>Oynani birinchi marta
ochdingiz»);
//Bu oqimga yozamiz
}
if(a==2)
{
id.document.open();
id.document.write(«<CENTER>Oynani ikkinchi marta
ochdingiz»);
}
if(a==3)
{
id.focus();
//diqqatni beramiz, keyin oldingi holdagi amallarni
bajaramiz
id.document.open();
id.document.write(«<CENTER>Oynani uchinchi marta
ochdingiz»);
}
id.document.write(«<FORM><INPUT TYPE=button
onClick=’window.close();’ VALUE=’Oynani
yopish’></CENTER>»);
id.document.close();
}
```

Yangi oynani eski (bosh) oynadan turib yozayotganligimiz uchun yangi obyektga ko‘rsatkich sifatida id o‘zgaruvchining qiymatidan foydalanamiz.

window.setTimeout()

setTimeout() metodi bajarilishi ikkinchi argumentda ko‘rsa-

tilgan millisekundlarcha keyingi qoldiriluvchi yangi hisoblash oqimini ochish uchun foydalaniladi:

idt = setTimeout(«JavaScript_kod»,Time);

Bu funksiyaning tipik qo‘llanilishi — obyektlar xossalarini avtomatik o‘zgarishini tashkil qilish. Masalan, forma maydonida soatni qo‘yish mumkin:

```
var flag=0;
var idp=null;
function myclock()
{
if(flag==1)
{
d = new Date();
window.document.c.f.value = d.getHours()+»:»+d.get
Minutes()+»:»+d.getSeconds();
}
idp=setTimeout(«myclock();»,500);
}
function flagss()
{
if(flag==0) flag=1; else flag=0;
}
...
<FORM NAME=c>
Joriy vaqt:<INPUT NAME=f size=8><INPUT
TYPE=button VALUE=»Start/Stop»
onClick=»flagss();myclock();»>
</FORM>
```

Shuni nazarda tutish kerakki, oqim har doim, hattoki soat to‘xtagan holatda ham paydo bo‘laveradi. Agar u flag o‘zgaruvchisining 1 ga teng qiymatidagina yaratilganida edi, u holda flag ning 0 ga teng qiymatida yo‘qolgan bo‘lar edi va u tugmani bosishda soatlar to‘xtab turishda davom etgan bo‘lardi.

window.clearTimeout

clearTimeout() metodi setTimeout() metodi tufayli hosil qilingan oqimni yo‘q qilishga imkon beradi. Aniqki, uni ishlatilishi hisoblash qurilmalari resurslarini effektivroq taqsimlash imkoniyatini beradi. Bu metodni soatlar bilan bo‘lgan misolda foydalanish uchun biz funksiyani va formani moslashtirishimiz kerak:

```
var idp1 = null;
function start()
{
d = new Date();
window.document.c1.f1.value =
d.getHours()+«:»+d.getMinutes()+«:»+d.getSeconds();
idp1=setTimeout(«start();»,500);
}
function stop()
{
clearTimeout(idp1);idp1=null;
}
...
<FORM NAME=c1>
Joriy vaqt:<INPUT NAME=f1 size=8>
<INPUT TYPE=button VALUE=«Start»
onClick=«if(idp1==null)start();»>
<INPUT TYPE=button VALUE=«Stop»
onClick=«if(idp1!=null)stop();»>
</FORM>
```

Berilgan misolda soatlarni to‘xtatish uchun clearTimeout() metodi ishlatiladi. Bunda ko‘plab oqimlarning paydo bo‘lmasligi uchun oqim obyektiga bo‘lgan ko‘rsatkichning qiymati tekshiriladi.

Document obyekti (window.document). Oynalarni yaratish.

Brauzerda yangi oynalarni yaratish — JavaScriptning juda katta imkoniyati. Siz yangi oynaga yangi hujjatlarni yuklashingiz (masalan, huddi o‘sha HTML hujjatlarini) yoki (dinamik ravishda) yangi materiallarni *yaratishingiz* mumkin. Avval yangi oynani qanday ochish mumkinligini, keyin esa bu oynaga qanday qilib HTML-sahifani yuklash mumkinligini va nihoyat, uni qanday qilib yopish mumkinligini ko‘rib chiqamiz.

Quyida keltirilgan script brauzerning yangi oynasini ochadi va unga qandaydir web-sahifani yuklaydi:

```
<html>
<head>
<script language=«JavaScript»>
<!— hidee
function openWin() {
myWin= open(«abc.html»);
```

```
}
// —>>
</script>
</head>
<body>
<form>
<input type=«button» value=«Yangi oynani ochish»
onClick=«openWin()»>
</form>
</body>
</html>
```

Keltirilgan misolda yangi oynaga open() metodi yordamida *abc.html* sahifasi yoziladi.

Shunki ko‘rishingiz mumkinki, siz oyna yaratish protsessining o‘zi ustidan nazorat o‘rnata olasiz. Masalan, siz yangi oyna status satri, instrumentlar paneli yoki menyuga ega bo‘lishi kerakligini ko‘rsata olasiz. Bundan tashqari, siz oynaning o‘lchamini ham bera olasiz. Masalan, keyingi skriptda 400x300 piksel o‘lchamli oyna ochiladi. U status satriga ham, instrumentlar paneliga ham, menyuga ham ega bo‘lmaydi.

