T04. Objektinis programavimas. Klasės

1. Klasė ir jos sudėtinės dalys:

konstruktoriai

kintamieji

metodai

2. Objektas :

objekto sukūrimas

metodų iškvietimas

parametrų perdavimas į metodus

reikšmės grąžinimas

Objektas:

- tai programinis komponentas, modeliuojantis realaus ar virtualaus pasaulio elementus: žmogus, gyvūnas, mašina, parduotuvė ir t. t.;

- turi atributus (**objekto kintamuosius**): dydį, formą, spalvą, svorį, greitį ir t. t.;

- turi elgseną ir manieras (**objekto metodus**): keičia dydį, kaupia sumą, formuoja atsakymą, pasipildo prekėmis ir t.t.

Objektinis programavimas

Klasė - tai objekto **tipas** (šablonas), kurį sudaro **duomenys** ir **metodai**;

- tai tam tikro tipo objekto duomenų struktūra (būsena) ir manipuliavimo ja taisyklės (elgesys);
- tai "tipo" sąvokos išplėtimas.

Objektas – tai realus klasės egzempliorius kompiuterio atmintyje (objektui sukurti dažnai naudojama funkcija **new**)

Objektinis programavimas

Tai programavimo stilius, naudojantis šias **sąvokas**:

1. **Objektas** : unikali duomenų struktūra ir manipuliacijos su jais;
2. **Inkapsuliacija** : duomenų slėpimas – jais manipuliuoti gali tik objekto metodai;
3. **Polimorfizmas** : tuo pačiu vardu metodai vykdomi skirtingai (perkrova ir užklotis);
4. **Paveldėjimas** : klasė konkretizuojama kita klase (tėvo ir vaiko klasės);
5. **Abstrakcija** : manipuliacija objekto duomenimis nežinant vidinio duomenų formato.

Inkapsuliacija

Inkapsuliacija – tai objekto vidinių duomenų (ir jų formato) slėpimas, kai jais galima manipuliuotis tik naudojant objekto viešus metodus pagal duotą sąsają.

Tai leidžia programuojant neprisirišti prie objekto vidinės struktūros – galima vykdyti vidinės objekto struktūros tobulinimo darbus, keisti vidinius kintamuosius, metodus. Tuo tarpu išorinis objekto elgesys lieka nepakitęs.

Kapsuliavimas leidžia rašyti programas „aukštesniu“ lygiu, t.y. pasitikėti esamais metodais ir taip išvengti klaidų.

Polimorfizmas - perkrova

Metodų perkrova (overloading).

Klasėje naudojama keletas metodų tuo pačiu vardu.

Būtina sąlyga – metodai privalo skirtis savo antraštėmis.
Grąžinamų reikšmių tipai nelaikomi skirtumu.

Metodo pasirinkimą atliekama kompiliatorius.

Polimorfizmas - užklotis

Metodų užklotis (overriding). Galima tik paveldėjime.

Užklotis – tai galimybė vienodai pavadintą veiksmą realizuoti skirtingais būdais skirtingose klasėse.

Būtinos sąlygos: sutampa metodų vardai, jų antraštės ir grąžinamų reikšmių tipai (vaiko metodas užkloja/pakeičia tėvo metodą).

Metodo pasirinkimas atliekamas vykdymo metu pagal objekto tipą.

Klasės pavyzdys

```
class Klientas {
//  1. Objekto kintamieji:
     private  String kodas;
private  double indėlis;
//  2. Klasės konstruktoriai:
public  Klientas(String kodas, double indėlis) {
this.kodas   = kodas;
this.indėlis   = indėlis;
   }
//  3. Klasės metodai:
public  double getIndėlis() {
return   indėlis;
}
public  String getKodas() {
return   kodas;
}
}
```

