Objektorientētā programmēšana Java valodā

polimorfisms un abstrakcija

Ainārs Skrubis 2PT

2025. gada jūnijs

1. TEORIJA

1.1. Objektorientētā programmēšana (OOP) OOP ir programmēšanas paradigma, kuras pamatā ir objekti. Galvenie OOP principi:

- Mantošana (Inheritance)
- Iekapsulēšana (Encapsulation)
- **Abstrakcija (Abstraction)**
- **Polimorfisms (Polymorphism)**

1.2. Polimorfisms(Polymorphism) ir koncepts kas nodrošina iespēju vienu darbību veikt dažādos veidos.

1.2.1. Polimorfisma galvenās iezīmes:

- **Vairākas uzvedības:** Viena un tā pati metode var uzvesties atšķirīgi atkarībā no objekta, kas to izsauc.
- **Metodes ignorēšana(Overriding):** Bērnu klase var pārdefinēt savas vecākklases metodi.
- **Metodes pārslodze(Overloading):** Var definēt vairākas metodes ar vienādu nosaukumu, bet atšķirīgiem parametriem.
- **Izpildes laika lēmums:** Izpildes laikā Java nosaka, kuru metodi izsaukt, atkarībā no objekta faktiskās klases.

1.3. Abstrakcija (Abstraction) ir process, kas progrmmas lietotājam parāda tikai tam nepieciešamo informāciju un slēpj datus, kas tam nav nepieciešamo. Galvenais abstrakcijas mērķis ir datu slēpšana.

1.3.1. Abstrakcijas galvenās iezīmes:

- Abstrakcija slēpj sarežģītas detaļas un parāda tikai būtiskās iezīmes.
- Abstraktām klasēm var būt metodes bez ieviešanas, un tās jāievieš apakšklasēs.
- Abstrahējot funkcionalitāti, izmaiņas ieviešanā neietekmē kodu, kas ir atkarīgs no abstrakcijas.

1.3.2. Saskarnes(Interface)

Saskarnes ir vēl viena abstrakcijas ieviešanas metode Java valodā. Galvenā atšķirība ir tā, ka, izmantojot saskarnes, Java klasēs var panākt **100% abstrakciju**. Java vai jebkurā citā valodā saskarnes ietver gan metodes, gan mainīgos, bet tām trūkst metodes pamatteksta. Papildus abstrakcijai saskarnes var izmantot arī mantojuma ieviešanai Java valodā.

2. KODA PIEMĒRI

2.1. Apstrakcijas piemērs

```java
//abstraktā klase ar abstraktām metodēm
abstract class Cilveks {
 abstract void ieslekt();
 abstract void izslekt();
}
class Pults extends Cilveks {
 @Override
 void ieslekt() {
     System.out.println("TV ir ieslekts.");
 }
 @Override
 void izslekt() {
     System.out.println("TV ir islekts.");
 }
}
//Main klase
public class Abstrakcija {
 public static void main(String[] args) {
     Cilveks remote = new Pults();
     remote.ieslekt();
     remote.izslekt();
 }
}
```

```
TV ir ieslekts.
TV ir islekts.
```

2.2. Polimorfisma piemērs

```java
class Virietis{
 void disk() {
     System.out.println("Es esmu vīrietis.");
 }
}

//Klase kas pārakstīs Cilveks klases metodi
class Tevs extends Virietis {

 // Pāraksta metodi (Overriding)
 @Override
 void disk() {
     System.out.println("Es esmu tēvs.");
 }
}

public class Polimorfisisms {
 public static void main(String[] args) {
     Virietis p = new Tevs();
     p.disk();
 }
}
```

```
Es esmu tēvs.
```

2.3. Saskarnes(Interface) piemērs

```java
//Definē interface vārdu Forma
interface Forma {
 double aprLauk(); //Abstraktā metode apreikina laukumu
}
//Izmanto mantošanu klasē Aplis
class Aplis implements Forma {
 private double r;    // rādius
 public Aplis(double r) {
   this.r = r;
 }
 //Implementē abstrakto metodi
 public double aprLauk()
 {
     return Math.PI * r * r;
 }
}
//Implementē interface clasē taisnstūris
class Taisnsturis implements Forma {
 private double garums;
 private double platums;
 public Taisnsturis(double garums, double platums)
 {
     this.garums = garums;
     this.platums = platums;
 }
 //Implementē apstrakto metodi
 public double aprLauk() {
   return garums * platums;
 }
}
//Main klase
public class Interface {
 public static void main(String[] args)
 {
     //Izveido apļa un taisnstūra objektus
     Aplis a = new Aplis(5.0);
     Taisnsturis t = new Taisnsturis(4.0, 6.0);
     System.out.println("Apļa laukums: "+ a.aprLauk());
     System.out.println("Taisnstūra laukums: "+ t.aprLauk());
 }
}
```

```
Apļa laukums: 78.53981633974483
Taisnstūra laukums: 24.0
```

