Array (masyvo) tipai

● Array tipai apibrėžia sutvarkytą vienatipių elementų masyvą.

● Masyvo elementais gali būti visi pirminiai tipai ir visi programuotojo apibrėžti tipai.

● Galima apibrėžti masyvą masyvų, masyvą masyvų masyvą ir taip toliau.

● Masyvo elementai pasiekiami naudojant indeksą, ar indeksus

● Masyvo pirmo elemento indeksas 0

● Jei galime sukurti kokį nors tipą, tai galime sukurti ir masyvą su to tipo elementais

draugai[draugai.length - 1])

testArray[i] = Integer.parseInt(stringValue);

}

System.out.println(Arrays.toString(testArray));

Visi metodai privalo nurodyti grįžties tipą. Jei nereikia grąžinti nieko, privalote nurodyti void grįžties tipą. void tipo metodai analogiški Paskalio procedūroms. Jei metodas nėra void tipo, jame turi būti return operatorius, nurodantis, kokį

kintamąjį grąžinti. Pateiksime ir vienos, ir kitos metodų rūšių pavyzdžių.

Visi pirminiai kintamieji perduodami savo reikšme:

○ prieš darant perdavimą, yra pasidaroma kintamojo kopija ir perduodama kopijos reikšmė

○ jei metodo viduje yra keičiama parametro reikšmė, tai po grįžtiesi parametro reikšmė nepakinta

**○ metodo viduje yra manipuliuojama su kintamojo kopija**

**String** tipo klasės objektas apibrėžia eilutės simbolių seką

● String klasė yra **java.lang** pakete, kuris nereikalauja jo importavimo sakinio.

● Kaip ir kitos klasės, String turi aibę konstruktorių ir metodų.

● Skirtingai nuo kitų klasių, String objektams galima taikyti dvi operacijas: + ir +=,

kurios apjungia eilutes.

//kviečiame tiesioginės eilutės metodą

char pirmojiRaidė = “Mindaugas”.charAt(0);

//kviečiame String kintamojo metodą

char pirmasisSimbolis = vardas.charAt(0);

**\**

● Kartą sukurta, eilutė negali būti pakeista

● Objektai pasižymintys tokia savybe vadinami **nekintamais (immutable).**

● Nekintami objektai patogūs tuo, kad visos nuorodos į juos yra saugios, nes nėra pavojaus, kad objekto turinys pakito ir skirtingos nuorodos žymi skirtingo turinio objektą.

String atvejis išskirtinis, nes inicijavimas atliekamas nenaudojant new operatoriaus. Tai padaryta sąmoningai, nes eilutės yra vienas dažniausiai naudojamų bet kokioje programavimo kalboje elementų ir norisi, kad darbas su eilutėmis būtų kuo paprastesnis.

String objektai yra nekintantys: visi metodai atliekantys manipuliacijas su tekstu, grąžina naują String objektą.

String zodis = “java";

char ch = Character.toUpperCase(zodis.charAt (0));

zodis = ch + zodis.substring (1);Java

● Tuščia eilutė neturi nei vieno simbolio; jos ilgis lygus 0.

Tuščia eilutė skiriasi nuo neinicijuotos:

char[] raides = {‘J’, ‘a’, ‘v’, ‘a’};

String zodis = new String(raides);//”Java”

int length();

char charAt(i);

● Grąžina kiek eilutėje yra simbolių

● Grąžina simbolį esantį i-ojoje

pozicijoje.

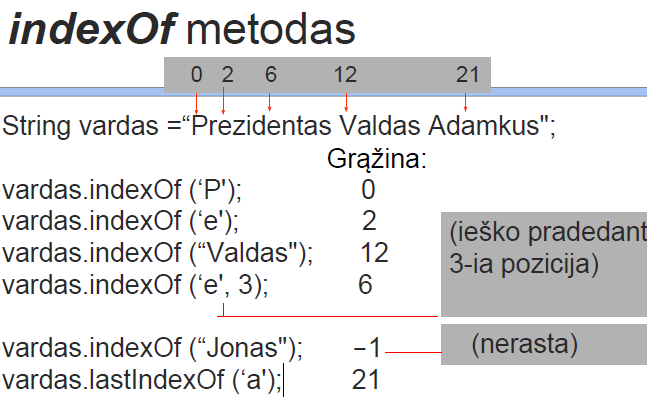
● String subs = zodis.substring (i, k);

● grąžina eilutę sudarytą iš simboliųesančių nuo i iki k-1 pozicijos

● String subs = zodis.substring (i);

○ grąžina eilutę pradedant nuo i-osios pozicijos iki galo.

