Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology

Göteborg 16 December 2019

COMPUTER PROGRAMMING part A TIN213

Date: 16 December 2019 Time: 08.30-11.30 Place: SB Multi Hall

---

Course responsible:	Anton Ekblad, tel. 070 7579 070 Will visit hall at 09.00 and 11.00
Examiner:	Krasimir Angelov
Allowed aids:	Skansholm, *Java Direkt med Swing* **or** Bravaco, Simonson, *Java Programming: From the Ground Up* (Underlinings and light annotations are permitted.) No calculators are permitted.
Grading scale:	Maximum total 30 points For this exam the following grades will be given: 3: 15 points, 4: 20 points, 5: 25 points
Exam review:	date **2020-01-29, 12:00-13:00** room **EDIT 6110**

- Read through the whole exam before answering any questions.
- Start each new question on a new page.
- Write your anonymous code and the question number on each page.
- You may write your answers in English or Swedish.
- Write clearly. Unreadable = wrong!
- Try to answer all the questions. Partial answers can still get partial credit. There are 4 questions.
- Points may be deducted for solutions which are unnecessarily complex or uses poor programming practices.
- A quick reference guide to Java is included, starting on page 6.

Good luck!

1 Matchstick Game

In the game of *matchsticks*, two players take turns removing 1, 2 or 3 matches from a row of matches on a table. Initially, there are 27 matches on the table. The player who removes the last match loses the game.

(a) **Your task:** write a class `Matches` which models the matchstick game. The class must have the following public constructors and methods:

- A constructor `Matches(int matches, int maxPick)`, which sets the initial number of matches on the table to `matches`, and the maximum number of matches a player can take during their turn to `maxPick` (the "max pick" game rule).
- A parameterless constructor `Matches()` which sets the number of matches and the max pick game rule to 27 and 3 respectively (i.e. the default rules of the game).
- A method `int getPlayer()` which returns 1 if it's player 1's turn to remove matches, or 2 if it's player 2's turn.
- A method `int getWinner()`, which returns 1 if player 1 has won the game, 2 if player 2 has won the game, or 0 is the game is still on.
- A method `int getMaxPick()`, which returns the maximum number of matches a player may remove during their turn. This method depends on both the max pick rule and on how many matches are left on the table: if there are only two matches left, `getMaxPick` should return at most 2, even if the maximum pick according to the game rules is 3.
- A method `void remove(int n)` which removes `n` matches from the table on behalf of the current player. It should then switch the turn to the other player. Repeatedly calling `remove(1)` should result in player 1 removing one match, then player 2 removing one, etc., until all matches are gone and the player who took the last turn loses the game.
- A method `String toString()` which returns a string representation of the current game state, giving the current player as well as a visual representation of the matches on the table. Use one pipe (`|`) character to represent each match still on the table. Fig. 1 shows the output of `toString()` called on a newly created `Matches` object.

```
|||||||||||||||||||||||||||
Next up: Player 1
```

Figure 1: A game with 27 matches and player 1 about to take a turn.

You may add any fields and (private) helper methods you need to implement the above specification. For full marks, your solution should respect the DRY principle: Don't Repeat Yourself. (6 points)

(b) **Your task:** write a static `main` method, which uses your `Matches` class to let two users play a game of matchsticks. Use `Scanner` for input, and `System.out.println` for output. For full marks, your main method should not allow players to break the rules of the game (i.e. by removing too few or too many matches), instead asking them to re-enter the number of matches to remove if they attempt to remove too few or too many. (4 points)

10 points total

2 Reading Code

(a) Consider the following code:

```
public static boolean p(String s) {
  for(int i = 0; i < s.length(); i++) {
    if(s.charAt(i) != s.charAt(s.length()-i-1)) {
      return false;
    }
  }
  return true;
}
```

Your task: explain briefly the purpose of the method `p`, and suggest a more descriptive name for it. (2 points)

(b) Consider the following code:

```
1  public static int foo(int x) {
2    if(x % 7 == 2) {
3      throw new IllegalArgumentException();
4    }
5    return x+x;
6  }
7
8  public static double bar(String s) {
9    double d = 2;
10   try {
11     d += foo(s.length())/2;
12   } catch (ArithmeticException e) {
13     System.out.println("foo");
14   } finally {
15     System.out.println("bar");
16   }
17   return d+1;
18 }
19
20 public static void main(String[] args) {
21   try {
22     System.out.println(bar("Merry Christmas!"));
23     System.out.println("And a Happy New Year!");
24   } catch (IllegalArgumentException e) {
25     System.out.println("baz");
26   }
27 }
```

