Examination for TDA540
Object-Oriented Programming
(Objektorienterad Programmering)

Institutionen för Datavetenskap
CTH HT-18, TDA540

Day: 2019-04-26, *Time:* 14.00-18.00

Course responsible:	Jesper Cockx
Examinator:	Krasimir Angelov
Contact:	+46 733 05 69 25, +46 729 74 49 08
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 (21 points)

Take a look at the program below[1]:

```java
class InsufficientBalance extends Exception {}

public class BankAccount {
    private int balance;
    public static double interestRate = 0.02;

    public BankAccount(int balance) { this.balance = balance; }

    public int getBalance() { return balance; }

    public void withdraw(int amount) throws InsufficientBalance {
        if (balance - amount < 0) throw new InsufficientBalance();
        balance -= amount;
    }

    public void addInterest() { balance += balance * interestRate; }

    public static void main(String[] args) throws InsufficientBalance {
        BankAccount acc1 = new BankAccount(5);
        BankAccount acc2 = new BankAccount(100);
        try {
            acc1.withdraw(10);
            System.out.println("Withdrew 10 SEK.");
        } catch (InsufficientBalance e) {
            System.out.println("Insufficient balance on account 1!");
            acc1.interestRate = 2.00;
        }
        acc1.addInterest();
        acc2.addInterest();
        try {
            acc2.withdraw(300);
            System.out.println("Withdrew 300 SEK.");
        } finally {
            System.out.println("Account 1 has " + acc1.getBalance());
            System.out.println("Account 2 has " + acc2.getBalance());
        }
    }
}
```

a) Is `InsufficientBalance` a checked or an unchecked exception? Explain the difference between a checked and an unchecked exception. What do you have to change to turn `InsufficientBalance` from a checked to an unchecked exception or vice versa? (4 points)

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

- The line number
- The return type of the method
- The *formal* and the *actual* parameters of the method call

(5 points)

---

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

c) What text is printed when we run the main method of this program? (4 points)

d) Explain the meaning of the keyword `static` on line 5. Would the program still compile if this keyword is removed? If yes, what would be the result of running the program? (4 points)

e) Implement a subclass of `BankAccount` called `CreditAccount` with an extra attribute `int maxCredit`, such that balance of a CreditAccount is allowed to go no lower than -maxCredit. Implement a constructor for `CreditAccount` and override any methods for which it is necessary. (4 points)

Question 2 (5 points)

A perfect square is a number which is the square of another number, such as 9 or 1024. The program below is meant to print all sums $p + q = r$ where p, q and r are perfect squares, $p \leq 20$ and $q < p$. If the program would work correctly, the output would be as follows:

```
16+9=25
64+36=100
144+25=169
144+81=225
225+64=289
256+144=400
```

However, the program contains 5 errors. For each error, give the line number and whether it is a compile-time or a run-time error.

```
1      public static void main(String[] args) {
2          int i = 1;
3          int j = 1;
4          while (i < 20) {
5              while (j < i) {
6                  int k = Math.sqrt(i*i + j*j);
7                  if (i*i + j*j = k*k) {
8                      System.Out.println("i*i + j*j = k*k");
9                  }
10                 j = j+1;
11             }
12             i = i+1;
13         }
14     }
```

Question 3 (6 points)

What is printed out when we run the `main` method of the following Java program?

```
1  public class Point {
2      public int x;
3      public int y;
4
5      public Point(int x, int y) {
6          this.x = x;
7          this.y = y;
8      }
```

```
 9
10        public Point copy() { return new Point(this.x, this.y); }
11
12        public boolean equals(Point other) {
13            return this.x == other.x && this.y == other.y;
14        }
15
16        public static void main(String[] args) {
17            Point a = new Point(2,5);
18            Point b = a;
19            b.y = 6;
20            Point c = a.copy();
21            Point d = a.copy();
22            d.x = 3;
23            System.out.println(a.equals(b));
24            System.out.println(a.equals(c));
25            System.out.println(c.equals(d));
26            System.out.println(a == b);
27            System.out.println(a == c);
28            System.out.println(c == d);
29        }
30    }
```

Question 4

In lab session 4, you implemented several transformations on images represented as matrices of integers (upDown, invert, contour, …). Another transformation that is often used is called *box blur*. For the box blur transformation, each pixel in the result takes the **average value of that pixel in the input image and all surrounding pixels**. In other words, box blur transforms the image according to the matrix

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

For example, a black pixel (value 0) surrounded by all white pixels (value 255) would get value $\frac{1}{9}(0 + 8 \cdot 255) = 227$ in the box blur of the picture. Your task is to implement a method int[][] boxBlur(int[][] image) that takes as input a representation of a greyscale image and returns the box blur of this image. You may assume the input is a square matrix.

Question 5

Design and implement two classes Donut and DonutBox.

- A Donut has a name, a type (one of "Plain", "Filled", or "Glazed"), and a price (an integer in SEK). Donut should have a method to calculate the price based on the type of the donut: 10 SEK for plain, 13 SEK for filled, 15 SEK for glazed. It should implement the toString method that prints the name, type, and price of the donut. It should also implement the equals method that returns **true** when comparing to another donut with the same name and type.

- A DonutBox contains a collection of Donuts. It should have methods to count the number of donuts in the box and to add a donut to the box. It should also have a

method to calculate the total price, which is the sum of the prices of the donuts in the box, with a discount applied (10% for ≥ 6 donuts, 20% for ≥ 12 donuts). Finally, it should implement the `toString` method that prints a list of the donuts in the box.

Additional notes:

- All attributes should be **private**.

- Implement at least one constructor for each class.

- To get full points, you should represent the different donut types as an **enum**.

Question 6 (8 points)

Take a look at the following classes that model different kinds of symbols.

```java
abstract class Symbol {
    public abstract char getCharacter();
}

class Digit extends Symbol {
    int value;
    public Digit(int i) {
        if (i < 0 || i >= 10) throw new IllegalArgumentException();
        this.value = i;
    }
    public char getCharacter() {
        return Integer.toString(value).charAt(0);
    }
}

class Letter extends Symbol {
    char value;
    public Letter(char c) {
        if (!Character.isAlphabetic(c)) throw new IllegalArgumentException();
        this.value = c;
    }
    public char getCharacter() {
        return value;
    }
    public Letter toUpper () { return new Letter(Character.toUpperCase(value)); }
    public Letter toLower () { return new Letter(Character.toLowerCase(value)); }
    public static boolean isVowel(char c) {
        return "aeiouåäö".indexOf(Character.toLowerCase(c)) != -1;
    }
}

class Vowel extends Letter {
    public Vowel(char c) {
        super(c);
        if (!isVowel(c)) throw new IllegalArgumentException();
    }
}

class Consonant extends Letter {
    public Consonant(char c) {
        super(c);
        if (isVowel(c)) throw new IllegalArgumentException();
    }
}
```

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)

```java
1    public static void main(String[] args) {
2        Symbol symbol1, symbol2, symbol3, symbol4, symbol5;
3        Digit digit;
4        Letter letter;
5        Vowel vowel;
6        Consonant consonant;
7        symbol1 = new Digit(5);
8        vowel = new Symbol('i');
9        symbol2 = new Symbol('!');
10       symbol3 = new Consonant('s');
11       consonant = symbol3;
12       vowel = new Letter('a');
13       letter = new Vowel('e');
14       symbol4 = letter;
15       consonant = new Consonant('C');
16       consonant = consonant.toLower();
17       vowel = new Vowel('u');
18       symbol5 = vowel.toUpper();
19   }
```