```
<html>
<head>
<script language=«JavaScript»>
<!— hidee
function openWin2() {
myWin= open(«abc.html», «displayWindow»,

«width=400,height=300,status=no,toolbar=no,menubar=no»);
}
// —>>
</script>
</head>
<body>
<form>
<input type=«button» value=«Yangi oynani ochish»
onClick=«openWin2()»>
</form>
</body>
</html>
```

Ko‘rib turganingizdek, oynaning xossalarini quyidagi satrda hosil qilamiz:

«width=400,height=300,status=no,toolbar=no,menubar=no».

Yana shu narsaga e’tibor beringki, siz bu satrda bo‘sh o‘rin simvollarini joylashtirishingiz kerak emas.

Siz boshqara olishingiz mumkin bo‘lgan oyna xossalarining ro‘yhati:

directories	yes|no
height	Piksellar soni
location	yes|no
menubar	yes|no
resizable	yes|no
scrollbars	yes|no
status	yes|no
toolbar	yes|no
width	Piksellar soni

JavaScript tilining 1.2 versiyasida bir nechta yangi xossalar qo‘shildi (ya’ni Netscape Navigator 4.0 da). Siz bu xossalardan Netscape 2.x, 3.x yoki Microsoft Internet Explorer 3.x lar uchun materiallar tayorlashda foydalanmasligingiz kerak, chunki bu brauzerlar 1.2 JavaScript tilini tushinmaydi. Oynalarning yangi xossalari:

alwaysLowered	yes|no
alwaysRaised	yes|no
dependent	yes|no
hotkeys	yes|no
innerWidth	Piksellar soni (width ning o‘rniga)
innerHeight	Piksellar soni (height ning o‘rniga)

outerWidth	Piksellar soni
outerHeight	Piksellar soni
screenX	Piksellar soni
screenY	Piksellar soni
titlebar	yes|no
z-lock	yes|no

Siz bu xossalarning ma'nolarini JavaScript 1.2 tilining bayonidan topishingiz mumkin. Bundan keyin ulardan ayrimlariga tushintirishlar va ishlatishga misollar keltiramiz.

Masalan, endi bu yangi xossalardan foydalangan holda yangi oyna ekrannning qaysi yerida joylashish kerakligini aniqlashingiz mumkin. JavaScript tilining eski versiyasida siz buni amalga oshira olmas edingiz.

Oynaning nomi. Ko'rib turganingizdek, oyna ochish vaqtida biz uchta argumentdan foydalanishimiz kerak:

myWin= open(«abc.html», «displayWindow»,

«width=400,height=300,status=no,toolbar=no,menubar=no»);

Ikkinchi argument nima uchun kerak? Bu oynaning nomi. Oldinroq biz uni target parametrida qanday ishlatilganini ko'rib o'tdik. Shunday qilib agar siz oynaning nomini bilsangiz, u holda unda quyidagi yozuv yordamida yangi sahifani yuklashingiz mumkin

<a href=»abc.html» target=»displayWindow»>

Bunda siz mos oynaning nomini ko'rsatishingiz kerak (agar bunday nomdagi oyna mavjud bo'lmasa, shu nomdagi yangi oyna yaratiladi).

E'tibor beringki, *myWin* — bu yangi oyna nomi emas. Lekin shu o'zgaruvchi yordamidagina siz oynaga kirishingiz mumkin. Va u oddiy o'zgaruvchi bo'lganligidan uning qo'llanish sohasi u aniqlangan skript xolos. Shu bilan birga oyna nomi (berilgan holatda u *displayWindow*) − bu brauzerning istalgan oynasida foydalanish mumkin bo'lgan o'ziga hos identifikator.

Oynalarni yopish. Siz yana JavaScript tili yordamida oynalarni yopishingiz ham mumkin. Buni amalga oshirish uchun sizga close() metodi kerak bo'ladi. Keling, oldinroq ko'rganimiz

kabi yangi oyna ochamiz. Va unga navbatdagi sahifani yuklaymiz:

```
<html>
<script language=»JavaScript»>
<!— hidee
function closeIt() {
  close();
}
// —>>
</script>
<center>
<form>
<input type=button value=»Yopish» onClick=»closeIt()»>
</form>
</center>
</html>
```

Endi agar siz yangi oynadagi tugmani bossangiz, u yopiladi. open() va close() —bu window obyektining metodlaridir. Biz shuni yodda tutishimiz kerakki, oddiygina qilib *open()* va *close()* shaklida emas, balki *window.open()* va *window.close()* ko‘rinishida yozish kerak. Ammo bizning holatda *window* obyektini tushirib qoldirishimiz mumkin — siz agar bu obyektning metodlaridan birini chaqirmoqchi bo‘lsangiz (va u faqatgina shu obyekt uchungina mumkin), u holda siz window prefiksini yozishingiz shart emas.

Hujjatni dinamik ravishda yaratish. Endi biz hujjatlarni dinamik ravishda yaratish kabi JavaScript ning ajoyib imkoniyatini ko‘rib chiqishga tayyormiz. Yani JavaScriptda tuzilgan skript ning o‘ziga yangi HTML-sahifalarni yaratishga ruxsat berish mumkin. Bundan tashqari shu yo‘l bilan webning VRML-ko‘rinishlar va shu kabi boshqa hujjatlarini ham yaratish mumkin. Qulaylik uchun bu hujjatlarni alohida oyna yoki freimga joylashtirish mumkin.

Avval biz oddiy HTML-hujjat yaratamiz va uni yangi oynada namoyish qilamiz.

Quyidagi skriptni ko‘ramiz.

```
<html>
<head>
<script language=»JavaScript»>
```

```
<!— hidee
function openWin3() {
myWin= open(«», «displayWindow»,
«width=500,height=400,status=yes,toolbar=yes,menubar=yes»;
// keyinchalik bosib chiqarish uchun document obyektini ochish
myWin.document.open();
// yangi hujjatni ochish
myWin.document.write(«<html><head><title>Тест»);
myWin.document.write(«</title></head><body>»);
myWin.document.write(«<center><font size=+3>»);
myWin.document.write(«Bu HTML-hujjat avval «);
myWin.document.write(«JavaScript yordamida ochilgan!»);
myWin.document.write(«</font></center>»);
myWin.document.write(«</body></html>»);
// hujjatni yopish - (lekin oynani emas!)
myWin.document.close();
}
// —>>
</script>
</head>
```