Your task: what will the program print when executed? Briefly motivate your answer. (2 points)

(c) **Your task:** what would the program from part (b) print if we were to change line 22 to `System.out.println(bar("God Jul!"));`? Briefly motivate your answer. (2 points)

6 points total

3 The Door

Your employer has recently gotten into the business of selling electronic doors. To save on costs, your boss has decided to buy the absolute cheapest door they can find, and have your team add the logic to turn this "stupid" door into a useful product. The stupid door you are working with only exposes a single method:

```
StupidDoor.setOpen(boolean open);
```

This method controls the physical state of the door: calling `StupidDoor.setOpen(true)` will open the door and leave it open until further notice, and `StupidDoor.setOpen(false)` will close it.

Unfortunately, your coworker, who was in charge of the project, quit before completing the task, and Fig. 2 shows the code they left behind. This code is in a very poor state, and does not fullfil the specification for the door control program as given to you by your boss.

The specification for the electronic door control program is as follows:

- The door is always in exactly one of the following states: locked, unlocked (but closed), or open.
- The door can only be locked and opened when it is unlocked and closed.
- The door can only be unlocked when it is locked.
- When opening the door, a number of seconds to keep the door open must be specified. For health and safety reasons the door must never be open for less than 10 seconds, and for security reasons it must never be open for more than 60 seconds. The method `DoorUtils.sleep(int secs)` is used to pause execution of the calling method for `secs` seconds.
- The door requires a PIN to unlock. The PIN to unlock the door must be passed as an argument to the door's constructor on startup. It must be impossible to change the PIN of a `SmartDoor` object that has already been constructed.
- Attempting to unlock the door with the wrong PIN must result in a `SecurityException` being thrown.
- If an argument passed to the class' constructor or methods would cause this specification to be violated, *regardless of the door's internal state*, an `IllegalArgumentException` must be thrown.
- If a method would cause a violation of this specification when called *while the door is in a particular state*, but might be permissible in some other state, an `IllegalStateException` must be thrown.

```
1  public class SmartDoor {
2    private boolean locked = false;
3    private boolean open = false;
4    private int pin;
5
6    public SmartDoor(int pin) {
7      this.pin = pin;
8    }
9
10   public void lock() {
11     this.locked = true;
12   }
13
14   public void unlock(int pin) {
15     if(this.pin == pin) {
16       this.locked = false;
17     }
18   }
19
20   public void open(int seconds) {
21     if(!this.locked) {
22       StupidDoor.setOpen(true);
23       DoorUtils.sleep(seconds);
24       StupidDoor.setOpen(false);
25     }
26   }
27 }
```

Figure 2: The unfinished door code.

Your task: modify the class given in Fig. 2 so that it satisfies the specification. For full marks, your solution must do its utmost to ensure that the specification can never be violated, even by programmer mistake.

7 points

4 Bubble Sort

Bubble sort is an algorithm which sorts a sequence of elements in ascending order. The algorithm is defined as follows:

1. Let the algorithm's input be a sequence of elements $a_{1..k}$ of length k.
2. For each element a_i in a_1, a_2 ... a_{k-1}:
   2.1. Compare a_i to a_{i+1}.
   2.2. If a_i is larger than a_{i+1} the sequence is unsorted, so let the elements a_i and a_{i+1} swap places with each other.
3. Repeat step 2 until the whole sequence a_i is sorted, i.e. all of step 2. is executed without any elements changing place.

(a) **Your task:** write a method `void sort(int[] arr)` which uses the bubble sort algorithm to sort the array `arr` in ascending order. (5 points)

(b) Recall that unnecessarily *mutating* values can often lead to bugs which are surprisingly hard to track down. Unfortunately, mutation is often crucial for performance. A common trick to write code which is both safe and fast is to use mutation *locally*, while presenting an immutable, safe interface to the user.

