Harjutustund 9

Ülesannete kontroll. Pärilus. Meetodite ülekatmine.

Pärilus

OO keeltes asuvad klassid teatud ülemklasside ja alamklasside hierarhias. Tahtes olemasolevale klassile lisada võimalusi ja omadusi (meetodid, väljad), kuid seda klassi ennast mitte muuta, võime luua uue klassi, milles kirjeldame lisatavad meetodid ja väljad. Niiviisi loodud klassi nimetame *alamklassiks* (*subclass*, ka *child class*, *extended class*, *derived class*) ja klassi, millest lähtusime, *ülemklassiks* (*superclass*, ka *parent class*, *base class*). Sellist tegevust võib üldjuhul korduvalt jätkata. Ülemklassi omadused kanduvad üle tema alamklassidele. Vastavat mehhanismi nimetatakse päriluseks (*inheritance*). Pärilus on üks objektorienteeritud programmeerimise alustalasid, võimaldades loodud klasside taaskasutamist ja tekitades klasside hierarhilise struktuuri. Hierarhia juureks on klass java.lang.Object, mis on kõigi teiste klasside otseseks või kaudseks ülemklassiks.

Eelmistes praktikumides loodud klasside puhul me ei märkinud kuidagi eraldi, et tegemist on klassi Object alamklassidega, seda arvestati vaikimisi. Kui aga loome mõne teise klassi alamklassi, siis tuleb seda eraldi märkida. Märkimaks, et klass B on klassi A alamklass, tuleb klassi nime järele lisada võtmesõna extends ja ülemklassi nimi:

```
class A {
  . . .
}

class B extends A {
  . . .
}
```

Kui *Eclipse*'is luua uus klass (File - New - Class), siis saab ülemklassi määrata vastaval real *Superclass*. Vaikimisi on seal real java.lang.Object.

Alamklassis on tegelikult olemas kõik need ülemklassi muutujad ja meetodid, mille piiritlejaks ei ole private. Eelmisess praktikumis lõime klassi `Student`. Püüame nüüd sellele klassile luua ülemklassi `Person`, milles on sõnetüüpi isendiväli isikukoodi märkimiseks.

```
public class Person {
    private String name;
    private String personalId;
}
public class Student extends Person {
    private String studentId;
}
```

Modifitseerime `Student` klassi nii, et selles klassis oleks ainult `studentId` väli, kuna ülejäänud väljad on juba määratud ülemklassis.

Lisame ka konstruktorid, mida klassi `Student` isendi loomisel saab kasutada. Kui kasutame *Eclipse*'it, siis saame *Source*-menüüst kasutada nii *Generate Constructors using Fields ...* kui ka *Generate Constructors from Superclass ...*

Võtmesõnaga super saame pöörduda ülemklassi konstruktori poole. Tingimuseks on, et see käsk (kui ta on vajalik), asuks konstruktori esimesel real (kommentaariridu arvestamata). (Ka võtmesõnaga this sama klassi teise konstruktori väljakutsumine peab olema esimesel real.) Kui selliseid väljakutseid konstruktoris kirjas pole, siis toimub siiski pöördumine ülemklassi ilma argumentideta konstruktori poole.

Konstruktor, millele saaks ette anda nii nime, pikkuse, kui ülikooli, näeb välja selline.

```
public Student(String name, double studentId, String personalId) {
    super(name, personalId);
    this.studentId = studentId;
}
```

Nüüd saame test-klassis `Student` isendeid luua.

```
Student s1 = new Student("Mari Tamm", "131416", "49007080909");
Student s2 = new Student("Mart Tamm", "131417", "39007080989");
```

Ülesanne 1

Looge klassi `Student` ülemklass `Person` vastavalt ülaltoodud programmilõikudele. Looge test-klassis mitmeid klassi `Student` ja `Person` isendeid.

Meetodid. Ülekatmine

Ülesanne 2

Alamklassi isend saab kasutada ülemklassi meetodit. Koostame nüüd uued meetodid, mis on vaid klassi `Student` isenditele.

Klassis `Person` - meetod `isPersonalIdValid()` - isikukoodi pikkus on 11 sümbolit ning sisaldab ainult numbreid.

Klassis `Student` - meetod `isStudentIdValid()` - pikkus on 6 sümbolit ning sisaldab ainult numbreid.

Testige meetodite kasutamist test-klassis.