String result = zodis1.**concat** (zodis2);



boolean b = zodis1.**equals**(zodis2);

grąžina true jei zodis1 sutampa su zodis2

boolean b = zodis1.**equalsIgnoreCase**(zodis2);

grąžina true jei zodis1 sutampa su zodis2

int diff = zodis1.**compareTo**(zodis2);

grąžina “skirtumą” zodis1 - zodis2

int diff = zodis1.**compareToIgnoreCase**(zodis2);

grąžina “skirtumą” zodis1 - zodis2,

ignoruojant didžiųjų/mažųjų skirtumą

Dažnai konkreti “skirtumo” zodis1 - zodis2 reikšmė nenaudojama, o tik skirtumo ženklas. Jei „skirtumas“ neigiamas, zodis1 eina prieš zodis2, lygus nuliui - zodis1 ir zodis2 sutampa, teigiamas - zodis1 eina po zodis2. Pvz.

String zodis2 = zodis**1.trim** ();

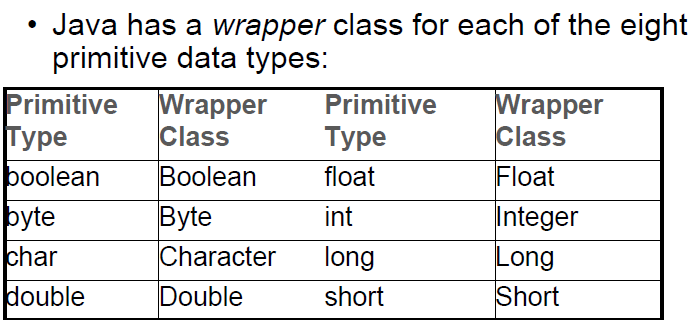
grąžina naują eilutę sudarytą iš zodis1 atmetus jos pradžioje ir gale esančius tarpus. Viduriniai tarpai neatmetami.

String zodis2 = zodis1.**replace**(senaChar, nujaChar);

grąžina naują eilutę, kurioje zodis1 eilutėje visi senaChar simboliai pakeisti naujaChar simboliu

String zodis2 = zodis1.**toUpperCase**();

String zodis3 = zodis1.**toLowerCase**();



Each wrapper class Type has a method typeValue to obtain the object’s value:

**Integer i1 = Integer.valueOf(42);**

**Boolean b1 = Boolean.valueOf(“false”);**

**System.out.println(i1.intValue());**

**System.out.println(b1.intValue());**

**Integer.parseInt(“42”) => 42**

**Boolean.parseBoolean(“true”) => true**

**Double.parseDouble(“2.71”) => 2.71**

Each Number Wrapper has a MAX\_VALUE constant:

**intObj = new Integer(Integer.MAX\_VALUE);**

**longObj = new Long(Long.MAX\_VALUE);**

Skaičių vertimas tekstu yra trys būdai tai padaryti:

1. String s = "" + skaicius;

2. String s = Integer.toString (i); String s = Double.toString (d);

3. String s = String.valueOf (num); Integer ir Double yra

● Operatoriai kartu su argumentais sudaro išraiškas

● Operatorius ima vieną ar kelis argumentus ir sukuria naują reikšmę

● Kai kurie operatoriai gali pakeisti ir pačius argumentus

● Beveik visi operatoriai skirti darbui tik su primityviaisiais tipais

● Tik operatoriai = == != dirba ir su objektais, o + ir += su String objektais

● Loginio operatoriaus rezultatas – loginė reikšmė true arba false

○ Ir (&&)

○ arba (||)

○ ne (!)

● Loginius operatorius galima taikyti tik loginiams (boolean) kintamiesiems ir

reiškiniams

○ boolean t1 = false && true;

**?** if-then-else statement

boolean someCondition = true;

result = someCondition ? value1 : value2;

● Java kalboje naudojamas **tipų konvertavimas** (angl. casting).

● Kur logiška, kintamojo tipas bus konvertuotas automatiškai.

○ Pvz., slankaus tipo kintamajam priskyrus sveikąją reikšmę, ji bus automatiškai paversta į slankaus kablelio formatą.

● Konvertavimą galima/reikia nurodyti ir programoje, tam prieš išraišką skliausteliuose tereikia nurodyti tipą, pvz., (int)a.