Your task: write a method `int[] safeSort(int[] arr)` which uses your `sort` method from part (a) to return an array containing all values from `arr` (including duplicates), sorted in ascending order, *without modifying `arr` itself.* (2 points)

7 points total

Java Quick Reference Guide

User Input and Output Java applications and applets can get input and output through the console (command window) or through dialogue boxes as follows:

```
System.out.println("This is displayed on the console");

Scanner scanner = new Scanner(System.in);
String input = scanner.nextLine();
int n = scanner.nextInt();

import javax.swing.*;
JOptionPane.showMessageDialog(null,
  "This is displayed in a dialogue box");

String input = JOptionPane.showInputDialog("Enter a string");
```

Data Types

`boolean`	Boolean type, can be `true` or `false`
`byte`	1-byte signed integer
`char`	Unicode character
`short`	2-byte signed integer
`int`	4-byte signed integer
`long`	8-byte signed integer
`float`	Single-precision fraction, 6 significant figures
`double`	Double-precision fraction, 15 significant figures

Operators

`+ - * / %`	Arithmetic operators (`%` means *remainder*)
`++ --`	Increment of decrement by 1 `result = ++i;` means increment by 1 first `result = i++;` means do the assignment first
`+= -= *= /= %=` etc.	E.g. `i+=2` is equivalent to `i = i + 2`
`&&`	Logical AND, e.g. `if (i > 50 && i < 70)`
`||`	Logical OR, e.g. `if (i < 0 || i > 100)`
`!`	Logical NOT, e.g. `if (!endOfFile)`
`== != > >= < <=`	Relational operators

Control Flow - `if ... else` `if` statements are formed as follows (the `else` clause is optional).

```
String dayname;
...
if (dayname.equals("Sat") || dayname.equals("Sun")) {
  System.out.println("Hooray for the weekend");
}
else if (dayname.equals("Mon")) {
  System.out.println("I don't like Mondays");
}
else {
  System.out.println("Not long for the weekend!");
}
```

Control Flow - Loops Java contains three loop mechanisms:

```
int i = 0;
while (i < 100) {
  System.out.println("Next square is: " +  i*i);
  i++;
}

for (int i = 0; i < 100; i++) {
  System.out.println("Next square is: " +  i*i);
}

int positiveValue;
do {
  positiveValue = getNumFromUser();
}
while (positiveValue < 0);
```

Defining Classes When you define a class, you define the data attributes (usually `private`) and the methods (usually `public`) for a new data type. The class definition is placed in a `.java` file as follows:

```
// This file is Student.java. The class is declared
// public, so that it can be used anywhere in the program
public class Student {
  private String name;
  private int    numCourses;

  // Constructor to initialize all the data members
  public Student(String name, int numCourses) {
    this.name = name;
    this.numCourses = numCourses;
  }

  // No-arg constructor, to initialize with defaults
  public Student() {
    this("Anon", 0);       // Call other constructor
  }

  // Other methods
  public void attendCourse() {
    this.numCourses++;
  }
}
```

To create an object and send messages to the object:

```
public class MyTestClass {
  public static void main(String[] args) {
    // Step 1 - Declare object references
    // These refer to null initially in this example
    Student me, you;

    // Step 2 - Create new Student objects
    me  = new Student("Andy", 0);
    you = new Student();

    // Step 3 - Use the Student objects
```

```
    me.attendCourse();
    you.attendCourse()
  }
}
```

Arrays An array behaves like an object. Arrays are created and manipulated as follows:

```
// Step 1 - Declare a reference to an array
int[] squares;          // Could write int squares[];

// Step 2 - Create the array "object" itself
squares = new int[5];

// Creates array with 5 slots
// Step 3 - Initialize slots in the array
for (int i=0; i < squares.length; i++) {
  squares[i] = i * i;
  System.out.println(squares[i]);
}
```

Note that array elements start at [0], and that arrays have a `length` property that gives you the size of the array. If you inadvertently exceed an array's bounds, an exception is thrown at run time and the program aborts.

Note: Arrays can also be set up using the following abbreviated syntax:

```
int[] primes = {2, 3, 5, 7, 11};
```

Static Variables A `static` variable is like a global variable for a class. In other words, you only get one instance of the variable for the whole class, regardless of how many objects exist. `static` variables are declared in the class as follows:

```
public class Account {
  private String accnum; // Instance var
  private double balance = 0.0; // Instance var
  private static double intRate = 5.0;  // Class var
  ...
}
```

Static Methods A `static` method in a class is one that can only access `static` items; it cannot access any non-`static` data or methods. `static` methods are defined in the class as follows:

```
public class Account {
  public static void setIntRate(double newRate) {
    intRate = newRate;
  }

  public static double getIntRate() {
    return intRate;
  }
  ...
}
```

To invoke a `static` method, use the name of the class as follows:

```
public class MyTestClass {
  public static void main(String[] args) {
    System.out.println("Interest rate is" +
```

```
      Account.getIntRate());
  }
}
```

Exception Handling Exception handling is achieved through five keywords in Java:

