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

Göteborg 27 August 2019

COMPUTER PROGRAMMING part B TIN213

Date: 27 August 2019 Time: 08.30-11.30 Place: Samhällsbyggnad

---

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 24 September 2019 14.00–16.00 EDIT 6466

- Answer all the questions. There are three (3) 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 4.

Good luck!

1. In Sweden, a vehicle's registration shows the vehicle's *kerb weight* (tjänstevikt) and *maximum load* (maxlost). Its *total weight* (totalvikt) is defined to be the sum of its kerb weight and maximum load.

   The following categories of driving licence exist:

   - A category A licence allows a person to drive a tractor with maximum speed at most 40kph.
   - A category B licence allows a person to drive a car or lorry with a total weight of at most 3500kg, or any tractor.
   - A category C1 licence allows a person to drive a lorry with a total weight of at most 7500kg, or any car.

   Write an abstract class `Vehicle` and non-abstract subclasses `Tractor`, `Car`, `Lorry` according to the following specification.

   An instance of `Vehicle` represents a vehicle. The abstract class `Vehicle` should have the following methods. (You may decide which methods should be abstract and which should not.)

   - a method `public int getKerbWeight()` that returns the vehicle's kerb weight in kg;
   - a method `public int getMaxLoad()` that returns the vehicle's maximum load in kg;
   - a method `public int getTotalWeight()` that returns the vehicle's total weight in kg;
   - a method `public boolean canDrive(String licence)` that takes a string "A", "B" or "C1", and returns `true` if the vehicle can be driven by somebody holding that class of licence, and `false` if not. It should throw an `IllegalArgumentException` if `licence` is not any of these three strings.

   The subclasses should have constructors:

   - `public Tractor(int kerbWeight, int maxLoad, int maxSpeed)` with parameters giving the tractor's kerb weight in kg, maximum load in kg and maximum speed in kph;
   - `public Car(int kerbWeight, int maxLoad)` with parameters giving the car's kerb weight in kg and maximum load in kg;
   - `public Lorry(int kerbWeight, int maxLoad)` with parameters giving the lorry's kerb weight in kg and maximum load in kg;

   (You may decide which instance variables each class should have, and whether `Vehicle` should have one or more constructors.) (7 points)

2. A *polynomial* is a function of the following form

   $$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

   for same constants a_0, a_1, …, a_n. The integer n is called the *degree* of the polynomial, and each a_i is called a *coefficient*.

   Write a class `Polynomial`. An instance of this class should represent a polynomial function. The class should have the following members:

   - an instance variable `degree` (type `int`) that holds the degree;
   - an instance variable `coefficients` (type `double[]`) that holds the coefficients in the order $\{a_0, a_1, \ldots, a_n\}$;
   - a constructor `Polynomial(double[] coefficients)` that creates a polynomial with the given coefficients;
   - a constructor `Polynomial(int degree)` that creates a polynomial of the given degree with coefficients all equal to 1, or throws an exception if `degree < 0`;
   - a method `Polynomial plus(Polynomial b)` such that `a.plus(b)` returns the sum of the polynomials `a` and `b`.

- a method `Polynomial times(Polynomial b)` such that `a.times(b)` returns the product of the polynomials `a` and `b`.
- a method `double evaluate(double x)` that finds the value of the polynomial at the given argument x

Recall that the sum of two polynomials of the same length is given by

$$(a_0 + a_1x + \cdots + a_nx^n) + (b_0 + b_1x + \cdots + b_nx^n) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_n + b_n)x^n$$

If we are adding two polynomials of different lengths, we assume the 'missing' coefficients are zero:

$(\mathrm{a}_0 + a_1x + \cdots + a_mx^m) + (b_0 + b_1x + \cdots + b_nx^n)$
$= (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_m + b_m)x^m + b_{m+1}x^{m+1} + \cdots + b_nx^n$
if $m < n$.

The product of two polynomials is given by

$$(a_0 + a_1x + \cdots + a_mx^m)(b_0 + b_1x + \cdots + b_nx^n) = c_0 + c_1x + \cdots + c_{m+n}x^{m+n}$$

where

$$c_i = a_0b_i + a_1b_{i-1} + \cdots + a_ib_0$$

Hint: You may find it useful to use the method `Arrays.fill(double[] a, double[] val)`, which assigns the value `val` to each element of `a`.

(13 points)

3. A *permutation* of a string is a string formed by rearranging its characters.

Write a method `void printPermutations(String word)` that prints out all the permutations of `word`. For example, given the string *abc*, it should output

```
abc
acb
bac
bca
cab
cba
```

Your method may print the permutations in any order, and it may print a permutation more than once.

Hint: You may find it useful to write a recursive method
`void printPermutationsWithPrefix(String prefix, String rest)`
that prints all strings consisting of `prefix` followed by a permutation of `rest`. (Other solutions are possible.)

