Tentti: T-106.1200 Ohjelmoinnin perusteet L

Tenttipäivä: 07.05.2008

Huom! Tämä on L-peruskurssin tentti.
Kurssilla T-106.1200 Ohjelmoinnin perusteet T on erillinen tenttipaperi.

Yleistä

Tehtävät 1 ja 2 muodostavat tentin pakollisen osan. Kummastakin näistä tehtävistä on saatava vähintään yksi piste päästäkseen läpi tentistä. Muita tehtäviä ei edes arvostella, ellei tätä pakollista osaa läpäise.

Toisaalta, jos molemmista pakollisista tehtävistä saa vähintään yhden pisteen, niin myös varmasti pääsee tentistä läpi vähintään arvosanalla 1. Arvosana määrittyy tällöin kaikkien tehtävien yhteenlasketun pistemäärän perusteella. Arvosanojen pisterajat määrätään vasta tentin jälkeen.

Tehtävä 1 (2 pistettä)

Tutustu tarkasti oheisessa liitteessä annettuihin luokkiin. Luokka `ParkingLot` kuvaa parkkipaikkakäytössä olevia alueita, joissa on kussakin tietty määrä paikkoja (*space*) autoille. Jokaisella paikalla on parkkimittari, joka osaa pitää kirjaa ajasta, jonka auto on ollut parkissa kyseisellä paikalla. Parkkimittareita kuvaa luokka `ParkingMeter` ja autoja luokka `Car`. Luokan `Test` avulla voi koekäyttää em. luokkia.

Mitä ohjelma tulostaa, kun ajetaan luokan `Test` käynnistysmetodi?

Tämän tehtävän tarkoitus on varmistaa, että jokainen kurssin läpäisevä opiskelija ymmärtää perusasiat Java-ohjelmakoodista ja ohjelman suorittamisesta. Tehtävä kuuluu tentin pakolliseen osaan ja siitä on saatava vähintään yksi piste läpäistäkseen tentin. Tehtävä arvostellaan asteikolla: 0 (hylätty) / 1 (hyväksytty) / 2 (erinomainen).

Täysiin pisteisiin riittää oikea tuloste. Kannattaa myös selittää ratkaisuasi sanallisesti, erityisesti jos et ole ihan varma ratkaisustasi ja haluat selventää ajatustenjuoksuasi tentin arvostelijalle. Järkevät perustelut voivat hyvinkin auttaa, jos tulosteeseen on lipsahtanut virhe, mutta ratkaisun perusidea on kunnossa.

Tehtävä 2 (2 pistettä)

Kirjoita Java-kielinen metodi, joka:

- saa parametrikseen merkkijonoja (`String`) sisältävän listan
- palauttaa sellaisten parametrilistassa olevien merkkijonojen lukumäärän, joiden pituus on suurempi kuin kaikkien listassa olevien merkkijonojen keskimääräinen pituus.

Esimerkiksi, jos metodin parametrina saamassa listassa on merkkijonot "a", "muu", "ananasakäämä", "lordi" ja "humppa", niin niiden pituuksien keskiarvo on 5,4. Kaksi merkkijonoista on tätä pidempi, joten metodin tulee palauttaa luku kaksi.

(Liukulukujen käytöstä johtuvia pyöristysvirheitä ei tarvitse tässä tehtävässä erikseen huomioida.)

Tämän tehtävän tarkoitus on varmistaa, että jokaisella kurssin läpäisevällä opiskelijalla on vähintäänkin auttavat taidot Java-ohjelmakoodin kirjoittamisessa. Tehtävä kuuluu tentin pakolliseen osaan ja siitä on saatava vähintään yksi piste läpäistäkseen tentin. Tehtävä arvostellaan asteikolla: 0 (hylätty) / 1 (hyväksytty) / 2 (erinomainen).

Tehtävä 3 (4 pistettä)

Esitä perusteltu mielipiteesi seuraavista Java-ohjelmointiin liittyvistä väitteistä.

Mielipide voi alkaa esimerkiksi: "Totta, koska...", "Totta, mutta...", "Ei ole mahdollista sanoa, koska...", "Ei pidä paikkaansa, joskin...", "Ei pidä paikkaansa, koska..." Perustelu on oleellinen, ei se, aloitatko vastauksesi "totta"- vai "tarua"-sanalla vai jollain muulla!

a) Väite: Java-muuttujan käyttöalueen (*scope*) määrää yksiselitteisesti se, millaista näkyvyysmäärettä (esim. `public`) sen määrittelyn yhteydessä käytetään.

b) Olkoon tutkittavana metodi:

```
public static int doStuff(int number, int another) {
  if (number < another) {
    return 1;
  } else {
    return 2 + doStuff(number - 1, another + 1);
  }
}
```

Väite: kun tätä metodia kutsutaan parametriarvoilla 6 ja 1, syntyy ns. loputon rekursio, joka johtaa ohjelman käytössä olevien muistiresurssien loppumiseen.

c) Väite: Bytecode-välikielen käyttö mahdollistaa sen, että samaa Java-virtuaalikoneohjelmaa (JVM:ää) voi käyttää Java-ohjelmien ajamiseen erilaisilla tietokoneilla.

d) Väite: tyyppimuunnoksen (*cast*) ideana on muuttaa arvon dynaaminen tyyppi (eli ajonaikainen tyyppi, *runtime type*) erilaiseksi kuin kyseisen arvon staattinen tyyppi.

Tämän ja seuraavan tehtävän ensisijainen tarkoitus on arvioida kykenetkö keskustelemaan ohjelmoinnin alkeisiin liittyvistä asioista ja esittämään perusteltuja näkemyksiään väittelyssä, jossa käytetään alan käsitteistöä ja termistöä. Kunkin tämän tehtävän kohdan arvo on 1p.

