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

Göteborg 12 June 2019

COMPUTER PROGRAMMING TIN212

Date: 12 June 2019 Time: 08.30-13.30 Place: "Maskin"-salar

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:	Monday 19 August 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 6.

Good luck!

Part A

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

1. Are the following statements true or false? (You do not need to justify your answers.) (5 points)
   (a) A `do-while` loop is always executed at least once.
   (b) If there are two nested loops in a method, then a `break` statement in the inner loop will terminate both loops.
   (c) A static variable can be accessed within an instance method of the same class.
   (d) An instance variable can be accessed within a static method of the same class.
   (e) If a class is to be extended, then it must have a no-argument constructor.

2. What will be the output from the following program `Main.java` when it is compiled and run? (5 points)

```
class C {
    private static int a;
    private int b;

    public C() {
        a = 0;
        b = 0;
    }

    public void add(int x) {
        a += x;
        b += x;
    }

    public void printOut() {
        System.out.println("a is " + a);
        System.out.println("b is " + b);
    }
}

public class Main {
    public static void main(String[] args) {
        C c1 = new C();
        C c2 = new C();

        c1.add(1);
        c2.add(2);

        c1.printOut();
        c2.printOut();

        C c3 = new C();
        c1.add(3);
        c2.add(3);
        c3.add(3);

        c3.printOut();
    }
}
```

3. Given two positive integers k and n, we say k is a *proper factor* of n if k divides n and $k \neq n$.

   A positive integer n is called *perfect* if the sum of its proper factors is n itself. We say it is *deficient* if the sum of its proper factors is $< n$, and *abundant* if the sum of its proper factors is $> n$.

   For example, the number 6 is perfect because its proper factors are 1, 2 and 3, and $1+2+3 = 6$. The number 18 is abundant because its proper factors are 1, 2, 3, 6 and 9, and $1+2+3+6+9 = 21 > 18$. The number 19 is deficient because the only proper factor is 1, and $1 < 19$.

   Write a program which asks the user to enter a positive integer, then outputs "perfect", "deficient", "abundant" or "error" depending on whether the user entered a perfect number, a deficient number, an abundant number, or a string that cannot be parsed as a positive integer. (You may either use the console for input and output, or use dialogue boxes.) (10 points)

4. Write a method `void printPermutations(String str)` that takes a string `str` and prints to the console all the *permutations* of `str`. A *permutation* of `str` is a string formed by rearranging the characters in `str`.

   For example, if the method is called with the string *abc*, it should print:

   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 make the method recursive.)

   (10 points)

Part B

All students should answer the questions in this part.

5. What will be the output of the following program `Main.java` when run? (7 points)

```
abstract class A {
    int value;

    A() {
        value = 1000;
    }

    A(int initial) {
        value = initial;
    }

    abstract void increase(int i);

    final void printOut() {
        System.out.println("My value is: " + value);
    }
}

class B extends A {
    B(int i) {
        super(3 * i);
    }

    @Override
    void increase(int i) {
        value += 2 * i;
    }
}

class C extends A {
    @Override
    void increase(int i) {
        value += 3 * i;
    }
}

public class Main {
    public static void main(String[] args) {
        A[] array = new A[10];

        for (int i=0; i<10; i+=2) {
            array[i] = new B(i);
            array[i+1] = new C();
        }

        for (int i=0; i<10; i++) {
            array[i].increase(i);
            array[i].printOut();
        }
    }
}
```

6. A library uses a program to keep track of which books have been loaned. When a book is borrowed, it must be returned after 28 days. The fee for a book being returned late is 3kr per day. A borrower may renew the loan up to three times. The book is then due back 28 days after the date on which it was renewed. A borrower cannot renew a book that is overdue.

   The program includes the interface to represent borrowers:

```
interface Borrower {
  private String firstName;
  private String surname;
  private long personnummer;
}
```

   (a) Write a class `Book` that has the following methods:
   - a constructor which takes two parameters for the title and author's name;
   - a method `public String getTitle` that returns the book's title;
   - a method `public String getAuthor` that returns the author's name;
   - a method `public void borrow(Borrower borrower)` that is called when the book is borrowed;
   - a method `public int returnBook()` that is called when the book is returned, and returns the fee that the borrower has to pay. It returns 0 if the book was returned on time.
   - a method `public boolean renew()` that is called when the borrower wants to renew the book. It returns `true` if the book has been renewed, or `false` if the book could not be renewed.
   - a method `public boolean isOverdue` that returns `true` if the book is overdue, and `false` otherwise;
   - a method `public int getFee` that returns the value of the fee that would have to be paid if the book was returned today. It returns 0 if the book is not overdue.

   (8 points)

   (b) Some books in the library are for short-term loan only, meaning they are borrowed for one week instead of four. The fee for a book of this type being overdue is 10kr per day. A short-term loan cannot be renewed.