● boolean tipo duomenims negalima taikyti tipų konvertavimo

Tipų palyginimo operatorius **instanceof**

Ciklo sakinys **for**(inicializavimas; sąlyga; žingsnis)

Trys ciklo išraiškos apskaičiuojamos skirtingais būdais.

1. Pirma išraiška (inicijavimo segmentas) įvykdomas vieną kartą pačioje ciklo vykdymo pradžioje.

2. Antroji išraiškos tikrinimo dalis atliekama cikliškai kiekvienos ciklo iteracijos pradžioje įskaitant ir patį pirmąjį kartą.

3. Paskutinė trečioji dalis įvykdoma po ciklo kamieno užbaigimo; šioje dalyje standartiniu atveju ciklo išraiškos kintamasis padidinamas arba sumažinamas vienetu.

● Java turi galimybę lanksčiai reguliuoti ciklo kamieno vykdymą.

○ Tarkime jūs turite ciklą kuris paprastai vykdomas 10 kartų, bet ketvirtadieniais jį reikia atlikti tik keturis kartus.

○ Panašaus tipo atvejais naudojami programos vykdymo eigą keičiantys

Java operatoriai **break ir continue.**

break - komanda naudojama perkelti programos vykdymą už ciklo ( for, do, while, switch ) konstrukcijos/kamino/bloko {} pabaigos.

● Ciklas bus užbaigiamas nepaisant jo palyginimo reikšmės ir bus vykdoma einanti po ciklo pabaigos komanda.

● Užbaigiant while ciklą galime panaudoti break komandą, be kurios ciklas gali ir niekada nesibaigti:

● continue komanda panaši į break komandą

● continue komanda priverčia programos vykdymą tęsti praleidžiant po sutikto continue ciklo kamieno komandas

break – nutraukia ciklą

continue – nutraukia dabartinę iteraciją ir grįžta į ciklo pradžią

\*\*\*Po šių operatorių gali būti rašoma žymė, tai leidžia iš karto nutraukti kelis įdėtus ciklus. Žymės sintaksė yra tokia: pirma rašomas žymės vardas, po to dvitaškis ir toliau einantis programos kodas. Praktiškai tai atrodo taip:

**loop**: for(int i=0; i< 10; i++) {

}

**return** komanda panaši į break, tik ja yra užbaigiamas ne ciklo bet metodo vykdymas.

● Java yra objektinio programavimo kalba. Todėl jos visos programos kodas yra rašomas klasėse

● Visos Java programos apibrėžia klasės “būseną” ir “elgesį”, būseną apibrėžiama klasės kintamaisiais, o

elgesys – jos metodais.

● Klasės tipo egzemplioriai (objektai) realizuoja įvairias tos pačios klasės būsenas ir individualią elgseną.

● Objektai naudoja metodus bendraujant tarpusavyje.

● Klasė apjungia savyje grupę kintamųjų.

● Klasių metodai apibrėžia objektų kintamųjų kitimo taisykles.

● Išskirtinė grupė metodų, vadinamų klasių konstruktoriais, apibrėžia klasės kintamuosius (“būseną”) naujo klasės egzemplioriaus kūrimo metu.

Klasė tik su savo laukais yra „negyva“, jų reikšmes gali keisti tik kitų klasių metodai, kuriems keblu palaikyti vieningą klasės elgesį

● Apibrėžyi toje pačioje klasėje metodai nurodo vieningą klasės laukų kitimo elgseną. Metodai rašomi po visų klasės laukų apibrėžimo, nors tai nėra privaloma taisyklė.

Klasės laukai (kintamieji) kartais vadinami

egzemplioriaus (arba objekto) kintamaisiais.

● Bet kuri klasė apibrėžia savo vardo naują tipą. Klasė Circle apibrėžia naują Java duomenų tipą vardu Circle

● Circle ir kitos klasės gali naudoti šį duomenų tipą:

○ Circle aCircle;

○ Circle bCircle;

● Bet kuri sukurta klasė gali būti panauduota apsibriažiant masyvus

● Jei galime susikurti Klasės kintamajį, tai galima susikuriti ir masyvo kintamajį panaudodami naują klasę:

○ Circle[] aCircles;

○ Circle[] bCircles;

Objektai kuriami dinamiškai (vykdymo metu) naudojant raktinį Java žodį new

*Klasė apibrėžia tipą, o to tipo (klasės) kintamuosius vadiname objektais. Objektai skiriasi nuo įprastų kintamųjų tuo, kad juose apibrėžiami* ***metodai aprašantys kintamųjų keitimo taisykles****.*

● Juos galima iškviesti nesukūrus objekto

● Statiniai metodai negali persidengti

Metodai su **static** žyme yra laikomi algoritmais, kuriems nėra reikalingi objekto duomenys ir metodai. Statiniams metodams ir kintamiesiems paskiriama statinė atmintis.