Tehtävä 4 (2 pistettä)

Kerro lyhyesti, mitä ovat tarkastettavat poikkeukset (*checked exceptions*)? Miten ne eroavat muista poikkeuksista? Käytä apuna ainakin yhtä esimerkkiä, joka osoittaa, että ymmärrät tarkastettavien poikkeusten idean.

```
1:
2:
3: public class ParkingLot {
4:
5:   private ParkingMeter[] meters;    // container
6:   private int count;                // stepper
7:
8:
9:   public ParkingLot(int size) {
10:    this.meters = new ParkingMeter[size];
11:    for (int space = 0; space < this.getSize(); space++) {  // space: stepper
12:      this.meters[space] = new ParkingMeter();
13:    }
14:    this.count = 0;
15:  }
16:
17:
18:  public int getSize() {
19:    return this.meters.length;
20:  }
21:
22:
23:  public boolean hasCar(int space) {
24:    return this.getCar(space) != null;
25:  }
26:
27:
28:  public int getCarCount() {
29:    return this.count;
30:  }
31:
32:
33:  public Car getCar(int space) {
34:    if (this.hasIndex(space)) {
35:      return this.meters[space].getCar();
36:    } else {
37:      return null;
38:    }
39:  }
40:
41:
42:  private boolean hasIndex(int space) {
43:    return space >= 0 && space < this.getSize();
44:  }
45:
46:
47:  public int park(Car car) {
48:    return this.park(car, this.findSpace());
49:  }
50:
51:
52:  private int findSpace() {
53:    for (int space = 0; space < this.getSize(); space++) {  // space: stepper
54:      if (!this.hasCar(space)) {
55:        return space;
56:      }
57:    }
58:    return -1;
59:  }
60:
61:
62:
63:
64:
65:
66:
67:
```

```
68:
69:
70:  public int park(Car car, int space) {
71:    if (this.find(car) < 0 && this.hasIndex(space) &&
72:        this.meters[space].start(car)) {
73:      this.count++;
74:      return space;
75:    } else {
76:      return -1;
77:    }
78:  }
79:
80:
81:  public int depart(Car car) {
82:    return this.depart(this.find(car));
83:  }
84:
85:
86:  private int find(Car car) {
87:    for (int space = 0; space < this.getSize(); space++) {  // space: stepper
88:      if (this.getCar(space) == car) {
89:        return space;
90:      }
91:    }
92:    return -1;
93:  }
94:
95:
96:  public int depart(int space) {
97:    if (this.hasCar(space)) {
98:      this.count--;
99:      return this.meters[space].free();
100:   } else {
101:     return -1;
102:   }
103: }
104:
105:
106:  public void advanceOneHour() {
107:    for (ParkingMeter meter : this.meters) {  // meter: most-recent holder
108:      meter.advanceHours(1);
109:    }
110:  }
111:
112:
113:  public String getDescription() {
114:    String description = "";                  // description: gatherer
115:    for (ParkingMeter meter : this.meters) {  // meter: most-recent holder
116:      description += meter.getDescription() + "\n";
117:    }
118:    return description;
119:  }
120:
121: }
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
```

```
135:
136:
137: public class ParkingMeter {
138:
139:   private Car currentCar;                // most-recent holder
140:   private int hourCount;                 // gatherer
141:
142:
143:   public ParkingMeter() {
144:     this.currentCar = null;
145:     this.hourCount = 0;
146:   }
147:
148:
149:   public boolean start(Car newCar) {
150:     if (this.currentCar == null) {
151:       this.currentCar = newCar;
152:       this.hourCount = 1;
153:       return true;
154:     } else {
155:       return false;
156:     }
157:   }
158:
159:
160:   public void advanceHours(int hours)  {
161:     if (this.currentCar != null) {
162:       this.hourCount += hours;
163:     }
164:   }
165:
166:
167:   public int free() {
168:     this.currentCar = null;
169:     int hours = this.hourCount;  // temporary
170:     this.hourCount = 0;
171:     return hours;
172:   }
173:
174:
175:   public Car getCar() {
176:     return this.currentCar;
177:   }
178:
179:
180:   public String getDescription() {
181:     if (this.getCar() == null) {
182:       return "[empty space]";
183:     } else {
184:       return this.getCar().getModel() + ", hours parked: " + this.hourCount;
185:     }
186:   }
187:
188: }
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
```

```
202:
203:
204: public class Car {
205:
206:   private String model;
207:
208:
209:   public Car(String model) {
210:     this.model = model;
211:   }
212:
213:
214:   public String getModel() {
215:     return this.model;
216:   }
217:
218: }
219:
220:
221:
222:
223:
224:
225: public class Test {
226:
227:   public static void main(String[] args) {
228:     ParkingLot parkingLot = new ParkingLot(3);
229:     Car car1 = new Car("Honda");
230:     Car car2 = new Car("Volvo");
231:     Car car3 = new Car("Mercedes");
232:     Car car4 = new Car("Saab");
233:     Car car5 = new Car("BMW");
234:
235:     System.out.println(parkingLot.getCarCount());
236:     System.out.println();
237:     System.out.println(parkingLot.park(car1, 2));
238:     System.out.println(parkingLot.park(car2));
239:     System.out.println(parkingLot.park(car3));
240:     System.out.println(parkingLot.park(car4));
241:     System.out.println(parkingLot.getDescription());
242:
243:     parkingLot.advanceOneHour();
244:     System.out.println(parkingLot.depart(0));
245:     parkingLot.advanceOneHour();
246:     System.out.println(parkingLot.depart(car2));
247:     System.out.println(parkingLot.depart(car1));
248:     System.out.println(parkingLot.park(car5));
249:     System.out.println(parkingLot.getDescription());
250:   }
251: }
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
```