Examination for TDA540
Object-Oriented Programming
(Objektorienterad Programmering)

Institutionen för Datavetenskap
CTH HT-18, TDA540

Day: 2019-01-19, *Time:* 14.00-18.00

Course responsible:	Jesper Cockx
Examinator:	Krasimir Angelov
Contact:	+46 729 74 49 08, +46 31 772 10 19, +46 733056925
Result:	Will be available via Ladok
Grade boundaries:	3:a 24 points 4:a 36 points 5:a 48 points max 60 points
Numbers in brackets:	Maximal number of points for each question
Allowed material:	Cay Horstmann: *Java for everyone* (2nd edition) or Jan Skansholm: *Java direkt med Swing* Markings and small footnotes are allowed. You can also use an English-Swedish dictionary if needed.
Please take note:	Write clearly and make sure to hand in your solutions in the right place. Start each question on a new piece of paper. Do not write on the back side of the paper.
Additional remarks:	The exam consists of 6 questions. Questions are not ordered by difficulty. Start by reading the whole exam before you start writing. All programs should be structured and easy to read. Don't forget to use indentation! For the correction of program code, logical flaws are considered more serious than minor syntax errors.

Good luck!

Question 1 (18 points)

Take a look at the two classes `Robot` and `ChattyRobot` below[1]:

```
1   class Robot {
2       public int position;
3       public boolean goingRight;
4
5       public Robot() {
6           position = 0;
7           goingRight = true;
8       }
9
10      public void move() {
11          if (goingRight)
12              position++;
13          else
14              position--;
15      }
16
17      public void move(int steps) {
18          for (int i = 0; i < steps; i++) {
19              move();
20          }
21      }
22
23      public void turn() {
24        goingRight = !goingRight;
25      }
26
27      public static void main(String[] args) {
28          Robot robot = new ChattyRobot();
29          int i = 0;
30          while (Math.abs(robot.position) < 3) {
31              robot.move(i);
32              robot.turn();
33              i++;
34          }
35      }
36  }
37
38  class ChattyRobot extends Robot {
39      public void move(int steps) {
40          int oldPos = this.position;
41          super.move(steps);
42          int newPos = this.position;
43          System.out.println("Moved from " + oldPos + " to " + newPos);
44      }
45  }
```

a) What is the meaning of the keyword **`public`** in line number 2? Would the code still work if we replace **`public`** with **`private`**? If not, indicate the line where an error would occur. (3 points)

[1] Line numbers are not part of the program.

b) List all *method calls* in this program (not method declarations). For each method call, give:

- The line number
- The class in which the method is declared
- The signature of the method (including the return type and the arguments)
- Whether the method is static or not

(3 points)

c) Indicate one example of *method overloading* and one example of *method overriding* in this program by giving the line numbers. (3 points)

d) What text is printed when we run the main method of this program? (6 points)

e) Change the `ChattyRobot` class so the program also prints "`turned around`" each time the `turn()` method is called. You should not change anything to the base class. (3 points)

Question 2

(4 points)

1. What is printed out when we run the following Java program? (2 points)

```
1  boolean found = true;
2  boolean past = false;
3  found = past;
4  past = true;
5  System.out.println(found);
6  System.out.println(past);
```

2. What is printed out when we run the following Java program? (2 points)

```
1  double[] list = {0.3, 0.5, 1.3, 7.8, 9};
2  double[] list2 = {9, 8.7, 3.34, 3.2};
3  list = list2;
4  list2[1] = 1.56;
5  System.out.println(list[1]);
6  System.out.println(list2[0]);
```

Question 3

(16 points)

A **magic square** is an n by n grid containing all numbers from 1 to n^2 such that the numbers on each row, column, and diagonal all add up to the same constant, called the **magic constant**. For example, the following squares are magic squares with magic constants 15 and 65:

2	7	6
9	5	1
4	3	8

21	3	4	12	25
15	17	6	19	8
10	24	13	2	16
18	7	20	9	11
1	14	22	23	5

Your task is to implement a method `boolean isMagicSquare(int[][] x)` that checks if the given 2-dimensional array is a magic square. You may assume that the given 2-dimensional array is square (so `x.length == x[i].length` for `i`$= 0, \ldots, n-1$).

Question 4

(6 points)

Take a look at the Java code below. The code in the `main` method contains 3 errors. Give for each error its line number and indicate whether it is a compile-time error or a run-time error.

```
1  class Cls {
2      static int x;
3      int y;
4  }
5
6  public class SpotTheErrors {
7      public static void main(String[] args) {
8          Cls obj = new Cls();
9          Cls.x = Cls.y;
10         Cls.x = obj.y;
11         Cls.y = Cls.x;
12         obj.y = Cls.x;
13         obj.x = obj.y;
14         obj.y = obj.x;
15         obj   = null;
16         System.out.println(obj.y);
17     }
18 }
```

Question 5

(8 points)

Implement a class `Student` that can be used for representing students at the university. Each student has a first and last name, a study program, a study year, a student number, and an average grade (as a percentage). The class should contain attributes, getters and setters, and at least one constructor. It should also override the 'toString' method so it prints all this data in a readable format. Finally, it should also implement the `Comparable` interface (given below). The `compareTo` method should compare two students according to their student number: it should return -1 / 0 / $+1$ if the student number of this student is less than / equal to / greater than the student number of the other student.

```
1  public interface Comparable<T> {
2      int compareTo(T var1);
3  }
```

Question 6 (8 points)

Take a look at the following interfaces and classes that model various kinds of animals.

```
1  interface Animal {
2    int numberOfLegs();
3  }
4
5  interface Mammal extends Animal { }
6
7  interface Reptile extends Animal { }
8
9  class Cat implements Mammal {
10     public int numberOfLegs() {
11         return 4;
12     }
13 }
14
15 class Tiger extends Cat { }
16
17 class Snake implements Reptile {
18     public int numberOfLegs() {
19         return 0;
20     }
21 }
```

1. Draw a class diagram where each type X in this hierarchy is represented by a node, and draw an arrow from X to Y if Y is a direct supertype of X. (3 points)

2. The `main` method below contains five type errors. Indicate the line number of each incorrect statement and explain in one sentence why it is wrong. (5 points)

```
1  public class Animals {
2      public static void main(String[] args) {
3          Animal animal1, animal2, animal3;
4          Mammal mammal1;
5          Cat cat1;
6          Snake snake1;
7          Tiger tiger1;
8          List<Animal> list1, list2;
9
10         animal1 = new Tiger();
11         snake1  = new Animal();
12         animal2 = new Animal();
13         cat1    = new Tiger();
14         mammal1 = new Snake();
15         tiger1  = mammal1;
16         animal3 = cat1;
17         list1   = new ArrayList<Animal>();
18         list2   = new ArrayList<Reptile>();
19     }
20 }
```