Exam: T-106.1203 Basics of Programming L

Date: May 7th, 2008

Note! This is the exam for the course Basics of Programming L
There is a separate question sheet for T-106.1200 Basics of Programming T.

General Info

The questions 1 and 2 are obligatory. You must get at least one point from each of these two questions in order to pass the exam. On the other hand, if you get at least one point from each of the obligatory questions, you will pass the exam with at least a grade of 1. In this case, your exam grade will be determined by the total sum of points from all questions.

Question 1 (2 points)

Examine the classes given in the attachment. The class `ParkingLot` represents parking lots, which have a number of parking spaces. Each parking space can be used for parking a single car and is associated with a parking meter, which keeps track of how long a car has been parked at the space. Parking meters are represented by class `ParkingMeter` and cars by class `Car`. The class `Test` can be used to experiment with the other three classes.

What does the program print out when the main method in class `Test` is executed?

The purpose of this question is to make sure that every student who passes the course understands the basics of Java syntax and program execution. The question is obligatory and you must receive at least one point from it to pass the exam. This question is graded on a scale of: 0 (failed) / 1 (passed) / 2 (excellent).

Question 2 (2 points)

Write a Java method, which:

- receives as a parameter a list of strings
- returns the number of such strings in the given list of strings, whose length is greater than the average length of all strings in the given list of strings.

For example, if the parameter list contains the strings "a", "muu", "ananasakäämä", "lordi", and "humppa", then the average length is 5.4. Two of the strings are longer than that, so the method should return the number two.

The purpose of this question is to make sure that every student who passes the course has at least some basic skills in writing Java program code. The question is obligatory and you must receive at least one point from it to pass the exam. This question is graded on a scale of: 0 (failed) / 1 (passed) / 2 (excellent).

Question 3 (4 points)

State your opinion of the following claims related to Java programming. Explain your reasoning! You do not need to strictly agree or disagree with a claim – a valid justification for your opinion is the most important thing in terms of grading this question.

a) Claim: The scope of any Java variable is unambiguously defined by use of visibility modifiers (e.g. `public`) in front of the variable's definition.

b) Let us take a look at the method:

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

Claim: When this method is called with the parameter values 6 and 1, a so-called infinite recursion occurs and eventually depletes the memory resources available to the program.

c) Claim: The use of the intermediate language Bytecode makes it possible to run the same Java virtual machine program (JVM) on computers of many kinds.

d) Claim: The purpose of *casting* (also known as type casting) is to change the runtime type (also known as the dynamic type) of a value to be different from the value's static type.

The primary purpose of this question (and the following one) is to assess whether you are equipped to take part in dialogue on introductory programming topics and formulate and justify your reasoning about such matters. Each item in this question is worth one point.

Question 4 (2 points)

Briefly describe what are *checked exceptions*. How do they differ from other kinds of exceptions? Give at least one example that illustrates that you understand the concept of checked exceptions.

```
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:
```