3.Uzdevumi

1.Uzd

Izveido abstraktu klasi `Transportlīdzeklis`, kurā ir abstrakta metode `parvietoties()`. Izveido divas apakšklases – `Auto` un `Velosipēds`, kuras realizē `parvietoties()` metodi atšķirīgi.

```java
// Abstrakta klase
abstract class Transportlidzeklis {
    abstract void parvietoties();
}
// Auto klase, kas manto no Transportlidzeklis
class Auto extends Transportlidzeklis {
    @Override
    void parvietoties() {
        System.out.println("Brauc ar auto pa ceļu.");
    }
}
// Velosipēds klase, kas manto no Transportlidzeklis
class Velosipeds extends Transportlidzeklis {
    @Override
    void parvietoties() {
        System.out.println("Min pedāļus pa veloceliņu.");
    }
}
// Galvenā klase ar main metodi
public class Abstrakcija {
    public static void main(String[] args) {
        Velosipeds velo = new Velosipeds();
        velo.parvietoties();

        Auto auto = new Auto();
        auto.parvietoties();
    }
}
```

```
Min pedāļus pa veloceliņu.
Brauc ar auto pa ceļu.
```

2.Uzd

Izveido bāzes klasi `Dzivnieks`, kurā ir metode `skaņa()`. Izveido klases `Suns`, `Kakis`, kurās šī metode tiek pārrakstīta. Tad izveido masīvu ar šiem objektiem un izsauc `skaņa()` visiem.

```java
class Dzivnieks {
    void skana() {
        System.out.println("Dzīvnieks izdod skaņu.");
    }
}
//Suns klase kas manto no Dzivnieks
class Suns extends Dzivnieks {
    @Override
    void skana() {
        System.out.println("Vau vau!");
    }
}
//Kakis klase kas manto no Dzivnieks
class Kakis extends Dzivnieks {
    @Override
    void skana() {
        System.out.println("Miau miau!");
    }
}
//Main klase
public class Polimorfs {
    public static void main(String[] args) {
        Dzivnieks[] dzivnieki = { new Suns(), new Kakis(), new Dzivnieks() };

        for (Dzivnieks d : dzivnieki) {
            d.skana(); // Polimorfisms: katrs objekts izpilda savu versiju
        }
    }
}
```

```
Vau vau!
Miau miau!
Dzīvnieks izdod skaņu.
```

3.Uzd

Izveido saskarni `Izdrukajams`, kurā ir metode `izdrukat()`. Izveido divas klases – `Gramata` un `Avize`, kas šo metodi īsteno. Demonstrē, kā izmantot šo saskarni dažādos objektos.

```java
// Saskarne Izdrukajams ar vienu metodi, kas jārealizē visām klasēm kas to implementē
interface Izdrukajams {
    void izdrukat(); // Abstrakta metode bez kermena
}
// Klase Gramata, kas realizē saskarni Izdrukajams
class Gramata implements Izdrukajams {
    @Override
    public void izdrukat() {
        System.out.println("Drukā grāmatu...");
    }
}
// Klase Avize, kas arī realizē saskarni Izdrukajams
class Avize implements Izdrukajams {
    @Override
    public void izdrukat() {
        System.out.println("Drukā avīzi...");
    }
}
//Main klase
public class Interface {
    public static void main(String[] args) {
        // Izveidojam mainīgos no tipa Izdrukajams, bet pieškiram dažādas klases
        Izdrukajams g = new Gramata();
        Izdrukajams a = new Avize();
        // Izsaucam metodes - katra klase izpilda savu versiju
        g.izdrukat();
        a.izdrukat();
    }
}
```

```
Drukā grāmatu...
Drukā avīzi...
```

4. Izmantotie avoti

- GeeksForGeeks (https://www.geeksforgeeks.org/java/polymorphism-in-java/)
- GeeksForGeeks (https://www.geeksforgeeks.org/java/abstraction-in-java-2/)
- W3School (https://www.w3schools.com/java/java_polymorphism.asp)
- W3School(https://www.w3schools.com/java/java_abstract.asp)
- ChatGPT
- Skolo.lv (Rāvalds java programmēšanas kursi)