```
<body>
<form>
<input type=button value=»Ochish» onClick=»open-
Win3()»>
</form>
</body>
</html>
```

Keling winOpen3() funksiaysini ko‘rib chiqaylik. Aniqki, biz avval brauzerning yangi oynasini ochamiz. Open() funktsiaysining birinchi argumenti — bo‘sh satr (“”), bu esa shuni bildiradiki, biz bu holatda aniq bir URL manzilini ko‘rsatishni hohlamaymiz. Brauzer faqatgina mavjud hujjatni qayta ishlamasligi kerak — JavaScript qo‘shimcha yangi hujjat yaratishi kerak.

Skriptda biz yangi *myWin* o‘zgaruvchini aniqlaymiz. Va uning yordamida biz yangi oynaga kirib bora olamiz. Shunga e’tibor beringki, bu holda bu maqsad uchun oyna nomi (*displayWindow*) dan foydalana olmaymiz.

Yangi oynani ochganimizdan keyin, document obyektini yozish uchun navbat keladi. Bu quyidagi komanda yordamida amalga oshiriladi:

```
// keyinchalik bosmaga chiqarish uchun document obyekti-
ni ochish myWin.document.open();
```

Bu yerda biz open()ga — document obyektining metodiga murojaat qilamiz. Lekin u window obyektining open() metodi bilan bir xil narsa emas! Bu komanda yangi oyna ochmaydi — u bosib chiqarish uchun documentni tayyorlaydi. Bundan tashqari biz *document.open()* ning oldiga *myWin* ni yangi oynaga yozish imkoniyatini olish maqsadida qo‘yishimiz kerak.

Skriptning keyingi satrlarida *document.write()* ni chaqirish yordamida yangi hujjatning matni hosil qilinadi:

```
// yangi hujjatni hosil qilish
myWin.document.write(«<html><head><title>On-the-
fly»);
myWin.document.write(«</title></head><body>»);
myWin.document.write(«<center><font size=+3>»);
myWin.document.write(«Bu HTML-hujjat avval «);
myWin.document.write(«JavaScript yordamida ochilgan!»);
myWin.document.write(«</font></center>»);
myWin.document.write(«</body></html>»);
```

Ko‘rinib turibdiki, biz hujjatga HTML tilining oddiy teglarini yozamiz. Ya’ni amalda biz HTML bo‘laklarini hosil qilamiz! Bunda HTML teglarining istalganidan foydalanish mumkin.

Buni yakunlagandan keyin biz hujjatni yana yopishimiz kerak. Bu quyidagi komanda orqali amalga oshiriladi:

```
// hujjatni yopish - (oynani emas!)
myWin.document.close();
```

Nafaqat hujjatlarni dinamik ravishda yaratish mumkin, balki ularni o‘z xohishicha u yoki bu freimlarga joylashtirish ham mumkin. Masalan, *frame1* va *frame2* nomli ikki framega ega bo‘lsak va f*rame2* ga yangi hujjatni hosil qilish kerak bo‘lsa, u holda *frame1*ga quyidagini yozib qo‘yish yetarli:

```
parent.frame2.document.open();
parent.frame2.document.write(«Buerda sizning HTML-kodingiz joylashadi»);
parent.frame2.document.close();
```

Ikkilamchi obyektlar. JavaScriptda ichki qurilgan obyektlarni ko‘rib chiqamiz. Foydalanuvchi tomonidan yaratilgan obyektlar va brauzerning obyektli modelini (bu model to‘g‘risida boshqa bir bo‘limda hikoya qilinadi) tashkil qilgan obyektlardan farq qilib, ichki qurilgan obyektlar xohlagan kontekstda, xoh u Microsoft Internet Explorer yoki Netscape Navigator bo‘lsin — chaqirilishi mumkin.

JavaScript tiliga bo‘lgan asosiy talablarni belgilab beruvchi ECMAScript (ECMAScript Language Specification, Standard ECMA-262 ni qarang) spetsifikatsiyasiga ko‘ra tilda quyidagi obyektlar amalga oshirilgan bo‘lishi kerak: Global, Object, Function, Array, String, Boolean, Number, Math va Date.

Array, Boolean, Date, Function, Math, Number va String ichki qurilgan obyektlarini ko‘rib chiqamiz.

Array obyekti

JavaScript tilida massivlarni yaratish uchun ichki qurilgan ma’lumotlar tipi yo‘q, shuning uchun bunday masalalarni hal qilishda Array obyektidan foydalaniladi. U massivlarni birlashtirish, tartiblash va o‘rin almashtirish uchun metodlarga ega, yana massivning o‘lchamini aniqlash imkoniyati ham mavjud.

Massiv — bu nomi va tartib raqami (indeksi) bo‘yicha chaqiriladigan qiymatlarning tartiblashgan to‘plamidir. Masalan, dasturda har biri o‘z tartib raqamiga ega bo‘lgan xabarlar

to‘plami — allMsg dan tuzilgan massivni yaratish mumkin. Shunday qilib, allMsg[0] birinchi xabar, allMsg[1] — ikkinchi xabar bo‘ladi va hokazo.

Array obyektini hosil qilish uchun ikki bir birini o‘rnini olishi mumkin bo‘lgan usulni qo‘llash mumkin.

New konstruktorini chaqiring va massiv o‘lchami (undagi elementlar soni)ni bering. Massivni to‘ldirish keyinroq ro‘y beradi.

Quyidagi misolni ko‘rib chiqamiz:

```
<html>
<head><title> JavaScript testi.</title>
<script language=«JavaScript»>
// yangi massivni yaratish
allStr = new Array(5);
// massivni tuldirish
allStr[0] = «Xabar №1»;
allStr[1] = «Xabar №2»;
allStr[2] = «Xabar №3»;
allStr[3] = «Xabar №4»;
allStr[4] = «Xabar №5»;
// massiv elementlarini ko‘rsatish funksiyasi
function showMsg(ndx)
{
alert(allStr[ndx]);
}
</script>
</head>
<— HHujjat yuklanganda Xabar №4 ni chiqarish—>>
<body onLoad=«showMsg(3);»>
</body>
</html>
```

Yuqorida keltirilgan misolda 5 ta elementdan tashkil topgan massiv yaratiladi, keyin u to‘ldiriladi.

New konstruktorini chaqirasiz va massivning barcha elementlarining qiymatlarini berasiz. Massiv o‘lchami bu holda oshkora holda ko‘rsatilmaydi.

Quyidagi misolni ko‘ramiz.

