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

Göteborg 26 April 2019

COMPUTER PROGRAMMING part A TIN213

Date: 26 April 2019 Time: 08.30-11.30 Place: "Maskin"-salar

Course responsible:	Robin Adams, tel. 076 856 48 64 Will visit hall at 09.00 and 11.00
Examiner:	Robin Adams
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:	Maxmimum total 30 points For this exam the following grades will be given: 3: 15 points, 4: 20 points, 5: 25 points
Exam review:	Monday 27 May 2019 14.00–16.00 EDIT 6466

- Answer all the questions. There are four (4) 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.
- A quick reference guide to Java is included, starting on page 5.

Good luck!

1. I am writing my own program to keep track of the marks of every student in this course. Write a class `Student` that will be part of this program. An instance of `Student` will represent one student taking the course.

   The class should have:

   - a constructor that takes a student's name and personnummer
   - a method `public void examA(int points)` that is called after the student completes an exam for part A. The method should throw an `IllegalArgumentException` if it is called and `points` is not in the range 0–30.
   - a method `public void examB(int points)` that is called after the student completes an exam for part B. The method should throw an `IllegalArgumentException` if it is called and `points` is not in the range 0–30.
   - a method `public void completeLabs()` that is called when the student completes all the laborations.
   - a method `public int getGrade()` that returns their final grade for this course (3, 4 or 5), or 0 if their final grade is U.
   - a method `public String toString()` that returns a string containing the information about the student in the following format:

   ```
   Alice Andersson (199811051234)  Part A: 3 (17/30)  Part B: 5 (27/30)  Labs: G
   Final grade: 4
   ```

   When a student sits a re-exam (omtenta), the method examA or examB is called again with the new score. The values returned by `getGrade` and `toString` should be based on the highest mark achieved by the student. So, for example, if a student sits the part A exam three times and scores 18, 23, 21, then `toString` should return a string including "Part A: 4 (23/30)".

   An exam always has a maximum of 30 points, with grades assigned as follows. For grade 3: 15 points, grade 4: 20 points, grade 5: 25 points.

   If the student has not yet passed both exams and completed the labs, then their final grade is U. Otherwise, it is the average of the number grades for the two parts, rounded down. So, for example, if a student achieves a grade 4 on part A and grade 5 on part B, their final grade is 4.

   When the class is finished, the following code should produce the output given on the following page.

   ```
   public class StudentMain {
       public static void main(String[] args) {
           Student alice = new Student("Alice Andersson", 199811051234L);

           alice.examA(12);
           System.out.println(alice);

           alice.completeLabs();
           alice.examB(19);
           System.out.println(alice);

           alice.examA(26);
           System.out.println(alice);
       }
   }
   ```

Output:

```
Alice Andersson (199811051234) Part A: U (12/30) Part B: U (0/30) Labs: U
Final grade: U
Alice Andersson (199811051234) Part A: U (12/30) Part B: 3 (19/30) Labs: G
Final grade: U
Alice Andersson (199811051234) Part A: 5 (26/30) Part B: 3 (19/30) Labs: G
Final grade: 4
```

(12 points)

2. What will be the output from the following program? (3 points)

```
class MyClass {
    private int value;

    public MyClass() {
        this.value = 0;
    }

    public MyClass(MyClass that) {
        this.value = that.value;
    }

    public void setValue(int newValue) {
        this.value = newValue;
    }

    @Override
    public String toString() {
        return "The value is: " + this.value;
    }
}

public class Main {

    public static void main(String[] args) {
        MyClass a = new MyClass();
        MyClass b = a;
        MyClass c = new MyClass(a);

        a.setValue(3);
        b.setValue(5);
        c.setValue(7);

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

3. A string is *k-periodic* if it is a repetition of a string of length k. For example, the string *abcabcabc* is 3-periodic. The string *xyxyxyxy* is 2-periodic and 4-periodic. The string *abcd* is not 1-perodic, 2-periodic or 3-periodic, but is 4-periodic.

   (a) Write a class method `static boolean isPrefix(String str, int i, int k)` which, when called with a non-empty string `str`, a non-negative integer `i` and a positive integer `k`, returns `true` if the substring of `str` starting at character 0 of length `k` is equal to the substring of `str` starting at character `i` of length `k` and `false` otherwise. If `str` has length < `i+k`, the method should returns `false`. (3 points)

   (b) Write a class method `static boolean isPeriodic(String str, int k)` which, when called with a non-empty string `str` and positive integer `k`, returns `true` if `str` is `k`-periodic, and `false` if not. (You may use the method `isPrefix` in this method, but you do not have to.) (4 points)

   (7 points total)

4. We say that an array A is a *subarray* of an array B if the elements of A all occur in B adjacent to each other in the same order. Thus for example $\{3, 0, 5, 1\}$ is a subarray of $\{2, 3, 0, 5, 1, 1, 2\}$ but $\{2, 4, 5\}$ is not a subarray of $\{1, 2, 3, 4, 5\}$. Every array is a subarray of itself, and the empty array is a subarray of every array.

   Write a class method `static boolean isSubArray(int[] a, int[] b)` that returns `true` if `a` is a subarray of `b`, and `false` if not.

   Note: For a maximum score on this question, your solution should be 'fast', i.e. should not read the values of `b` more than once. A 'slow' solution will score a maximum of 5 points. (8 points)

Java Quick Reference Guide

User Input and Output Java applications 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 dont 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 {
    ...
  }
}
```