Hint: You may find it useful to use these methods.

- If `s` has type `String`, then `String s.substring(i)` returns the substring of `s` that starts with the character at index `i` and ends at the end of `s`. It throws an `IndexOutOfBoundsException` if `i < 0` or `i > s.length()`.
- If `s` has type `String`, then `String s.substring(i, j)` returns the substring of `s` that starts with the character at index `i` and ends with the character at index `j-1`. It throws an `IndexOutOfBoundsException` if `i < 0` or `j > s.length()` or `i > j`.
- 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)

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.

```
abstract class Vehicle {
    private int kerbWeight;
    private int maxLoad;

    Vehicle(int kerbWeight, int maxLoad) {
        this.kerbWeight = kerbWeight;
        this.maxLoad = maxLoad;
    }

    public int getKerbWeight() {
        return kerbWeight;
    }

    public int getMaxLoad() {
        return maxLoad;
    }

    public int getTotalWeight() {
        return kerbWeight + maxLoad;
    }

    public abstract boolean canDrive(String licence) throws
IllegalArgumentException;
}

class Tractor extends Vehicle {
    private int maxSpeed;

    public Tractor(int kerbWeight, int maxLoad, int maxSpeed) {
        super(kerbWeight, maxLoad);
        this.maxSpeed = maxSpeed;
    }

    @Override
    public boolean canDrive(String licence) throws IllegalArgumentException
{
        if (licence.equals("A")) {
            return (maxSpeed <= 40);
        }
        if (licence.equals("B")) {
            return true;
        }
        if (licence.equals("C")) {
            return false;
        }
        throw new IllegalArgumentException("Unrecognised licence type: " +
licence);
    }
}

class Car extends Vehicle {
    public Car(int kerbWeight, int maxLoad) {
        super(kerbWeight, maxLoad);
    }

    @Override
    public boolean canDrive(String licence) throws IllegalArgumentException
{
        if (licence.equals("C")) {
            return (getTotalWeight() <= 7500);
        }
        if (licence.equals("B")) {
            return (getTotalWeight() <= 3500);
        }
        if (licence.equals("A")) {
            return false;
        }
        throw new IllegalArgumentException("Unrecognised licence type: " +
licence);
    }
}

class Lorry extends Vehicle {
    public Lorry(int kerbWeight, int maxLoad) {
        super(kerbWeight, maxLoad);
    }

    @Override
    public boolean canDrive(String licence) throws IllegalArgumentException
{
        if (licence.equals("C")) {
            return (getTotalWeight() <= 7500);
        }
        if (licence.equals("A") || licence.equals("B")) {
            return false;
        }
        throw new IllegalArgumentException("Unrecognised licence type: " +
licence);
    }
}
```

2.

```
public class Polynomial {
    private int degree;
    private double[] coefficients;

    public Polynomial(double[] coefficients) {
        this.degree = coefficients.length - 1;
        this.coefficients = coefficients.clone();
    }

    public Polynomial(int degree) {
        this.degree = degree;
        this.coefficients = new double[degree + 1];
        for (int i = 0; i <= degree; i++) {
            coefficients[i] = 1;
        }
// Alternatively: Arrays.fill(coefficients, 1);
    }

    double getCoefficient(int i) {
        return (i <= degree ? coefficients[i] : 0);
    }

    public Polynomial plus(Polynomial b) {
        int newDegree = Math.max(degree, b.degree);
        double[] newCoefficients = new double[newDegree + 1];
        for (int i = 0; i <= newDegree; i++) {
            newCoefficients[i] = getCoefficient(i) + b.getCoefficient(i);
        }
        return new Polynomial(newCoefficients);
    }

    public Polynomial times(Polynomial b) {
        int newDegree = degree + b.degree;
        double[] newCoefficients = new double[newDegree + 1];
        for (int i = 0; i <= newDegree; i++) {
            double sum = 0;
            for (int k = 0; k <= i; k++) {
                sum += getCoefficient(k) * b.getCoefficient(i-k);
            }
            newCoefficients[i] = sum;
        }
        return new Polynomial(newCoefficients);
    }

    public double evaluate(double x) {
        double value = 0;
        for (int i = 0; i <= degree; i++) {
            value += coefficients[i] * Math.pow(x, i);
        }
        return value;
    }
}
```

3.

```
static void printPermutationsWithPrefix(String prefix, String rest) {
    if (rest.isEmpty()) {
        System.out.println(prefix);
    }
    for (int i = 0; i < rest.length(); i++) {
        printPermutationsWithPrefix(prefix + rest.charAt(i),
rest.substring(0, i) + rest.substring(i + 1));
    }
}

static void printPermutations(String word) {
    printPermutationsWithPrefix("", word);
}
```