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

Göteborg 17 December 2018

COMPUTER PROGRAMMING part A TIN213

Date: 17 December 2018 Time: 08.30-11.30 Place: SB Multi Hall

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:	Tuesday 29 January 2019 09.00-11.00 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. A positive integer n is called a *perfect number* if n is equal to the sum of all its proper factors (i.e. all the factors of n that are not equal to n). For example, 28 is perfect because its factors are 1, 2, 4, 7, 14, 28; and

$$28 = 1 + 2 + 4 + 7 + 14 \ .$$

   (a) Write a class method `private static int sumOfFactors(int n)` which, when given a positive integer `n`, returns the sum of all the proper factors of `n`. (3 points)

   (b) Write a class method `private static boolean isPerfect(int n)` which, when given a positive integer `n`, returns `true` if `n` is a perfect number and `false` if `n` is not. (You method may call the method `sumOfFactors` from part 1a.) (2 ponts)

   (c) Write the `main` method of a program that asks the user for an integer. If they enter a positive integer n, the program prints out a list of all the perfect numbers from 1 to n. Your program may use the class methods that you wrote in parts 1a and 1b. (2 points)

   (d) Now write a new `main` method. The program should ask the user for a positive integer n, then print out all the perfect numbers from 1 to n together with their proper factors, in the following format. If the user enters the integer 1000, for example, the program should output the following:

```
6 = 1 + 2 + 3
28 = 1 + 2 + 4 + 7 + 14
496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248
```

   Your program may use the class methods that you wrote in parts 1a and 1b. (4 points)

(11 points total)

2. Write a class `TableTennis` that describes keeps track of the score in a game of table tennis (*bordtennis*).

   The scoring rules for table tennis are as follows:

   - When a player wins a serve, he or she scores 1 point.
   - If one player reaches 11 points and the other player has 9 or fewer, then the player with 11 points wins.
   - If the score becomes 10-10, this is known as *deuce* in English. After that, the first player to score 2 more points than the other player wins.

   The class should have:

   - instance variables for the two players' names and their scores, and a boolean instance variable `deuce` that denotes whether the score has ever been 10-10.
   - a constructor that takes the names of the two players as parameters, and sets the values of all the instance variables as appropriate for the start of a game.
   - four 'getter' methods for instance variables called `getPlayerOneName`, `getPlayerTwoName`, `getPlayerOneScore` and `getPlayerTwoScore`.
   - a method
`public void scoreOne()`
that is called when player one scores a point.
   - a method
`public void scoreTwo()`
that is called when player two scores a point.
   - a method
`public String toString()`
that returns a string displaying the current state of the game in the following format:
`Falck Mattias:  7 Karlsson Kristian:  9`
   - a method
`public int winner()`
that should return 1 if player one has won, 2 if player two has won, or 0 if the game is not yet over.

   (9 points)

3. I am trying to solve the following problem.

   > Let A be the point $(1,1)$, B be the point $(3,1)$ and C be the point $(1,3)$. Let D be the midpoint of A and B, and E the midpoint of A and C. What are the coordinates of D and E?

   I have written the following code which I think should answer the problem.

```
public class Point {
  private double x;
  private double y;

  public Point(double x, double y) {
      this.x = x;
      this.y = y;
  }

  public String toString() {
      return String.format("(%.1f, %.1f)", x, y);
  }

  public Point midPoint(Point p) {
      this.x = (this.x + p.x) / 2;
      this.y = (this.y + p.y) / 2;
      return new Point(this.x, this.y);
  }

  public static void main(String[] args) {
      Point pointA = new Point(1, 1);
      Point pointB = new Point(3, 1);
      Point pointC = new Point(1, 3);
      Point pointD = pointA.midPoint(pointB);
      Point pointE = pointA.midPoint(pointC);
      System.out.println("Point D is " + pointD);
      System.out.println("Point E is " + pointE);
  }
}
```

   However, to my surprise, the program produces the following output.

```
Point D is (2.0, 1.0)
Point E is (1.5, 2.0)
```

   I am sure this is not the right answer!

   How should I change my program to fix the bug?

   (3 points)

4. Write a method `public int missingElement(int[] a)`. The method takes an array `a` of length 99 which contains all the integers from 1 to 100 (not necessarily in order), except one number is missing. The method should return the value of the missing number.

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

   (7 points)

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 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 {
    ...
  }
}
```

1. a)

```
    private static int sumOfFactors(int n) {
        int sum = 0;
        for (int i = 1; i < n; i++) {
            if (n % i == 0) {
                sum += i;
            }
        }
        return sum;
    }
```

1. b)

```
    private static boolean isPerfect(int n) {
        return (n == sumOfFactors(n));
    }
```

1. c)

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Please enter a positive integer");
        int n = scanner.nextInt();
        for (int i = 1; i <= n; i++) {
            if (isPerfect(i)) {
                System.out.println(i);
            }
        }
    }
```

1. d)

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Please enter a positive integer");
        int n = scanner.nextInt();
        for (int i = 1; i <= n; i++) {
            if (isPerfect(i)) {
                System.out.printf("%d = 1", i);
                for (int j = 2; j < i; j++) {
                    if (i % j == 0) {
                        System.out.printf(" + %d", j);
                    }
                }
                System.out.println();
            }
        }
    }
```

2.

```
public class TableTennis {
    private String playerOneName;
    private String playerTwoName;
    private int playerOneScore;
    private int playerTwoScore;
    private boolean deuce;

    public TableTennis(String playerOneName, String playerTwoName) {
        this.playerOneName = playerOneName;
        this.playerTwoName = playerTwoName;
        this.playerOneScore = 0;
        this.playerTwoScore = 0;
        this.deuce = false;
    }

    public String getPlayerOneName() {
        return this.playerOneName;
    }

    public String getPlayerTwoName() {
        return this.playerTwoName;
    }

    public int getPlayerOneScore() {
        return this.playerOneScore;
    }

    public int getPlayerTwoScore() {
        return this.playerTwoScore;
    }

    public void scoreOne() {
        this.playerOneScore++;
        if (this.playerOneScore == 10 && this.playerTwoScore == 10) {
            this.deuce = true;
        }
    }

    public void scoreTwo() {
        this.playerTwoScore++;
        if (this.playerOneScore == 10 && this.playerTwoScore == 10) {
            this.deuce = true;
        }
    }

    @Override
    public String toString() {
        return String.format("%s: %2d %s: %2d", this.playerOneName,
this.playerOneScore, this.playerTwoName, this.playerTwoScore);
    }

    public int winner() {
        if (!this.deuce && this.playerOneScore > 10) {
            return 1;
        }
        if (!this.deuce && this.playerTwoScore > 10) {
            return 2;
        }
        if (this.deuce && this.playerOneScore >= this.playerTwoScore + 2) {
            return 1;
        }
        if (this.deuce && this.playerTwoScore >= this.playerOneScore + 2) {
            return 2;
        }
        return 0;
    }
}
```

3. Change the midPoint method to:

```
public Point midPoint(Point p) {
  double x = (this.x + p.x) / 2;
  double y = (this.y + p.y) / 2;
  return new Point(x, y);
}
```

4. A slow solution:

```
   public int missingElement(int[] a) {
        for (int i = 1; i <= 100; i++) {
            if (isMissing(a, i)) {
                return i;
            }
        }
    }

    private boolean isMissing(int[] a, int i) {
        for (int j : a) {
            if (j == i) {
                return false;
            }
        }
        return true;
    }
```

A fast solution:

```
    public int missingElement(int[] a) {
        int sum = 0;
        for (int j : a) {
            sum += j;
        }
        return 5050 - sum;
    }
```