**Statiniai atributai ir metodai**

● Bendri visai klasei

● Bendri visiems klasės objektams/egzemplioriams

● Pasiekiama ne per objektą, o per klasę

○ Pvz. Account.createAccount();

○ Pvz. Account.num\_of\_accounts;

● Naudojama globaliems duomenims bei veiksmams aprašyti

Daugeliu atvejų apibrėždami metodus mes nevartojome modifikatoriaus static. Šiuo atveju metodas interpretuojamas dinaminiu (pagal nutylėjima)

● Visi kintamieji ir metodai turi būti apibrėžti klasėje, statinis modifikatorius pažymi tuos metodus ir kintamuosius, kurie nepriklauso nuo egzemplioriaus.

● Kuriant naują egzempliorių neišskiriama nauja vieta statiniams klasės metodams ir kintamiesiems ir jie visi prieinami pagal fiksuotą klasės pavadinimą.

● Statiniai kintamieji analogiški kitose kalbose naudojamiems global kintamiesiems, skirtumas tik tas, kad jie prieinami tik žinant ir panaudojant klasės pavadinimą.

○ Tai yra Java kalbos privalumas, nes nereikia jaudintis, kad kažkokioje programoje koks nors global kintamojo vardas jau buvo panaudotas ir skirtingos esmės kelių kintamųjų pavadinimas vienu vardu iškels sunkiai aptinkamos programavimo klaidos problemą.

● **Raktas final** prie kintamojo reiškia, kad kintamojo reikšmė nekis – ji galutinė

● Kintamajam reikšmę galima priskirti jį deklaruojant

● Jei reikšmę priskiria kompiliatorius, kintamojo vardas rašomas didžiosiomis

raidėmis, žodžiai skiriami pabraukimais

● Pvz., final float MANO\_PI = 3.14;

**final** modifikatorius nustato, kad kintamojo reikšmė negali būti pakeista. Šis kintamasis iš karto turi būti inicijuojamas ir bet kuris bandymas jį keisti iššauks kompiliavimo klaidą. final modifikatorius paprastai naudojamas apibrėžti

konstantas.

Konstantos dažniausiai nurodomos tokiu modifikatorių rinkiniu:

1. public - viešai preinamas;

2. static - globalus visoms klasėms ir objektams;

3. final - nekintamas.

**public static final int *ETA*** = 64;

Raktų pora static final rodo, kad tai konstanta visiems klasės objektams

● Jei metodas yra final, jo negalima perrašyti paveldėtose klasėse

● Privatūs metodai ir taip yra final net nerašant raktinio žodžio

● Jei final parašysime prieklases, tai iš jos negalima paveldėti

**Paketas** Vienetas apjungiantis Java klases į vieną grupę.

**Klasė** Duomenų tipo apibrėžimas, kuriame yra duomenys ir metodai (paprogramės).

**Metodas** Javoje vartojamas vardas vadinti paprogramės egzempliorių.

**Konstravimas** Programos vykdymo metu klasės kintamojo kūrimas.

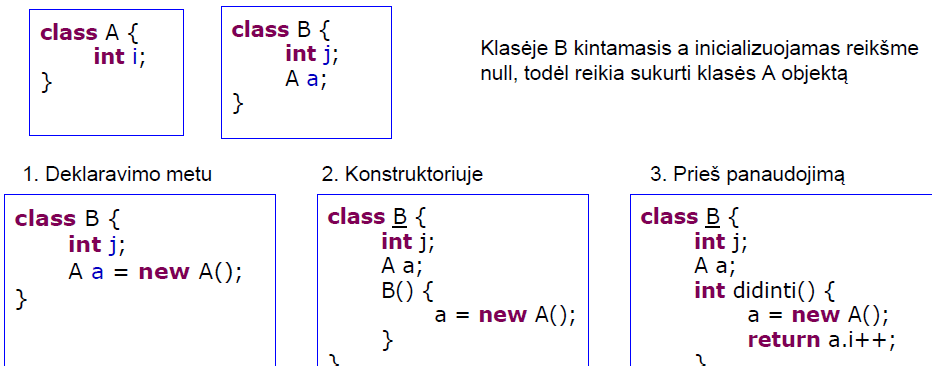
**Panaudojimo modifikatorius** Aprašo, kokia klasių aibė turi teisę keisti duotąjį klasės narį.

