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

Göteborg 23 August 2019

COMPUTER PROGRAMMING TIN212

Date: 23 August 2019 Time: 08.30-13.30 Place: Samhällsbyggnad

---

Course responsible:	Robin Adams, tel. 076 856 48 64 Will visit hall at 09.00 and 13.00
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 scales:	**If you have completed the Dugga:** Maxmimum total 30 points For this exam the following grades will be given: 3: 15 points, 4: 20 points, 5: 25 points **If you have not completed the Dugga:** Maximum total 60 points For this exam the following grades will be given: 3: 28 points, 4: 38 points, 5: 48 points
Exam review:	24 September 2019 14.00–16.00 EDIT 6466

- This exam is divided into two parts. Part A has four (4) questions and part B has three (3) questions, for a total of seven (7) questions.
- **If you have completed the Dugga:** You should only answer Part B of this exam.
- **If you have not completed the Dugga:** You should answer Parts A and B of this exam.
- 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 7.

Good luck!

Part A

Answer the questions in this part only if you have **not** completed the Dugga.

1. (a) Write a code fragment (kodavsnitt) that prints the sequence

$$1, 4, 9, 16, 25, \ldots, 100$$

   using a `for`-loop. (2 points)

   (b) Write a code fragment (kodavsnitt) that prints the sequence

```
a
bb
ccc
```

   etc., ending with 26 z's. (4 points)

   (6 points total)

2. (a) What is the output of the following Java program?

```
public class Exercise2a {
   public static void main(String[] args) {
      for (int i = 0; i < 3;) {
         System.out.println("Java");
      }
   }
}
```

   (3 points)

   (b) What is the output of the following Java program?

```
public class Exercise2b {
   public static void main(String[] args) {
      int i = 0, j = 9;
      do {
         i++;
         if (j-- < i++) {
            break;
         }
      } while (i < 5);
      System.out.println(i + "" + j);
   }
}
```

   (3 points)

(c) What is the output of the following Java program?

```
class First
{
   First() {
      System.out.println("a");
   }

   void method() {
      System.out.println("First method");
   }
}

class Second extends First
{
   Second() {
      System.out.println("b");
   }

   @Override
   void method() {
      super.method();
      System.out.println("Second method");
   }
}

class Third extends Second
{
    Third() {
       System.out.println("c");
    }

    @Override
    void method() {
       System.out.println("Third method");
       super.method();
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        Third c = new Third();
        c.method();
    }
}
```

(6 points)

(12 points total)

3. I have 100.000kr in the bank, and my bank account pays 10% interest (ränta) per year. I wrote this program to calculate the first year when I will have 10.000.000kr, if I do not add to the account or take any money out of the account. I think it is giving the wrong answer. How should it be modified to give the correct answer?

```
public class BecomeRich {

    public static void main(String[] args) {
        int years = 1;
        int balance = 100_000;
        while (balance < 10_000_000) {
            balance *= 1.1;
            years++;
        }
        System.out.printf("You will be rich in the year %d\n", 2019 + years);
    }

}
```

(2 points)

4. A sequence of words is said to be in *camel case* if it is written without spaces, with the first word all in lower case letters (små bokstäver), and the other words with the first letter a captial letter (stor bokstav) and the other letters all lower case.

   Write a Java program that takes a string as input from the user. If the string is a sequence of words in camel case, the program should print the number of words. If not, it should print the message "Not camel case". (For this question, a word is any sequence of letters.)

   Examples:

   - Input: 'thisSentenceIsWrittenInCamelCase'
     Output: '7 words'
   - Input: 'abcDefgHijkl'
     Output: '3 words'
   - Input: ''
     Output: '0 words'
   - Input: 'camel'
     Output: '1 word'
   - Input: 'The 27 quick Brown Foxes'
     Output: 'Not camel case'

   Hint: You may use the following methods from the standard library.

   - `boolean Character.isLetter(char c)` returns `true` if `c` is a letter, otherwise `false`;
   - `boolean Character.isUpperCase(char c)` returns `true` if `c` is a capital letter, otherwise `false`;
   - `boolean Character.isLowerCase(char c)` returns `true` if `c` is a lower case letter, otherwise `false`.
   - If `s` has type `String`, then `char s.charAt(int n)` returns the character at position `n` in the string, where the indexes range from 0 to `s.length() - 1`.
     It throws an `IndexOutOfBoundsException` if `n >= s.length()`.