(Quick Reference Guide adapted from `https://web.fe.up.pt/∼aaguiar/teaching/pc/`.)

1.

```
public class Student {
    private String name;
    private long personnummer;
    private int examAscore;
    private int examBscore;
    private boolean labsCompleted;

    public Student(String name, long personnummer) {
        this.name = name;
        this.personnummer = personnummer;
        examAscore = 0;
        examBscore = 0;
        labsCompleted = false;
    }

    public void examA(int points) {
        if (points < 0 || points > 30) {
            throw new IllegalArgumentException("Invalid score for exam A: "
+ points);
        }
        if (points > examAscore) {
            examAscore = points;
        }
    }

    public void examB(int points) {
        if (points < 0 || points > 30) {
            throw new IllegalArgumentException("Invalid score for exam B: "
+ points);
        }
        if (points > examBscore) {
            examBscore = points;
        }
    }

    public void completeLabs() {
        labsCompleted = true;
    }

    public int getGrade() {
        int examAgrade = scoreToGrade(examAscore);
        int examBgrade = scoreToGrade(examBscore);
        if (examAgrade == 0 || examBgrade == 0 || ! labsCompleted) {
            return 0;
        } else {
            return (examAgrade + examBgrade) / 2;
        }
    }

    public String toString() {
        return name + " (" + personnummer + ")  "
                + "Part A: " + gradeToString(scoreToGrade(examAscore)) + "
(" + examAscore + "/30)  "
                + "Part B: " + gradeToString(scoreToGrade(examBscore)) + "
(" + examBscore + "/30)  "
                + "Labs: " + (labsCompleted ? "G" : "U") + "\n"
                + "Final grade: " + gradeToString(getGrade());
    }
    private int scoreToGrade(int points) {
        if (points < 15) {
            return 0;
        } else if (points < 20) {
            return 3;
        } else if (points < 25) {
            return 4;
        } else {
            return 5;
        }
    }

    private String gradeToString(int grade) {
        if (grade == 0) {
            return "U";
        } else {
            return Integer.toString(grade);
        }
    }
}
```

2.

```
The value is: 5
The value is: 5
The value is: 7
```

3.

```
static boolean isPrefix(String str, int i, int k) {
    if (str.length() < i + k) {
        return false;
    }
    return str.substring(0, k).equals(str.substring(i, i+k));
}

static boolean isPeriodic(String str, int k) {
    for (int i = 0; i < str.length(); i += k) {
        if (! isPrefix(str, i, k)) {
            return false;
        }
    }
    return true;
}
```

4.

A slow solution:

```
    static boolean isSubArray(int[] a, int[] b) {
        for (int i = 0; i < b.length - a.length; i++) {
            boolean foundSubArray = true;
            for (int j = 0; j < a.length; j++) {
                if (a[j] != b[i+j]) {
                    foundSubArray = false;
                }
            }
            if (foundSubArray) {
                return true;
            }
        }
        return false;
    }
```

A fast solution:

```
static boolean isSubArray(int[] a, int[] b) {
        // Two pointers to traverse the arrays
        int i = 0, j = 0;

        // Traverse both arrays simultaneously
        while (i < a.length && j < b.length) {

            // If element matches
            // increment both pointers
            if (a[i] == b[j]) {
                i++;
                j++;

                // If array b is completely
                // traversed
                if (j == b.length) {
                    return true;
                }
            }
            // If not,
            // increment i and reset j
            else {
                i++;
                j = 0;
            }
        }

        return false;
    }
```