Pakartotinio kodo naudojimo būdai

● \*Kodo kopijavimas ir taisymas

● Kompozicija

● Paveldėjimas



● Konstruktoriai nėra paveldimi

● Tėvinės klasės kintamieji turi būti inicijuoti vaikinėje klasėje

● Galima kviesti tėvinės klasės konstruktorių iš vaikinės klasės konstruktoriaus

● Kuriant vaikinių klasių objektus, yra tokia veiksmų seka:

○ Išskiriama vieta

○ Kintamieji inicializuojami pradžioje tėvinėse klasėse, paskui vaikinėse

○ Konstruktorius kviečiamas pradžioje tėvinėms klasėms, paskui vaikinėms\*\*\*\*

● Tėvinės klasės konstruktorius kviečiamas su super() nurodant parametrus

● Kuriant objektus, pradžioje kviečiamas tėvinės klasės konstruktorius, o tik tada vaikinės klasės konstruktorius.

● Taip būna net tada, jei programoje neparašyta, kad reikia kviesti tėvinės klasės konstruktorių

● Kviečiant tėvinės klasės konstruktorių, negali būti jokio kito žingsnio vaikinės klasės konstruktoriuje prieš šį žingsnį

● Tos pačios klasės konstruktorius kviečiamas panaudojant this()

**Perrašomi metodai ir kintamieji**

● Metodo perrašymas (angl. overriding) - tai metodo kūno (angl. body) perrašymas vaikinėje klasėje

● Perrašyti metodai ir kintamieji turi tokį pat aprašymą kaip tėvinėje klasėje

● Vaikinėje klasėje abi versijos yra pasiekiamos

● Prie tėvinės klasės kintamojo arba metodo galima prieiti panaudojant super

● Perrašant metodus, jų priėjimo teisių negalima sumažinti

**Klasės narių panaudojimas**  
Java kalboje galima kontroliuoti klasės narių panaudojimą. Kol kas visi nariai buvo vieši (public). Šis modifikatorius pažymi, kad klasės metodas ar kintamasis yra "demokratiškas" ir juo gali naudotis toje pačioje ir kitose klasėse laisvai.

public alternatyvos: private, (Nutylimas), protected

**Modifikatorius Aprašymas**

**Private** Narys pasiekiamas tik klasės viduje.

**(Nutylimas)** Narys pasiekiamas toms paketo klasėms, kurios nepraplečia šią klasę.

**Protected** Narys pasiekiamas paketo klasėms.

**Public** Narys pasiekiamas kitoms klasėms.

Klasės narių pasiekiamumas (panaudojimas)

● Panaudojimo modifikatorius leidžia programuotojui apsaugoti savo klasę nuo vartotojo neapgalvotų veiksmų.

● Gerai apgalvoti modifikatorių reikšmes naudinga ir pačiam programuotojui, nes grįžus po ilgesnės pertraukos prie klasės redagavimo užsimiršta detalės ir galima padaryti sunkiai aptinkamą klaidą neatsargiai pakeitus kokio nors kintamojo reikšmę.

● Antra vertus private modifikatoriumi neverta piktnaudžiauti, nes jis mažina jūsų klasės panaudojimo lankstumą.

● Tokia metodų ir kintamųjų panaudojimo valdymo konsepcija vadinama "duomenų maskavimu" ( angl. data hiding).

Kintamų panaudojimo valdymas suvaržo jūsų klasės panaudojimą. Objektiniame programavime plačiai naudojamas paveldimumas inheritance, kuris yra vienas pagrindinių šios pragramavimo paradigmos privalumų. Pažymėti, kad naujai rašoma klasė turi paveldėti visus anksčiau parašytos klasės kintamuosius ir metodus, rašomas Java kalbos raktinis žodis extends ir praplečiamos klasės pavadinimas.

● Kalbėdami apie duomenų apsaugos privalomus pristatėme **private** modifikatorių.

○ Šis modifikatorius leidžia naudoti kintamąjį ar klasę tik tos klasės viduje.

○ Yra dar du kiti klasės metodų ir kintamųjų modifikatoriai, tai protected.

**Protected** modifikatorius suteikia teisę naudotis kintamuoju arba metodu tik klasėms, kurios yra tame pačiame pakete. Kokiam paketui yra priskiriama klasė yra paskelbiama jos aprašymo pradžioje:



**Specialūs kintamieji**

Java turi tris iš anksto apibrėžtus kintamuosius: null, this ir super. Pirmieji du yra Object tipo. null žymi neegzistuojantį objektą, o this nurodo tą patį objekto egzempliorių. super nurodo į tiesioginę superklasę.