```
<html>
<head><title> JavaScript testi.</title>
```

```
<script language=«JavaScript»>
// yangi massivni yaratish va to‘ldirish
allStr = new Array(«Xabar №1», «Xabar №2», «Xabar №3»,
«Xabar №4», « Xabar №5»);
// massiv elementlarini chiqarish
function showMsg(ndx)
{
alert(allStr[ndx]);
}
</script>
</head>
<— Hujjat yuklanganda Xabar №4 ni chiqarish —>->
<body onLoad=«showMsg(3);»>
</body>
</html>
```

Bu yerda massiv elementlarining qiymatlari new konstruktorini chaqirish vaqtidayoq bevosita beriladi.

Array obyektining metodlari

Array obyekti quyidagi metodlarga ega.

Metod	Bayoni
join	Massivning barcha elementlarini satrga birlashtiradi
reverse	Massivdagi elementlar tartibini o'zgartiradi — birinchi element oxirgi elementga, oxirgisi esa birinchi elementga aylanadi
sort	Massiv elementlarini tartiblaydi

Array obyekti metodlarining ishlatilishlarini ko‘rib chiqamiz. Aytaylik bir nechta elementga ega bo‘lgan massiv va ikkita — showAll va showElement funksiyalari berilgan bo‘lsin. Birinchi massivning barcha elementlarini namoyish qiluvchi funksiya sikl ichida massivning berilgan elementini namoyish qiluvchi showElement funksiyasini chaqiradi:

```
<html>
<head><title>JavaScript testi.</title>
<script language=»JavaScript»>
```

```
myArray = new Array(«Ota», «Ona», «Aka», «Singil», «Do‘st»);
function showElement(ndx)
{
alert(myArray[ndx]);
}
function showAll()
{
for( i = 0; i <= myArray.length-1; i++)
{
showElement(i);
}
}
</script>
</head>
<body onLoad=»showAll();»>
</body>
</html>
```

Agar yuqorida keltirilgan faylni yuklasak, har birida myArray massivining bitta elementi — Mother, Father, Sister, Brother va Uncle tasvirlangan xabarlar paneli ketma-ketligini ko‘rishimiz mumkin.

Quyidagi kod yozilgan test funksiaysini yaratamiz:

```
function test()
{
alert(myArray.join());
}
```

Va <body> tegini o‘zgartiramiz:

```
<body onLoad=»test();»>
```

Joint metodi massiv elemetlarini ajratib turuvchini berish mumkin bo‘lgan majburiy bo‘lmagan parametrga ega. Ko‘rsatilmagan holda “,” belgisi ishlatiladi. Masalan:

```
function test()
{
alert(myArray.join(« _|_ «));
}
```

reverse metodi massiv elementlarini o‘rin almashtirishlari uchun foydalaniladi. test funksiaysiga shu metodni chaqirilishini qo‘shamiz:

```
function test()
{
myArray.reverse();
alert(myArray.join(«;»));
}
```

Massivning birinchi elementi oxirgi o‘rinni egalladi, ikkinchisi — oxiridan oldingi va hokazo.

Sort metodi massiv elementlarini tartiblash uchun foydalaniladi. Test funksiyasiga shu metodni chaqirilishini qo‘shamiz:

```
function test()
{
myArray.sort();
alert(myArray.join(«;»));
}
```

Ko‘p o‘lchamli massivlarni yaratish.

Array obyekti ko‘p o‘lchamli massivlarni yaratishga imkon beradi. Quyida ko‘p o‘lchamli massivni yaratishga misol keltirilgan.

```
<html>
<head><title> JavaScript testi. </title></head>
<body>
<center>
<font size=5><b>Ko‘p o‘lchovli massiv </b></font><p>
<script language=«JavaScript»>
a = new Array(4);
for( i=0; i < 4; i++)
{
a[i] = new Array(4);
for( j=0; j < 4; j++)
{
a[i][j] = «[«+i+»,»+j+»]»;
}
}
for( i=0; i < 4; i++)
{
str = «Satr «+i+»:»;
for( j=0; j < 4; j++)
{
```

```
str += a[i][j];
}
document.write(str, «<br>»);
}
</script>
</center>
</body>
</html>
```

Yuqorida keltirilgan dasturda to‘rtta elementdan tashkil topgan massiv hosil qilinadi: uning elementlari har biri ham to‘rtta elementdan iborat massivdir. Har bir elementga element indeksini beruvchi i, j juftlikning qiymatlari kiritiladi. Keyin sikl ichida berilgan massiv elementlari namoyish qilinadi.

Boolean obyekti

Ichki qurilgan Boolean obyekti mantiqiy bo‘magan qiymatlarni mantiqiy qiymatlarga aylantirish uchun foydalaniladi. Boolean obyektining namunasini hosil qilish uchun new konstruktori ishlatiladi**:**

```
bfalse = new Boolean(false);
btrue  = new Boolean(true);
```

valueOf metodi mantiqiy o‘zgaruvchining mantiqiy ko‘rinishdagi qiymatini ko‘rsatadi, toString metodi esa satrli ko‘rinishdagi qiymatini ko‘rsatadi. Bu metodlarning ishlatilishiga misol quyida keltirilgan.