Objekto sukūrimas

2. Objektai:

```
Klientas  x; //   x  yra tik  Klientas tipo kintamasis
      //          Tai dar ne objektas  !!!

x  = new Klientas( ); //   x  yra klasės Klientas objektas
                  //   new  yra speciali funkcija objektui sukurti;
//      Klientas( )  yra klasės Klientas konstruktorius
```

Arba tas pats vienu sakiniu:

```
Klientas  x = new Klientas( );
```

Formalus klasės aprašas

```
///  [požymis] – tai public, private, protected arba tuščia
[požymis]  class KlasėsVardas
[extends    TėvoKlasėsVardas ]
          [implements InterfeisoVardas [, InterfVardas ]]  {
//  1. Kintamieji :
[požymis]  tipas  kintamojoVardas;
[požymis]  tipas  kintamojoVardas =  pradinėReikšmė;

//  2. Konstruktoriai :
KlasėsVardas(  [parametrų sąrašas] )
{
konstruktoriaus   sakiniai
}
```

Formalus klasės aprašas - tęsinys

// **3. Metodai :**

[požymis] **rezultatoTipas metodoVardas(** [parametrų_sąrašas] **) {**

metodo kintamieji;

metodo sakiniai;

return grąžinama reikšmė;

} // metodo pabaiga

} // klasės pabaiga

/// Suteiktas matomumo požymis (modifikatorius) galioja tik tam kintamajam, konstruktoriui, metodui.

Klasės pavyzdys

```
class Klientas {

//  0. Klasės kintamasis – visi klientai to paties banko:
public  static final String bankoKodas="LT09";

//  1. Objekto kintamieji:
private  String kodas;
private  int amžius;
private  double indėlis;
//    …
//    …
//    …
```

Klasės pavyzdys - konstruktoriai

```
//  2. Klasės konstruktoriai:

//  Šis konstruktorius sukuria objektą su nulinėmis savybėmis
public  Klientas() {
//      Įvedus savo konstruktorius, numatytas (default)
//      konstruktorius  automatiškai nebesukuriamas
}

//  Šis konstruktorius sukuria objektą su nurodytomis savybėmis
//  Toks konstruktorius gali būti generuojamas automatiškai
public  Klientas(String kodas, int amžius, double indėlis) {
this.kodas    = kodas;
this.amzius    = amžius;
this.indėlis    = indėlis;
    }
```

Klasės pavyzdys - metodai

```
//  3. Klasės metodai (geteriai ir seteriai ? ?)
//  jie gali būti generuojami automatiškai

public  double getIndėlis() {
return   indėlis;
}
public  String getKodas() {
return   kodas;
}
public  int getAmžius() {
return   amzius;
}
```

Klasės pavyzdys - metodų tęsinys

```
//   3. Klasės  esminių veiksmų metodai
public  void keistiIndėlį(double pokytis) {
indėlis   += pokytis;
}
//  .....
//  3. Klasės  objektų vaizdavimo metodas
//  Perrašomas (užklojamas) Object metodas toString,
//     skirtas savybėms pavaizduoti.
public  String toString() {
return   String.format("%7s %3d %9.2f‟
,       kodas, amžius, indėlis);
    }

} // klasės Klientas pabaiga
```

Objekto sukūrimas

Bendras formatas:

KlasėsArbaInterfeisoVardas **objektoVardas** = **new** KlasėsKonstruktorius([parametrai]);

Dažniausias formatas:

KlasėsVardas **objektoVardas** = **new** KlasėsVardas([parametrai]);

Objekto sukūrimas

Kiti formatai:

TėvoKlasėsVardas **objektoVardas** = **new** VaikoKlasėsVardas([parametrai]);

InterfeisoVardas **objektoVardas** = **new** InterfeisąDiegiančiosKlasėsVardas([parametrai]);