Null

● Esama aptarę, kad bet koks klasę žymintis kintamasis turi būti inicijuojamas.

● Jei jis nėra inicijuotas, jo reikšmė lygi null specialiam kintamajam.

● null objektas neturi jokių kintamųjų ir metodų, todėl su juo negalime atlikti jokių manipuliacijų.

○ Gan dažnai pasitaikančios programavimo klaidos, kai bandoma panaudoti neinicijuotą objektą. Tuomet gausite standartinį pranešimą: NullPointerException.

null panaudojimo pavyzdys, kurio vykdymas sustotų ir būtų gautas pranešimas NullPointerException:

ReplaceChars B;

someMethod(B);

Kad programa nesustotų, teks tikrinti ar objektai nėra null. Jeigu klasės tipo kintamasis nėra null - galėsime juo naudoti, priešingu atveju - nesinaudosime, kad negauti NullPointerException klaidos pranešimą.

**this kintamasis**

● Kartais jums reikės perduoti nuorodą einamo objekto į metodą ar konstruktrių. Tuo atveju ir naudojamas this.

● Tarkime Son ir Daughter konstruktoriai naudoja Mom kintamąjį. Šiuo atveju ir pravers this kintamasis.

Kai Mom konstruoja savo Sons ir Daughters, jei reikės perduoti nuorodą į save pačią. Praktiškai tai gali atrodys taip

**public** Mom() {

firstSon = **new** Son(**this**);

secondSon = **new** Son(**this**);

firstDaughter = **new** Daughter(**this**);

secondDaughter = **new** Daughter(**this**);

**super** pagalba galime pasiekti tėvinės kalsės kintamosius ir metodus

Konstruktoriai, kaip ir metodai gali naudoti super kintamąjį Mūsų poklasė gali panaudoti parašytą konstruktoriuje

kodą panaudodama super kintamąjį.

Naudojant super kintamąjį su konstruktoriais reikia atsiminti porą išimčių. Pirma, šį kintamąjį galite naudoti tik

konstruktoriaus kamiene. Antra, jis turi būti panaudotas pačioje konstruktoriaus kamieno pradžioje.

**Klasių hierarchija**

● Visos klasės paveldi klasės **java.lang.Object**

● Jei nėra nurodyta tėvinė klasė, tai pagal nutylėjimą paveldės Object savybes

● Jei kokia nors tėvinė klasė nurodyta, tada paveldės Object savybes per tėvinę klasę

**Superclass – TEVAS**

Java kalboje kiekviena klasė turi tik vieną tiesioginę superklasę

● Kodėl tokia globali hierarchija naudinga?

○ kadangi žinome, kad visos klasės turi bendrą superklasę Object, jos metodus ir kintamuosius gali naudoti bet kuri kita klasė.

● Object klasė apibrėžia equality metodą skirtą patikrinti, ar dviejų klasių turinys vienodas.

● Taip pat Object klasėje yra realizuotas daugiagijiškumo (multithreading) savybės.

● Taip pat atpuola daugelio hierarchijų tarpusavio sąsajų problema, nes visos jos yra vienos globalios hierarchijos dalys.

● Naudinga ir tai, kad esame garantuoti, kad kiekviena klasė turi savo superklasę.

**Keletas klasės Object metodų**

● equals – lygina objektų turinį

● clone – sukuria naują objektą - to paties objekto kopiją

● toString – grąžina tekstinę objekto informaciją

**instanceof** - patikrina kokios klasės objektas

Paveldėjimo privalumą mes grindėme tuo, kad jis leidžia neperrašinėti iš naujo klasės kodo ir visą jį paveldėti poklasėms. Tačiau tai tik viena naudinga paveldimumo savybė.

Kitas paveldimumo panaudojimas yra susietas su klasių hierarchijos konstravimu.

Duomenų tipo pakeitimas (casting). Tarkime turėjome lowfatMilkType kintamąjį ir jį norime pakeisti dairyProduct tipu.

**LowfatMilk m=new LowfatMilk();**

**DairyProduct d=m;**

**if (d.sourThisWeek()) (**

**System.out.println("Nepirk");}**

**Paveldėjimas Inheritance**

Java kalboje kiekvienas paveldėtos klasės objektas gali būti naudojamas ten, kur reikalingas tėvinės klasės objektas!