```
<html>
<head><title> JavaScript testi.</title></head>
<body>
<script language=«JavaScript»>
// ikki mantiqiy o‘zgaruvchi yaratamiz
bfalse = new Boolean(false);
btru = new Boolean(true);
// ularning qiymatlarini chiqaramiz (mantiqiy qiymatlar)
document.write(bfalse.valueOf()+«<br>»);
document.write(btrue.valueOf()+«<br>»);
// satrli qiymatlarni chiqaramiz
document.write(bfalse.toString()+«<br>»);
document.write(btrue.toString()+«<br>»);
</script>
</body>
</html>
```

Date obyekti.

Date obyekti va uning metodlaridan skript dasturlarda sana va vaqt bilan ishlashda foydalaniladi. Bu obyekt sanani o‘rnatish, uning qiymatini olish va turli xil almashtirishlarni bajarish uchun katta sondagi metodlar to‘plamiga ega. Date obyekti xossalarga ega emas.

Shuni ta’kidlab o‘tamizki, JavaScriptda sana huddi Javadagi singari saqlanadi — u 1-yanvar 1970-yildan buyon o‘tgan millisekundlar sonini ko‘rsatadi. Shunday qilib, undan oldingi sanalar qo‘llanmaydi.

Date obyektining namunasini hosil qilish uchun Date konstruktoridan foydalaniladi:

MyDate = new Date([parametrlar]);

Quyidagi parametrlarni ko‘rsatish mumkin:

- parametrlarsiz — namuna joriy sana va vaqtni ko‘rsatadi. Masalan, today = new Date();
- sanani quyidagi formatda ko‘rsatuvchi satr: «Oy, kun, yil vaqt: minutlar: sekundlar». Masalan, someDate = new Date(«May 15, 1996»). Agar soat, minut yoki sekundlar soni ko‘rsatilmagan bo‘lsa, ularning qiymatlari 0 ga teng deb olinadi;
- yil, oy va kunlarning butun sondagi qiymatlari to‘plami. Masalan, otherDay = new Date (96, 4, 15);
- yil, oy, kun, soat, minut va sekundlarning butun sondagi qiymatlari to‘plami. Masalan, sameDay = new Date(96, 4, 15, 15, 30, 0);

Turli xil parametrlarning ishlatilishiga oid misol quyida keltirilgan.

```
<html>
<head><title> JavaScript testi.</title></head>
<body>
<center>
<script language=«JavaScript»>
today = new Date();
document.write(«today=»+today+«<br>»);
someDate = new Date («May 16, 1996»);
document.write («someDate=»+someDate+»<br>»);
otherDay = new Date( 96, 4, 15);
document.write («otherDay=»+otherDay+«<br>»);
sameDay = new Date (96, 4, 16, 15, 30, 0);
```

```
document.write («sameDay=»+sameDay+«<br>»);
</script>
</center>
</body>
</html>
```

Date obyektining metodlari.

Date obyekti tomonidan sana va vaqtni boshqarish uchun ishlatiladigan metodlarni quyidagi kategoriyalarga bo‘lish mumkin:

- *o‘rnatish metodlari (set)* – Date obyekti namunalarida sana va vaqtni o‘rnatish uchun metodlar;
- *aniqlash metodlari (get)* – Date obyekti namunalaridan sana va vaqtni olish uchun metodlar;
- *shakl almashtirish metodlari (to)* – sana va vaqtni satrga aylantirish uchun metodlar;
- *sanani qayta ishlash uchun metodlar.*

O‘rnatish va *aniqlash* metodlari sekund, minut, soat, oy kuni, hafta kuni, oy va yil qiymatlarini olish, o‘zgartirish uchun foydalanilish mumkin. Masalan, hafta kunini aniqlash mumkin bo‘lgan getDay() metodi mavjud, lekin setDay() metodi mavjud emas, chunki hafta kuni avtomatik ravishda o‘rnatiladi.

Bu metodlarning barchasi quyidagi jadvalda keltirilgan va butun sonli qiymatlardan foydalanadi.

Qiymati	Oralig‘i
Minut va sekundlar soni	0...59
Soatlar soni	0...23
Hafta kuni	0...6
Sana	1...31
Oy	0...11 (Yanvar...Dekabr)
Yil	1900-yildan boshlab

Quyidagi misolni ko‘ramiz. Aytaylik, siz sanani 15-May 1996-yil deb berdingiz. Quyida Date obyektining bir nechta metodlarining ishlatilishi namoyish qilingan.

```
<html>
<head><title>JavaScript </title></head>
<body>
<center>
<p>
<script language=«JavaScript»>
someDate = new Date («May 15, 1996»);
document.write(«someDate=»+someDate+«<br>»);
document.write(«getDay  =»+someDate.getDay()+«<br>»);

document.write(«getMonth=»+someDate.getMonth()+«<br>»;
document.write(«getYear  =»+someDate.getYear()+
«<br>»);
</script>
</center>
</body>
</html>
```

getTime va setTime metodlari sanalarni taqqoslash uchun qulay. getTime metodi millisekundlar sonini beradi. Masalan, bu yildagi qolgan kunlar sonini qanday aniqlash quyida keltirilgan.

```
<html>
<head><title>JavaScript </title></head>
<body>
<center>
<br><br><br>
<script language=«JavaScript»>
today = new Date();
// sanani berish
endYear = new Date(«December 31, 1990»);
// yilni o‘zgartirish
endYear.setYear(today.getYear());
// kundagi millisekundlar sonini hisoblash
msPerDay = 24 * 60 * 60 * 1000;
// kunlar sonini olish
daysLeft = (endYear.getTime() — today.getTime()) /
msPerDay;
```

```
// yahlitlash
daysLeft = Math.round(daysLeft);
// ko‘rsatish
document.write(«Number of days left in the year:«
+daysLeft);
</script>
</center>
</body>
</html>
```

Parse() metodi vaqtning satrli tasvirini shakl almashtirish uchun foydalanilishi mumkin. Masalan,

```
someDate = new Date();
someDate.setTime(Date.parse(«May 15, 1996»));
```

Date obyektini ko‘rishni kichkina bir amaliy misol bilan yakunlaymiz. Biz joriy vaqtni ekranga chiqaruvchi dasturni yaratamiz — xohlasangiz uni siz o‘z HTML-sahifangizga joylashtirishingiz mumkin. Vaqt “tugma” interfeis elementining sarlavhasi sifatida chiqariladi. Mashq sifatida dasturni shunday to‘ldirishni taklif qilinadiki, tugmani bosishingiz bilan joriy sana namoyish qilinsin.

Vaqtni namoyish qiluvchi dastur matni quyida keltirilgan.