Klasės *Klientas* testas

```
public class Testas  {
    public static void main(String p[ ]) {
    Klientas  b1= new Klientas();
    Klientas  b2= new Klientas("SEB476", 42, 533.20);
    Klientas  b3= new Klientas("SWE293", 12,  23.10);
    //  spausdinant suveikia metodas toString()
    System.out.println("Atskiras  1-as klijentas -> " + b1);
    System.out.println("Atskiras  2-as klijentas -> " + b2);
    double  sumaInd= b1.getIndėlis() + b2.getIndėlis() +
        b3.getIndėlis();
    System.out.println("Indėlių  suma ="+sumaInd);
    }
}
```

Rezultatas:

```
Atskiras 1-as klijentas -> null      0      0,00
Atskiras 2-as klijentas -> SEB476   42    533,20
Indėlių suma =556.3000000000001
```

Objekto kintamieji ir metodai

* Kiekvienam klasės objektui bus sukuriama atskira šio **kintamojo** kopija.

* Objekto **metodas** bus bendras visiems šios klasės objektams.

* Objekto metodas gali naudoti tiek **objekto**, tiek ir **klasės** kintamuosius (*static*).

Objekto metodas gali kviesti **statinius** metodus, naudojant klasės vardą.

* Objekto metodas gali kviesti tos pačios klasės metodus ir tiesiogiai (be objekto vardo).

Klasės (*static*) kintamieji ir metodai

Visi šios klasės objektai naudos **tą patį** *static* kintamąjį;

static metodas gali naudoti klasės kintamuosius, bet negali tiesiogiai dirbti su objekto kintamaisiais.
Prieš tai būtina sukurti klasės objektą ir kreiptis per jį (jei tai leidžia kintamojo matomumo požymis).

static metodas **negali** tiesiogiai (be objekto vardo) kviesti net ir tos tos pačios klasės **objekto** metodų.

static kintamieji ir metodai

static kintamajam ir **static** metodui negalima naudoti ***this*** („šis” objektas).

* Tos pačios klasės ***static*** tipo metodą galima kviesti ir be klasės vardo.

* **static** metodas negali turėti vidinių ***static*** tipo kintamųjų.

Konstruktorius

Tai **metodas**, kurio vardas sutampa su klasės vardu.

Paskirtis – kintamųjų pradinių reikšmių nustatymas.

* Klasė gali turėti daug konstruktorių (perkrova).
  Reikiamas pasirenkamas objekto sukūrimo metu.

* Praleidus klasėje konstruktorius, automatiškai sukuriamas numatytas (be parametrų) konstruktorius. Toks konstruktorius objekto kintamiesiems priskiria nulines reikšmes (pagal kintamojo tipą).

Konstruktorius gali turėti parametrų, bet negali grąžinti reikšmės.
Jis neturi ir ***void*** .

* Java neturi **destruktoriaus**.

Konstruktorių perkrova su *this*

```
public class Metai {
private  int metai;
private  int menuo;
private  int diena;
 public  Metai() {
this(0,   0, 0);
}
 public  Metai(int m) {
this(m,   0, 0);
}
 public  Metai(int m, int men) {
this(m,   men, 0);
}
public  Metai(int m, int men, int d) {
metai   = m;
menuo   = men;
diena   = d;
}
```

** Sakinys **this** privalo būti pirmu konstruktoriaus sakiniu (pirmi trys kviečia ketvirtą konstruktorių).

Metodo iškvietimas

Galimi trys metodo iškvietimo formatai:

objektoVardas.metodoVardas([parametrai]**)**;// visiems metodams

KlasėsVardas.metodoVardas([parametrai]**)**; // tik klasės metodui

Specialus atvejas – metodas kviečia tos pačios klasės metodą:

metodoVardas([parametrai]**)**;

Parametrai (this)

Parametro vardas gali sutapti su objekto kintamojo vardu.

Vardų konfliktą išsprendžia žodelis **this**

```
class Vardai {
    private int x, y;
    public void keistiXY(int x1, int y) {
     x = x1;
     y = y; //  rezultas blogas !! (objekto kintamasis liks 0)
     this.y = y; // gerai
    }
}
```