   (10 points)

Part B

All students should answer the questions in this part.

5. Write a class named `Species`. An instance of `Species` will represent a population of animals which is either increasing or decreasing by a constant percentage rate every year.

   The population is always an integer. Whenever we calculate the population of the species, we round down. Thus, a species with population 100 and growth rate 2% will have population 102 next year. A species with population 50 and growth rate 3% will have population 51 next year. A species with population 1 and growth rate -10% will sadly have population 0 next year, and so will be extinct.

   The class should have the following members:

   - An instance variable `name` (type `String`) for the species's name.
   - An instance variable `population` (type `int`) for the current size of the species's population.
   - An instance variable `area` (type `double`) for the size of the area that the species covers, in km^2.
   - An instance variable `growthRate` (type `double`) for the rate at which the population increases every year (with a negative value if the population is decreasing). A rate of increase by 10% per year should be represented by a value of `0.1` in this variable; a rate of decrease by 5% per year should be represented by a value of `-0.05`.
   - A constructor
     `public Species(String name, int population, double area, double growthRate)`
     that sets the initial values of the instance variables. The constructor should throw an exception if we try to create a species with a negative population or negative area.
   - Getters for each of the instance variables.
   - A method `public String toString()` that returns the information about the species in the following format:

     ```
     Moose  Population: 2743  Growth rate: 0.02  Area: 2754.36 km2
     ```

   - A method `public double getDensity()` that returns the population density; that is, the number of animals per km^2.
   - A method `public int predictPopulation(int years)` that returns what the size of the population will be after the given number of years.
   - A method `public int predictExtinction()` that returns the number of years after which the population will be extinct, or -1 if the population will never become extinct.

   (10 points)

6. Write a class `Matrix`. An instance of this class should represent a matrix of integers. The class should have:

   - instance variables `rows` and `cols` (type `int`) that hold the number of rows and the number of columns in the matrix;
   - an instance variable `entry` of type `int[][]` that holds the entries of the matrix, with `entry[i-1][j-1]` being the (i, j)-entry;
   - a constructor `public Matrix(int rows, int cols, int[][] entry)` which sets the instance variables, or throws an `IllegalArgumentException` if `rows <= 0`, `cols <= 0` or `entry` does not have the correct size;
   - getters for `rows` and `columns`;
   - a method `public int getEntry(int i, int j)` that returns the (i, j)-entry of the matrix, or throws an `ArrayIndexOutOfBoundsException` if there is no (i, j)-entry;

- a method `public Matrix plus(Matrix b)` such that the expression `a.plus(b)` should return the sum of the two matrices `a` and `b`, or throw a `RuntimeException` if `a` and `b` cannot be added;
- a method `public Matrix times(Matrix b)` such that the expression `a.times(b)` should return the product of the two matrices `a` and `b`, or throw a `RuntimeException`

A reminder: a matrix with m rows and n columns

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

is called an $m \times n$-matrix. The value a_{ij} is called the (i, j)-entry.

Matrices A and B can be added if and only if they have the same number of rows and the same number of columns. The sum is then defined by:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{pmatrix}$$

Matrices A and B can be multiplied if and only if the number of columns in A is equal to the number of rows in B. The product is then defined by:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

where $c_{ij} = a_{i1}b_{1j} + \cdots + a_{in}b_{nj}$. (12 points)

7. Consider the sequence of positive integers formed using only the digits 4 and 7, arranged in order:

$$4, 7, 44, 47, 74, 77, 444, \ldots$$

Write a method `int foursAndSevens(int n)` which, given a positive integer `n`, returns the nth number in this sequence. (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. (a)

```
for (int i = 1; i <= 10; i++) {
   System.out.println(i * i);
}
```

(b)

```
char letter = 'a';
for (int i = 1; i <= 26; i++) {
    for (int j = 0; j < i; j++) {
        System.out.print(letter);
    }
    System.out.print("\n");
    letter++;
}
```

2 (a)

```
Java
Java
Java
... (infinitely often)
```

(b)

```
66
```

(c)

```
a
b
c
Third method
First method
Second method
```

3.