```
<html>
<head><title>JavaScript </title>
<script language=«JavaScript»>
function showTime()
{
// joriy vaqtni olamiz
var time = new Date();
// soatlar sonini bilib olamiz
var hour = time.getHours();
// minutlar sonini bilib olamiz
var minute = time.getMinutes();
// sekundlar sonini bilib olamiz
var second = time.getSeconds();
// soatlarni ko‘rsatamiz
var temp = hour;
// minutlarni ko‘rsatamiz (0 ni  0—9ga qo‘shib)
```

```
temp += ((minute < 10) ? «:0» : «:») + minute;
// sekundlarni ko‘rsatamiz (0 ni  0-9ga qo‘shib)
temp += ((second < 10) ? «:0» : «:») + second;
// namoyish qilamiz
document.forms[0].clock.value = temp;
// har bir sekundni ko‘rsatamiz
id = setTimeout(«showTime()», 1000);
}
</script>
</head>
<body onLoad=«showTime();»>
<center>
<font size=5><b>
 JavaScript Date & Time Demo
</b></font>
<form name=«DateDemo»>
<input type=«button» name=«clock» value=«xxxxxxxx»>
</form>
</center>
</body>
</html>
```

Sanani namoyish qiladigan funktsiya quyidagi ko‘rinishda bo‘lishi mumkin:

```
function showDate()
{
//joriy sanani olamiz
var D = new Date();
// hafta kunini bilib olamiz
var DOW = D.getDay();
// sanani bilib olamiz
var Day = D.getDate();
// oyni bilib olamiz
var Month = D.getMonth();
// yilni bilib olamiz
var Year = D.getYear();
// namoyish qilamiz
temp = Day + «/» + Month + «/» + Year;
document.forms[0].clock.value = temp;
}
```

DOW o‘zgaruvchisida hafta kunining nomeri saqlanadi.

Yuqorida ko‘rib o‘tilgan Array obyektidan foydalangan holda, hafta kunlari nomlarining massivini hosil qilishimiz mumkin:

```
dayNames = new
Array(«yakshanba», «dushanba», «seshanba», «chrshanba»,
«payshanba», «juma», «shanba»);
```

Va showDate() funksiyasining so‘nggi satrlarida quyidagilarni qo‘shish:

```
temp = Day + «/» + Month + «/» + Year +
dayNames[DOW];
```

Function obyekti.

Ichki qurilgan *Function* obyekti satrga funksiya deb baholash mumkin bo‘lgan JavaScriptdagi kodni berishga imkon beradi. *Function* obyektining namunasini yaratish uchun *new* konstruktoridan foydalaniladi:

```
var newBgColor = new Function(«c», «document.
bgColor=c»);
```

Bu funksiyadan foydalanishga misol quyida berilgan.

```
<html>
<head><title>JavaScript </title></head>
<body>
<center>
<script language=»JavaScript»>
var newBgColor = new Function(«c», «document.-
bgColor=c»);
function fnDemo()
{
newBgColor(«#a7b7c7»);
}
</script>
<form>
<input type=«button» value=«bgColor» onClick=«fn-
Demo();»>
</form>
</center>
</body>
</html>
```

Yana bir misolni ko‘rib chiqamiz. Aytaylik, bizga ikki argumentning qiymatlarini ko‘paytiradigan funksiya kerak bo‘lsin. Bu funksiyani myMult deb ataymiz:

```
var myMult = new Function(«x», «y», «return x*y»);
```

Undan quyidagicha foydalanish mumkin:

```
<html>
<head><title>JavaScript </title></head>
<body>
<center>
<script language=«JavaScript»>
var myMult = new Function(«x», «y», «return x*y»);
document.write(«10*20=»+myMult(10, 20));
</script>
</center>
</body>
</html>
```

Biz, hattoki, kichkina forma hosil qilishimiz va uni turli xil sonlarning ko‘paytmasini topish uchun foydalanishimiz mumkin:

```
<html>
<head><title>JavaScript 12.97</title></head>
<body>
<center>
<script language=«JavaScript»>
var myMult = new Function(«x», «y», «return x*y»);
function calc()
{
document.forms[0].sum.value =
myMult(document.forms[0].x.value, document.forms [0].-
y.value);
}
</script>
</center>
<form>
<input type=«text» name=«x»   value=10  size=4>
<input type=«text» name=«y»   value=10  size=4>
<input type=«text» name=«mul» value=«...» size=4>
<input type=«button» value=«calc» onClick=«calc();»>
</form>
</body>
```

```
</html>
```

Math obyekti.

Ichki qurilgan *Math* obyekti turli hil matematik konstantalarni olish va matematik funktsiyalarni bajarish uchun hossa va metodlarga ega. Quyida shu qiymatlarni ekranga chiqaradigan dasturning matni berilgan.