Polimorfizmas - galimybė turėti skirtingas elgsenas (skirtingas metodų realizacijas) skirtingose situacijose naudojant tipus

○ Polimorfizmu vadiname Java savybę, kuri suteikia galimybę grupei metodų naudoti tą patį metodą, tačiau kiekvienas metodo panaudojimas gali duoti skirtingus rezultatus.

● Jei imame nuorodą į objektą ir ją naudojame kaip nuorodą į bazinę klasę, toks veiksmas vadinamas nukirtimu iš viršaus (angl. upcasting)

○ Pvz., Kvadratas a = new Kvadratas();

○ Figura b = a;

● Nukirtimas iš apačios (angl. downcasting) yra nuorodos į bazinę klasę naudojimas vietoje nuorodos į paveldėtą klasę

○ Pvz., Figura a = new Kvadratas();

○ Kvadratas b = (Kvadratas)a;

Programinis kodas lengvai rašomas ir skaitomas

Klasės sąsaja vienoda, tipų specifika svarbi realizacijoje. Ne įtakojamos klasės – naudotojos

Jei reikia vaikinės klasės savybių, klasėse-naudotojose reikia būtinai panaudoti downcast veiksmą

○ Pvz., Figura a = new Kvadratas();

○ Kvadratas b = (Kvadratas)a;

● Abstrakti klasė (angl. abstract class) parūpina bendrą sąsają klasėms, kurios paveldi abstrakčios klasės savybes

● Abstraktus metodas bei klasė žymimi raktiniu žodžiu abstract

● Abstraktus metodas turi deklaraciją, bet neturi realizacijos

Abstrakčios klasės objektai nekuriami - kompiliatorius neleis šito daryti

○ abstract klasės objektą (egzempliorių) draudžiama kurti, tačiau apibrėžti šio tipo kintamąjį yra leistina.

● Abstraktūs metodai realizuojami vaikinėse klasėse

● Abstrakti klasė taip pat gali būti vaikine abstrakčios klasės klase.

● Abstrakti klasė gali turėti konstruktorius

● Abstrakčioji klasė - ne visiškai realizuota klasė, turinti deklaruotų bet nerealizuotų (abstract) metodų.

○ Įgalina abstraktųjį programavimą, kai ne visos klasės veikimo detalės yra žinomos kodo rašymo metu, t.y., paliekamos realizuoti išvestinėse klasėse.

● Jeigu klasė būdama abstrakčiosios klasės palikuoniu nerealizuoja visų bazinių klasių abstrakčiųjų metodų, ji taip pat turi būti deklaruojama kaip abstract.

Interfeisus (interface) reikėtų suprasti kaip visiškai abstrakčias klases. Įgalina modeliuoti klasių daugialypį paveldėjimą, kuris Javoje draudžiamas.

● Kiekviena klasė gali turėti (extend) tik vieną bazinę klasę, tačiau įgyvendinti/realizuoti (implement) keletą interfeisų.

○ Savo ruožtu, interfeisas gali išplėsti (extend) vieną ar daugiau interfeisų, ir tai suprasime kaip baziniuose interfeisuose deklaruotų metodų aibių apjungimą.

● Kiekvienas **interfeiso metodas traktuojamas kaip "public abstract", o laukai - "public static final" - konstantos**.

● Leidžiama į interfeisą įtraukti "default" metodų realizacijas. Nuo java 8-sios versijos.

Interface klasė deklaruojama žodžiu interface, nerašant žodžio class(!)

● Vaikinė interface klasės klasė turi naudoti raktinį žodį implements ir realizuoja atitinkamus metodus

● Pagal nutylėjimą interface klasės **metodai yra public,** todėl jų realizacijos irgi **public**

● Interface klasė gali būti ir interface klasės vaikinė klasė

● interface neturi konstruktorių

● Tiek interfeisas, tiek abstrakti klasė pateikia programuotojui sąsają, o realizacija nebūtina

● Be to, interfeisas leidžia nukirsti (upcast) iki daugiau nei vienos klasės

● Interfeisas turi visus abstrakčios klasės privalumus

**Išimtys**

● Ten, kur įvyko klaida, gali būti neaišku, kaip ją spręsti. Vykdymas sustabdomas. Klaidos apdorojimas perkeliamas į aukštesnį lygį

● To paties tipo klaidų apdorojimą galima perkelti į vieną vietą. Klaidų apdorojimas atskiriamas nuo kito kodo (try catch)

Kuo skiriasi išimtinė situacija nuo įprastos problemos?