   Write a subclass `ShortTermBook` of `Book` whose objects represent books available for short-term loan. (5 points)

   Hint: You can use these two classes from the standard library: `java.time.LocalDate` and `java.time.temporal.ChronoUnit`. An object of type `LocalDate` represents a calendar date. These classes have the following methods:

   - `LocalDate date = LocalDate.now();` sets the value of `date` to be equal to the current date.
   - `LocalDate date2 = date.plusDays(3);` sets the value of `date2` to be 3 days after the value of `date`.
   - `LocalDate date2 = date.minusDays(3);` sets the value of `date2` to be 3 days before the value of `date`.
   - If `date1` and `date2` both have type `LocalDate`, then
     `long n = ChronoUnit.DAYS.between(date, date2);`
     sets `n` equal to the number of days between `date` and `date2`. This will be positive if `date2` is after `date`, negative if `date2` is before `date`, or 0 if `date` and `date2` represent the same date.

   (13 points total)

7. Write a method `static int minSum(int[] digits)` which takes an array of integers between 0 and 9, and returns the smallest integer that can be written as the sum of two numbers that can be formed using all the digits in the array.

   For example, given the array $\{6, 8, 4, 5, 2, 3\}$, the method should return 604, because the smallest number that can be formed as a sum of two numbers using all these digits is $246 + 358 = 604$.

   You may assume that `digits` has size ≥ 2, and that not all its elements are 0.

   (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/`.)

Part A

1.
a. True
b. False
c. True
d. False
e. False

2.
```
a is 3
b is 1
a is 3
b is 2
a is 9
b is 3
```

3.
```
public class Perfect {
    private static int sumOfFactors(int n) {
        int sum = 0;
        for (int i = 1; i < n; i++) {
            if (n % i == 0) {
                sum += i;
            }
        }
        return sum;
    }

    public static void main(String[] args) {
        int n = 0, sumOfFactors;

        Scanner scanner = new Scanner(System.in);
        System.out.println("Please enter a positive integer");
        try {
            n = scanner.nextInt();
        } catch (InputMismatchException e) {
            System.out.println("error");
            System.exit(1);
        }

        sumOfFactors = sumOfFactors(n);
        if (n < 0) {
            System.out.println("error");
        } else if (sumOfFactors < n) {
            System.out.println("deficient");
        } else if (sumOfFactors == n) {
            System.out.println("perfect");
        } else {
            System.out.println("abundant");
        }
    }
}
```

4.
```
public static void printPermutations(String str) {
    printPermutationsHelper(str, "");
}

/** Print all the permutations of str, prepending prefix to each of them */
private static void printPermutationsHelper(String str, String prefix) {
    if (str.length() == 0) {
        System.out.println(prefix);
    }

    for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);
        String rest = str.substring(0, i) + str.substring(i + 1);
        printPermutationsHelper(rest, prefix + c);
    }
}
```

5.

```
My value is: 0
My value is: 1003
My value is: 10
My value is: 1009
My value is: 20
My value is: 1015
My value is: 30
My value is: 1021
My value is: 40
My value is: 1027
```

6. (a)

```
public class Book {
    private String title;
    private String author;
    private Borrower borrower;
    private LocalDate borrowed;
    private int renewals;

    static int renewalPeriod = 28;
    static int feePerDay = 3;
    static int maxRenewals = 3;

    Book(String title, String author) {
        this.title = title;
        this.author = author;
        borrower = null;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public void borrow(Borrower borrower) {
        this.borrower = borrower;
        borrowed = LocalDate.now();
        renewals = 0
    }

    public int returnBook() {
        int fee = getFee();

        borrower = null;
        borrowed = null;
        return fee;
    }

    public boolean renew() {
        if (borrowed == null || renewals == maxRenewals) {
            return false;
        }

        renewals++;
        borrowed = LocalDate.now();
        return true;
    }

    public boolean isOverdue() {
        if (borrowed == null) return false;
        return (ChronoUnit.DAYS.between(borrowed.plusDays(renewalPeriod),
LocalDate.now()) > 0);
    }

    public int getFee() {
        if (borrowed == null) return 0;
        long overdue =
ChronoUnit.DAYS.between(borrowed.plusDays(renewalPeriod), LocalDate.now());
        if (overdue > 0) {
            return (int) (overdue * feePerDay);
        } else {
            return 0;
        }
    }
}

class ShortTermBook extends Book {
    ShortTermBook(String title, String author) {
        super(title, author);
    }

    static int renewalPeriod = 7;
    static int feePerDay = 10;
    static int maxRenewals = 0;
}
```

7.
Here is one solution. There are other possible methods.

```
static int minSum(int digits[])
{
    Arrays.sort(digits);

    int num1 = 0;
    int num2 = 0;

    // Generating numbers alternatively
    for (int i = 0; i < digits.length; i++) {

        if (i % 2 == 0)
            num1 = num1 * 10 + digits[i];
        else
            num2 = num2 * 10 + digits[i];
    }

    // Return the minimum possible sum
    return num1 + num2;
}
```