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]:

```
1  class InsufficientBalance extends Exception {}
2
3  public class BankAccount {
4      private int balance;
5      public static double interestRate = 0.02;
6
7      public BankAccount(int balance) { this.balance = balance; }
8
9      public int getBalance() { return balance; }
10
11     public void withdraw(int amount) throws InsufficientBalance {
12         if (balance - amount < 0) throw new InsufficientBalance();
13         balance -= amount;
14     }
15
16     public void addInterest() { balance += balance * interestRate; }
17
18     public static void main(String[] args) throws InsufficientBalance {
19         BankAccount acc1 = new BankAccount(5);
20         BankAccount acc2 = new BankAccount(100);
21         try {
22             acc1.withdraw(10);
23             System.out.println("Withdrew 10 SEK.");
24         } catch (InsufficientBalance e) {
25             System.out.println("Insufficient balance on account 1!");
26             acc1.interestRate = 2.00;
27         }
28         acc1.addInterest();
29         acc2.addInterest();
30         try {
31             acc2.withdraw(300);
32             System.out.println("Withdrew 300 SEK.");
33         } finally {
34             System.out.println("Account 1 has " + acc1.getBalance());
35             System.out.println("Account 2 has " + acc2.getBalance());
36         }
37     }
38 }
```

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) **Answer:** *It is a checked exception. Checked exceptions must be either declared or caught, while unchecked exceptions can be used without declaring them. To turn `InsufficientBalance` into an unchecked exception, you should change the superclass from `Exception` to `RuntimeException`.*

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

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

- The *formal* and the *actual* parameters of the method call

(5 points) **Answer:**

- *Line 19: return type* `BankAccount`*, formal* `int balance`*, actual* `5`
- *Line 20: return type* `BankAccount`*, formal* `int balance`*, actual* `100`
- *Line 22: return type* `void`*, formal* `int amount`*, actual* `10`
- *Line 23: return type* `void`*, formal* `String x`*, actual* `"Withdrew 10 SEK."`
- *Line 25: return type* `void`*, formal* `String x`*, actual* `"Insufficient balance on account 1!"`
- *Line 28: return type* `void`*, no arguments*
- *Line 29: return type* `void`*, no arguments*
- *Line 31: return type* `void`*, formal* `int amount`*, actual* `300`
- *Line 32: return type* `void`*, formal* `String x`*, actual* `"Withdrew 300 SEK."`
- *Line 33: return type* `void`*, formal* `String x`*, actual* `"Account 1 has " + acc1.getBalance()`
- *Line 33: return type* `int`*, no arguments*
- *Line 34: return type* `void`*, formal* `String x`*, actual* `"Account 2 has " + acc2.getBalance()`
- *Line 34: return type* `int`*, no arguments*

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

Insufficient balance on account 1!
Withdrew 300 SEK.
Account 1 has 15
Account 2 has 0

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) **Answer:** `static` *means the attribute belongs to the class rather than to a specific object. Without this keyword, the program would still compile but the withdrawal from* `acc2` *would no longer succeed.*

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) **Answer:** *See below.*

```
1   public class CreditAccount extends BankAccount {
2
3       private int maxCredit;
4
5       public CreditAccount(int maxCredit, int balance) {
6           super(balance);
7           this.maxCredit = maxCredit;
8       }
9
10      public int getMaxCredit() {
11          return maxCredit;
12      }
13
14      public void setMaxCredit(int maxCredit) {
15          this.maxCredit = maxCredit;
```

```
16        }
17
18        @Override
19        public void withdraw(int amount) throws InsufficientBalance {
20            int oldBalance = getBalance();
21            int newBalance = oldBalance - amount;
22            if (newBalance < -maxCredit) throw new InsufficientBalance();
23            setBalance(newBalance);
24
25        }
26    }
```

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      }
```

Answer:

- *Line 6: compile-time error (`sqrt` returns a float instead of an int)*
- *Line 7: compile-time error (should use == instead of =)*
- *Line 8: compile-time error (System.Out instead of System.out)*
- *Line 8: run-time error (quotes in wrong places)*
- *Line 12: run-time error (forgot to set `j = 0`).*

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  }
```

Answer:
true
true
false
true
false
false

Question 4

(12 points)

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.