Change the line "int years = 1;" to "int years = 0;"

4.

```
public class CamelCase {
    // Returns the number of words in the given camel case sentence
    // or -1 if the string is not a sentence in camel case
    private static int camelCaseWords(String sentence) {
        int words = 1;

        if (sentence.isEmpty()) {
            return 0;
        }
        if (! Character.isLowerCase(sentence.charAt(0))) {
            return -1;
        }

        for (int i = 1; i < sentence.length(); i++) {
            if (! Character.isLetter(sentence.charAt(i))) {
                return -1;
            }
            if (Character.isUpperCase(sentence.charAt(i))) {
                words++;
            }
        }
        return words;
    }

    public static void main(String[] args) {
```

```
        Scanner in = new Scanner(System.in);

        System.out.println("Please enter a sentence in camel case.");
        String sentence = in.nextLine().trim();
        int words = camelCaseWords(sentence);
        if (words == -1) {
            System.out.println("Not camel case");
        } else if (words == 1) {
            System.out.println("1 word");
        } else {
                System.out.printf("%d words\n", words);
        }
    }
}
```

5.

```
public class Species {

    private String name;
    private int population;
    private double area;
    private double growthRate;

    public Species(String name, int population, double area, double
growthRate) {
        if (population < 0) {
            throw new IllegalArgumentException("Tried to create species
with population " + population);
        }
        if (area < 0) {
            throw new IllegalArgumentException("Tried to create species
with area " + area);
        }

        this.name = name;
        this.population = population;
        this.area = area;
        this.growthRate = growthRate;
    }

    public String getName() {
        return name;
    }

    public int getPopulation() {
        return population;
    }

    public double getArea() {
        return area;
    }

    public double getGrowthRate() {
        return growthRate;
    }

    public String toString() {
        return String.format("%s  Population: %d  Growth rate: %.2f  Area:
%.2f km2", name, population, growthRate, area);
    }
```

```
    public double getDensity() {
        return population / area;
    }

    public int predictPopulation(int years) {
        int predictedPopulation = population;
        for (int y = 1; y <= years; y++) {
            predictedPopulation *= 1 + growthRate;
        }
        return predictedPopulation;
    }

    public int predictExtinction() {
        if (growthRate <= 0) {
            return -1;
        }

        int years = 1;
        while (predictPopulation(years) > 0) {
            years++;
        }
        return years;
    }
}
```

6.

```
public class Matrix {
    private int rows;
    private int cols;
    private int[][] entry;

    Matrix(int rows, int cols, int[][] entry) {
        this.rows = rows;
        this.cols = cols;
        this.entry = entry.clone();
        // Accept: this.entry = entry;
    }

    public int getRows() {
        return rows;
    }

    public int getCols() {
        return cols;
    }

    public int getEntry(int i, int j) {
        return entry[i][j];
    }

    public Matrix plus(Matrix b) {
        if (rows != b.rows || cols != b.cols) {
            throw new RuntimeException("Attempted to add matrices of
different shapes");
        }

        int[][] newEntry = new int[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                newEntry[i][j] = entry[i][j] + b.entry[i][j];
            }
```

```
        }

        return new Matrix(rows, cols, newEntry);
    }

    public Matrix times(Matrix b) {
        if (cols != b.rows) {
            throw new RuntimeException("Attempted to multiply matrices of
incompatible shapes");
        }

        int[][] newEntry = new int[rows][b.cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < b.cols; j++) {
                int sum = 0;
                for (int k = 0; k < cols; k++) {
                    sum += entry[i][k] * b.entry[k][j];
                }
                newEntry[i][j] = sum;
            }
        }

        return new Matrix(rows, b.cols, newEntry);
    }
}
```

7.

```
int foursAndSevens(int n) {
    int[] arr = new int[n];
    arr[0] = 4;
    arr[1] = 7;

    for (int i=2; i<n; i++)
    {
        if (i%2 != 0)
            arr[i] = arr[i/2]*10 + 4;
        else
            arr[i] = arr[(i/2)-1]*10 + 7;
    }
    return arr[n-1];
}
```

Alternative solution:

```
// Returns true if n only uses the digits 4 and 7
boolean allFoursAndSevens(int n) {
    while (n > 0) {
        if (n % 10 != 4 && n % 10 != 7) {
            return false;
        }
        n = n / 10;
    }
    return true;
}

int foursAndSevens(int n) {
    int k = 3;
    while (n > 0) {
        k++;
        if (allFoursAndSevens(k)) {
            n--;
```

```
            }
        }
        return k;
    }
```