● Išimtinė situacija - tai problema, neleidžianti tęsti metodo ar kodo bloko vykdymo. Problemai spręsti einamajame kontekste trūksta informacijos. Viskas, ką galima padaryti, - tai peršokti iš einamojo konteksto į aukštesnį lygį. Tai atliekama generuojant išimtį

○ throw new “exception type”

● Įprasta problema – kai einamajame kontekste yra pakankamai informacijos problemai išspręsti. Tokia problema sprendžiama vietoje.

○ PVZ: įvestas netinkamas duomuo paprašome dar kartą įvesti

1. Sukuriamas išimties objektas new

2. Iki šiol vykdyta veiksmų seka stabdoma

3. Į išimties objektą įdedama nuoroda į einamąjį kontekstą

4. Įsijungia išimčių apdorojimo mechanizmas ir nuo šios vietos pradeda ieškoti atitinkamo išimties apdorotojo (angl. exception handler)

**private static void** printUpper(String tekstas) {

**if** (tekstas == **null**) {

**throw new** IllegalArgumentException(

"Kintamasis tekstas turi buti ne null!!");

}

● Išimties objektas kuriamas naudojant new, kuris išskiria atmintį ir iškviečia konstruktorių

● Yra N konstruktoriai: be parametrų ir su vienu String tipo parametru ir...

● Išimties objekto tėvinė klasė yra Throwable

● Informacija apie klaidą yra išimties objekte, be to, klaidos pobūdį nusako išimties objekto klasės vardas

● Apdorotas išimtis sunaikina šiukšlių surinkėjas

● Checked & Unchecked Exceptions

● Unchecked Exceptions:

○ paveldi iš RuntimeException ir Error

○ Specifikuoti nereikia: pvz. void f() ~~throws TooBig, TooSmall {~~

● Checked Exceptions:

○ Kuriant metodą būtina nurodyti, kokias išimtis gali generuoti šis metodas, pvz., void f() throws TooBig, TooSmall {

○ Jei throws nėra (pvz., void f() {), reiškia metodas išimčių (ne RuntimeException) generuoti negali.

○ Leidžiama specifikuoti throws daugiau išimčių, nei iš tikrųjų gali būti generuojama

Gaudant bazinės klasės išimtis sugaunamos ir vaikinių klasių išimtys, t. y. catch(Exception e) {

Šiuo atveju gaunama mažai informacijos apie konkrečią išimtį, tačiau ją galima gauti pasinaudojus bazinės klasės Throwable metodais:

● String getMessage()

● String getLocalizedMessage()

● String toString()

● void printStackTrace()

● Throwable fillInStackTrace()

● Object metodas getClass() ir klasės metodas getName()

Išimčių vardai:

● ArrayIndexOutOfBoundsException

● ClassCastException

● ClassNotFoundException

● IllegalArgumentException

● NegativeArraySizeException

● NoSuchFieldException

● NullPointerException

● NumberFormatException

**try {**

//kodas, galintis generuoti išimtis

**} catch** (Tipas1 id1) {

//išimčių, kurių tipas Tipas 1, apdorojimas

} catch (Tipas2 id2) {

//išimčių, kurių tipas Tipas 2, apdorojimas

**} finally {**

//veiksmai, kurie atliekami visada

}

Kada naudoti finally?

● Nenaudojamas atminčiai atlaisvinti, tai atlieka šiukšlių surinkėjas

● Naudojamas failams uždaryti, tinklo ryšiams nutraukti, ekranui išvalyti ...

● finally vykdomas visada, net jei išimtis nebuvo generuota ar nebuvo sugauta

● Trūkumas: jei sakinyje finally generuosim naują išimtį, tai prieš tai buvusi bus prarasta

**Apribojimai konstruktoriams**

● Konstruktoriai gali generuoti bet kokias išimtis, t.y. apribojimai, kurie galioja

○ kitiems metodams, konstruktoriams netaikomi

● Paveldėtos klasės konstruktorius turi specifikuoti ir visas bazinės klasės konstruktoriaus išimtis (nes paveldėtos klasės konstruktorius kviečia bazinės klasės konstruktorių), jei išimtys nepaveldi RuntimeException

● Paveldėtos klasės konstruktorius negali gaudyti bazinės klasės konstruktoriaus generuojamų išimčių

● Catch sakinyje galima nurodyti ne gaudomą, o bazinę išimtį

● Pirmiau nurodžius bazinę išimtį, o po to paveldėtą išimtį, kompiliatorius duos klaidą