Answer: *See code below.*

```
public int[][] boxBlur(int[][] image) {

  int[][] result = new int[image.length][image[0].length];

  for (int i = 0; i < image.length; i++) {
      for (int j = 0; j < image[0].length; j++) {
          int total = 0;
          int count = 0;
          for (int p = i - 1; p <= i + 1; p++) {
              for (int q = j - 1; q <= j + 1; q++) {
                  if (p >= 0 && p < image.length && q >= 0 && q < image[0].length) {
                      total += image[p][q];
                      count += 1;
                  }
              }
          }
          result[i][j] = total / count;
      }
  }
  return result;

}
```

Question 5

(8 points)

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`.

Answer: *See code below.*

```
enum Type { Plain , Filled , Glazed }

```

```java
public class Donut {

    private String name;
    private Type type;

    public Donut(String name, Type type) {
        this.name = name;
        this.type = type;
    }

    public String getName() {
        return name;
    }

    public Type getType() {
        return type;
    }

    public int getPrice() {
        if (type == Type.Plain) {
            return 10;
        } else if (type == Type.Filled) {
            return 13;
        } else {
            return 15;
        }
    }

    public String toString() {
        return name + " (" + type + ", " + Integer.toString(getPrice()) + " SEK)";
    }

    public boolean equals(Object o) {
        if (o instanceof Donut) {
            Donut other = (Donut)o;
            return this.name == other.name && this.type == other.type;
        } else {
            return false;
        }
    }
}

import java.util.HashSet;

public class DonutBox {

    private HashSet<Donut> donuts;

    public DonutBox() {
        this.donuts = new HashSet<>();
    }

    public int countDonuts(){
        return donuts.size();
    }

    public void addDonut(Donut donut) {
        donuts.add(donut);
    }

    public int getPrice() {
        int total = 0;
```

```
65          for (Donut d : donuts) {
66              total += d.getPrice();
67          }
68          double discount = 0;
69          if (countDonuts() > 12) {
70              discount = 0.2;
71          } else if (countDonuts() > 6) {
72              discount = 0.1;
73          }
74          total -= discount*total;
75          return total;
76      }
77
78      public String toString() {
79          String result = "";
80          for (Donut d : donuts) {
81              result += d.toString() + " ";
82          }
83          return result;
84      }
85
86  }
```

Question 6

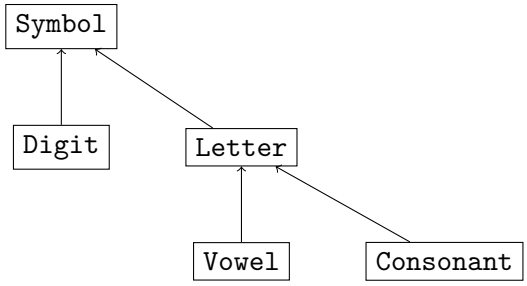
(8 points)

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

```
1   abstract class Symbol {
2       public abstract char getCharacter();
3   }
4
5   class Digit extends Symbol {
6       int value;
7       public Digit(int i) {
8           if (i < 0 || i >= 10) throw new IllegalArgumentException();
9           this.value = i;
10      }
11      public char getCharacter() {
12          return Integer.toString(value).charAt(0);
13      }
14  }
15
16  class Letter extends Symbol {
17      char value;
18      public Letter(char c) {
19          if (!Character.isAlphabetic(c)) throw new IllegalArgumentException();
20          this.value = c;
21      }
22      public char getCharacter() {
23          return value;
24      }
25      public Letter toUpper () { return new Letter(Character.toUpperCase(value)); }
26      public Letter toLower () { return new Letter(Character.toLowerCase(value)); }
27      public static boolean isVowel(char c) {
28          return "aeiouåäö".indexOf(Character.toLowerCase(c)) != -1;
29      }
30  }
31
32  class Vowel extends Letter {
33      public Vowel(char c) {
```

```
34          super(c);
35          if (!isVowel(c)) throw new IllegalArgumentException();
36      }
37  }
38
39  class Consonant extends Letter {
40      public Consonant(char c) {
41          super(c);
42          if (isVowel(c)) throw new IllegalArgumentException();
43      }
44  }
```

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) **Answer:**



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 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      }
```

Answer:

- *Line 8: cannot create new `Symbol` since class is abstract / `Symbol` is not a subclass of `Vowel`*
- *Line 9: cannot create new `Symbol` since class is abstract*
- *Line 11: `Symbol` is not a subclass of `Consonant`*
- *Line 12: `Letter` is not a subclass of `Vowel`*
- *Line 16: `toLower` returns a `Letter`, which is not a subclass of `Consonant`*