```
<html>
<head><title>JavaScript </title></head>
<body bgcolor=»#a7b7c7»>
<center><font size=4><b>Math</b> Object Properties
</font><p>
<script language=«JavaScript»>
s = «<TABLE BORDER=2>» +
«<TR><TD><B>E</B></TD><TD>» + Math.E +
«</TD></TR>» +
«<TR><TD><B>LN2</B></TD><TD>» + Math.LN2
+ «</TD></TR>» +
«<TR><TD><B>LN10</B></TD><TD>» + Math.LN10
+ «</TD></TR>» +
«<TR><TD><B>LOG2E</B></TD><TD>» + Math.-
LOG2E + «</TD></TR>» +
«<TR><TD><B>LOG10E</B></TD><TD>» + Math.-
LOG10E + «</TD></TR>» +
«<TR><TD><B>PI</B></TD><TD>» + Math.PI +
«</TD></TR>» +
«<TR><TD><B>SQRT1_2</B></TD><TD>» + Math.-
SQRT1_2 + «</TD></TR>» +
«<TR><TD><B>SQRT2</B></TD><TD>» + Math.-
SQRT2 + «</TD></TR>» +
«</TABLE>»;
document.write(s);
</script>
</center>
</body>
</html>
```

Keyingi jadvalda *Math* obyektida mavjud bo‘lgan metodlar sanab o‘tilgan.

Metod nomi	Bayoni
abs	Argument qiymatining modulini beradi
sin, cos, tan	Standart trigonometrik funksiyalar, argumentlar radianda beriladi
acos, asin, atan	Teskari trigonometrik funksiyalar, radiandagi qiymatlarni beradi
exp, log	Eksponenta va natural logarifm, asosi — e.
ceil	Argumentga teng yoki undan katta bo‘lgan butun sonni beradi
floor	Argumentga teng yoki undan kichik butun sonni beradi
min, max	Ikki argumentlardan kichigi yoki kattasini beradi
pow	Argument darajasini beradi
round	Argumentni unga eng yaqin butun songacha yaxlitlaydi
sqrt	Argumentning kvadrat ildizini beradi

Math obyektining ko‘plab sondagi xossa va metodlaridan foydalanganda, masalan, qandaydir hisoblashlarni bajarganda, *with* operatoridan foydalanish qulay:

```
with( Math )
{
a = PI * r*r;
b = floor(c);
x = r * sin( theta );
y = r * cos( theta );
z = round( b );
}
```

Number obyekti.

Ichki qurilgan *Number* obyektining xossalari maksimal qiymat, “not-a-number” (NaN) va cheksizliklar kabi tipdagi turli

xil sonli o‘zgarmaslarning qiymatlarini olish uchun foydalaniladi. Quyida *Number* obyekti xossalari qiymatlarini chiqaradigan dasturning matni berilgan.

```
<html>
<head><title>JavaScript </title></head>
<body bgcolor=»#a7b7c7»>
<center><font size=4><b>Number</b> Object Properties
</font><p>
<script language=«JavaScript»>
s = «<TABLE BORDER=2>» +
«<TR><TD><B>MAX_VALUE</B></TD><TD>» +
Number.MAX_VALUE + «</TD></TR>» +
«<TR><TD><B>MIN_VALUE</B></TD><TD>» +
Number.MIN_VALUE + «</TD></TR>» +
«<TR><TD><B>NaN</B></TD><TD>» +
Number.NaN + «</TD></TR>» +
«<TR><TD><B>NEGATIVE_INFINITY</B></TD><T
D>» +
Number.NEGATIVE_INFINITY + «</TD></TR>» +
«<TR><TD><B>POSITIVE_INFINITY</B></TD><TD
>» +
Number.POSITIVE_INFINITY + «</TD></TR>» +
«</TABLE>»;
document.write(s);
</script>
</center>
</body>
</html>
```

String obyekti

JavaScript tilida *String* ko‘rinishidagi ichki qurilgan ma’lumotlar tipi yo‘q. Shunga qaramasdan *String* obyekti va uning metodlaridan foydalangan holda siz skript dasturlarida satrlar bilan ishlashingiz mumkin. *String* obyekti satrlar ustida bajariladigan juda ko‘plab metodlarga va satrning uzunligini aniqlashga imkon beruvchi bitta xossa (*length*)ga ega.

String obyektining namunasi *new* konstruktorini chaqirish orqali hosil qilinadi:

```
myString = new String(«myString Object»);
```

String obyekti ikki xil tipdagi metodlarni namoyish qiladi. Birinchi tipga shakl almashtirishlar yoki satrlar ustida boshqa operatsiyalarni bajaradigan metodlar kiradi: *substring*, *toUpperCase* va h.k. shuni ta'kidlab o'tamizki, HTML-versiyasi satrini beruvchi funksiyalar ECMAScript standartiga kirmaydi.

String obyekti metodlari quyidagi jadvalda sanab o'tilgan.

Metod	Bayoni
anchor	anchor-HTML-elementini yaratadi
big, blink, bold, fixed, italics, small, strike, sub, sup	Mos HTML-elementini yaratadi
charAt	Satrning ko'rsatilgan o'rnida joylashgan belgini beradi
indexOf, lastIndexOf	Mos ravishda satrdagi qism-satrning o'rnini yoki satrdagi ohirgi qism-satrning o'rnini beradi
link	HTML-o'tish(boglanish)ni yaratadi
split	String obyektini satrlarni qism-satrlarga aylantirgan holda satrlar massiviga bo'ladi
substring	Ko'rsatilgan qism-satrni beradi
toLowerCase, toUpperCase	Satrdagi belgilarni mos ravishda quyi va yuqori registrdagi belgilarga aylantiradi

Bu obyekt metodlaridan foydalanishga bir nechta misollar ko'rib o'tamiz.

```
<html>
<head><title>JavaScript </title></head>
<body bgcolor=«#a7b7c7»>
<center>
<script language=«JavaScript»>
//yangi satrni yaratamiz
var s = new String('Netscape Navigator');
//qism-satrni olamiz
```

```
var n = s.substring(0,8);
//satr va qism satrni ko‘rsatamiz
document.write(«string=»+s+«<br>substring=»+n);
//quyi registrga aylantiramiz
var l = s.toLowerCase(s);
// ko‘rsatamiz
document.write(«<br>»+l);
// yuqori registrga aylantiramiz va ko‘rsatamiz
document.write(«<br>»+s.toUpperCase(s));
</script>
</center>
</body>
</html>
```

Bular shakl almashtirish metodlari edi. Endi HTML-versiyalar satrlarini beruvchi metodlarni ko‘rib chiqamiz:

```
<html>
<head><title>JavaScript </title></head>
<body bgcolor=«#a7b7c7»>
<center>
<script language=«JavaScript»>
// yangi satr yaratamiz
var s = new String(‘Netscape Navigator’);
//  <b> va </b> larni qo‘shamiz
document.write(s.bold(s)+«<br>»);
//  <i> va </i> larni qo‘shamiz
document.write(s.italics(s)+«<br>»);
// <tt>  va </tt> larni qo‘shamiz
document.write(s.fixed(s));
// <sub>  va  </sub> larni qo‘shamiz
document.write(s.sub(s)+«...»);
// <sup> va  </sup> larni qo‘shamiz
document.write(s.sup(s)+«<br>»);
//  <strike>  va  </strike> larni qo‘shamiz
document.write(s.strike(s)+«<br>»);
</script>
</center>
</body>
</html>
```

Satrni o‘tishga aylantirish uchun link metodini chaqirish kerak:

```
var s = new String(‘Netscape Navigator’);
document.write(s.link(«http://www.netscape.com»));
```

s o‘zgaruvchi uchun hujjatda HTML-matn quyidagicha ko‘rinishga ega bo‘ladi:

```
<A HREF=«http://www.netscape.com»>Netscape Navigator</A>
```

ADABIYOTLAR RO‘YXATI

1. *Гради Буч.* Объектно-ориентированной анализ и проектирование с примерами приложений на C++. Невский диалект, 560 стр, 2001 г.
2. *Грехем И.* Объектно-ориентированные методы. Принципы и практика. Вильямс. 879 стр, 2004 г.
3. *Иванова Г.С.* Объектно-ориентированное программирование. Учебник. МГТУ им Баумана. 320 стр, 2003 г.
4. *Фаулер М., Скотт К.* UML в кратком изложении. Применение стандартного языка объектного моделирования. М., Мир, 1999.
5. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML: руководство пользователя. М., ДМК, 2000.
6. *Пол Айра.* Объектно-ориентированное программирование на C++. Второе издание. М.: Бином, 1999.
7. *Подбельский В.В.* Язык C++ М.: Финансы и статистика, 1996.
8. *Страуструп Б.* Язык программирования C++. Третье издание, М.: Бином, 1999.
9. *Либерти Д.* Освой самостоятельно C++: 10 минут на урок. Пер с англ. Вильямс, 374 стр,2004 г.
10. *Шмидский Я.К.* Прораммирование на языке C++: Самоучитель. Учебное пособие. Диалектика. 361 стр, 2004 г.
11. *Крупник А.Б.* Изучаем C++. Питер. 251 стр, 2003 г.
12. *Мейерс С.* Наиболее эффективное использование C++. 35 новых рекомендаций. ДМК-Пресс, 304 стр, 2000 г.
13. *Фейсон Т.* Объектно-ориентированное программирование на C++ 4.5. Киев: Диалектика, 1996.
14. *Шилдт Г.* Самоучитель C++. Второе издание. СПб.: BHV, 1998.
15. *Шилдт Г.* Теория и практика C++. СПб.: BHV, 1996.
16. *Элджер Дж.* C++: библиотека программиста СПб: Питер, 1999.
17. *Николаенко Д.В.* Самоучитель по Visual C++.Спб, 2001.
18. *Киммел П.* Borland C++5. СПб.: BHV, 1997.
19. *Грегори К.* Использование Visual C++6. М., Вильямс, 1999.

20. *Крейг Арнуш.* Borland C++: освой самостоятельно М.: Бином, 1997.

21. *Лейнекер Р.* Энциклопедия Visual C++6. Спб, Питер, 1999.

22. *Секунов Н.Ю.* Самоучитель Visual C++6. Спб, BHV, 1999.

23. *Паппас К., Мюррей У.* Visual C++6: Руководство разработчика. Киев: BHV, 2000.

24. *Ш. Пауэрс.* Динамический HTML, 1999. 384 с.

25. *Вайк А., Джиллиам Дж.* JavaScript: Полное руководство: Перевод с английского. 2004. 719 с.

26. *Лещев Д.* Создание интерактивного web-сайта: Учебный курс. 2003.27. Дронов В. JavaScript в Web-дизайне. СПб-Петербург, 2002. 880 с.

28. *Джейсон Кренфорд.* Теги DHTML и CSS для Internet. 2005. 520 с.

29. *Гаевский А.Ю., Романовский В.А.* Самоучитель по созданию Web-страниц.

30. *Матросов, Сергеев, Чаунин.* HTML 4.0 в подлиннике. BHV, СПб, 2000, 672 с.

31. *Уилтон П.* JAVASCRIPT. Основы. Символ-плюс. 2002. 1056 с.

32. *Кингли-Хью Э., Кингли-Хью К.* JAVASCRIPT 1.5: Учебный курс. Питер. 1-е издание. 2002.

33. *Бранденбау.* JAVASCRIPT. Сборник рецептов для профессионалов. СПб. 2001.

SH. NAZIROV, R. QOBULOV, F. NURALIYEV

OBYEKTGA MO‘LJALLANGAN DASTURLASH

WEB-SAHIFALARNI DASTURLASH

Axborot-kommunikatsiya texnologiyalari sohasidagi kasb-hunar kollejlarining «Axborot-kommunikatsiya tizimlari (3521916)» mutaxassisligi o‘quvchilari uchun o‘quv qo‘llanma

«Sharq» nashriyot-matbaa
aksiyadorlik kompaniyasi
Bosh tahririyati
Toshkent — 2007

Muharrir *Akbar Bahromov*
Badiiy muharrir *Tolib Qanoatov*
Texnik muharrir *Diana Gabdraxmanova*
Sahifalovchi *Tatyana Ogay*

Bosishga ruxsat etildi 28.09.2007. Bichimi 60x90$^{1}/_{16}$. «Tayms» garniturasi. Ofset bosma. Shartli bosma tobog‘i 8,5. Nashriyot-hisob tobog‘i 8,0. Adadi 800 nusxa. Buyurtma № 3878. Bahosi kelishilgan narxda.

**«Sharq» nashriyot-matbaa
aksiyadorlik kompaniyasi bosmaxonasi,
100083, Toshkent shahri, Buyuk Turon, 41.**