`try` Statements that could cause an exception are placed in a '`try`' block

`catch` The block of code where error processing is placed

`finally` An optional block of code after a '`try`' block, for unconditional execution

`throw` Used in the low-level code to generate, or 'throw' an exception

`throws` Specifies the list of exceptions a method may throw

Here are some examples:

```
public class MyClass {
  public void anyMethod() {
    try {
      func1();
      func2();
      func3();
    }
    catch (IOException e) {
      System.out.println("IOException:" + e);
    }
    catch (MalformedURLException e) {
      System.out.println("MalformedURLException:" + e);
    }
    finally {
      System.out.println("This is always displayed");
    }
  }

  public void func1() throws IOException {
    ...
  }

  public void func2() throws MalformedURLException {
    ...
  }

  public void func3() throws IOException, MalformedURLException {
    ...
  }
}
```

Answers for TIN213 exam 2019-12-16

Question 1: Matchsticks

```
import java.util.Scanner;

public class Matches {
    //////////////////
    //   Part (a)   //
    //////////////////

    private int matches;
    private final int maxPick;
    private int currentPlayer;

    // Private helper method
    private int nextPlayer() {
        return this.currentPlayer == 1 ? 2 : 1;
    }

    public Matches(int matches, int maxPick) {
        this.currentPlayer = 1;
        this.matches = matches;
        this.maxPick = maxPick;
    }

    public Matches() {
        this(27, 3);
    }

    public int getPlayer() {
        return this.currentPlayer;
    }

    public int getMaxPick() {
        return Math.min(this.matches, this.maxPick);
    }

    public int getWinner() {
        if(this.matches > 0) {
            return 0;
        } else {
            return this.getPlayer();
        }
    }

    public void remove(int matches) {
        this.matches -= matches;
        this.currentPlayer = this.nextPlayer();
```

```
    }

    public String toString() {
        String s = "";
        for(int i = 0; i < this.matches; i++) {
            s = s + "|";
        }
        s += "\nNext up: Player " + this.getPlayer();
        return s;
    }



    //////////////////
    //   Part (b)   //
    //////////////////

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        Matches game = new Matches();

        while(game.getWinner() == 0) {
            System.out.println(game.toString());
            game.remove(promptMatches(scan, game));
            System.out.println();
        }
        System.out.println("Winner: Player " + game.getWinner());
    }

    // Part b: input helper for main method
    private static int promptMatches(Scanner scan, Matches game) {
        int matches;
        System.out.print("Take 1 to " + game.getMaxPick() + " matches: ");
        matches = scan.nextInt();
        while(matches < 1 || matches > game.getMaxPick()) {
            System.out.print("Bad number of matches, try again: ");
            matches = scan.nextInt();
        }
        return matches;
    }
}
```

Question 2: Reading Code

a. The method returns true if its input is a palindrome, so a good name for it would be `isPalindrome`. b. bar baz c. bar 11.0 And a Happy New Year!

Question 3: The Door

```
public class Door {
    private boolean locked = false;
    private boolean open = false;
    private final int pin;

    public Door(int pin) {
        this.pin = pin;
    }

    public void lock() {
        if(this.open || this.locked) {
            throw new IllegalStateException();
        }
        this.locked = true;
    }

    public void unlock(int pin) {
        if(this.pin != pin) {
            throw new SecurityException();
        }
        if(!this.locked || this.open) {
            throw new IllegalStateException();
        }
        this.locked = false;
    }

    public void open(int seconds) {
        if(this.locked || this.open) {
            throw new IllegalStateException();
        }
        if(seconds < 10 || seconds > 60) {
            throw new IllegalArgumentException();
        }
        this.open = true;
        StupidDoor.setOpen(true);
        DoorUtils.sleep(seconds);
        StupidDoor.setOpen(false);
        this.open = false;
    }
}
```

Question 4: Bubble Sort

```
public class Sort {
    // Part (a)
    public static void sort(int[] a) {
        boolean sorted;
        do {
            sorted = true;
            for(int i = 0; i < a.length-1; i++) {
                if(a[i] > a[i+1]) {
                    int tmp = a[i];
                    a[i] = a[i+1];
                    a[i+1] = tmp;
                    sorted = false;
                }
            }
        } while(!sorted);
    }

    // Part (b)
    public static int[] safeSort(int[] a) {
        int[] b = new int[a.length];
        for(int i = 0; i < a.length; i++) {
            b[i] = a[i];
        }
        sort(b);
        return b;
    }
}
```