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

Göteborg 9 March 2019

COMPUTER PROGRAMMING part B TIN213

Date: 9 March 2019 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:	Monday 15 April 2019 14.00–16.00 EDIT 6466

- Answer all the questions. There are five (5) 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 6.

Good luck!

1. Are the following statements true or false? You do not need to justify your answers.

   (a) Two reference variables can refer to the same object. (1 point)

   (b) A `finally` block is executed before the `catch` block but after the `try` block. (1 point)

   (c) An abstract class cannot have non-abstract methods. (1 point)

   (d) A `protected` instance variable can be accessed by a subclass in another package. (1 point)

   (4 points total)

2. What will be the output of the following program when run?

```
public class Exercise2 {
  private void f(int i) {
    i += 2;
  }

  public static void main(String args[]) {
    int i = 1;
    System.out.print(i+" , ");
    f(i);
    System.out.println(i);
  }
}
```

   (2 points)

3. Consider the following abstract class for accounts in a bank.

```
public abstract class Account {
 private int accountNo;

 public Account(int accountNo) {
     this.accountNo = accountNo;
 }

 public int getAccountNo() {
     return accountNo;
 }

 public abstract void deposit(double amount);

 public abstract boolean withdraw(double amount);

 public abstract double getBalance();

 public abstract void endOfMonth();
}
```

Every subclass of `Account` should be written to satisfy the following specification:

- `getAccountNo` should return the account number
- `deposit` is called when the customer makes a deposit (*insättning*) into the bank account. It should update the account balance appropriately.
- `withdraw` is called when the customer tries to make a withdrawal (*uttag*) from the bank account. If the customer is allowed to make the withdrawal, then the account balance should be updated appropriately and the method should return `true`. Otherwise, the balance should be unchanged and the method should return `false`.
- `getBalance` should return the current balance (*saldo*) of the account.
- `endOfMonth` is called at the end of every month. It should make any changes that must be made once per month.

(a) Write the code that should replace the four 'TODO' comments in the following code to complete the subclass `SimpleAccount`. With an account of this type, the customer is allowed to make a withdrawal if this would not make the balance negative. If the customer tries to withdraw more money than is in the account, the withdrawal should fail. (3 points)

```
public class SimpleAccount extends Account {
  private double balance;

  public SimpleAccount(int accountNo, double balance) {
      // TODO
  }

  @Override
  public void deposit(double amount) {
      // TODO
  }

  @Override
  public boolean withdraw(double amount) {
      // TODO
```

```
    }

    @Override
    public double getBalance() {
        // TODO
    }

    @Override
    public void endOfMonth() {}
}
```

(b) Write a subclass `CheckingAccount`. The constructor should have three parameters: the account number, starting balance, and overdraft limit. The balance is not allowed to go below the overdraft limit. There is a monthly fee of 100kr for having an account of this type. (4 points)

(c) Write a subclass `TransactionAccount`. The constructor should have two parameters: the account number and starting balance. With an account of this type, the customer is allowed four free transactions every month. For every transaction after the fourth, there is a 50kr fee. The customer receives 0,3% interest on the balance of the account every month. The balance is not allowed to become negative. (5 points)

(12 points total)

4. Consider the following methods.

```
int f(int[] a, int n) {
  if (n == 0) {
    return a[0];
  }
  int x = f(a, n-1);
  if (x > a[n]) {
    return x;
  } else {
    return a[n];
  }
}

int g(int[] a) {
  return f(a, a.length - 1);
}
```

   (a) What does the following code output?

   ```
   int[] a = {2, 9, -5, 11, 0};
   System.out.println(f(a, 4));
   ```

   (**Hint**: begin by calculating `f(a, 0)`, `f(a, 1)`, etc.) (2 points)

   (b) Describe briefly what the function `g` calculates when passed a non-empty array. You do not have to explain how the function works; just say what value it returns. (3 points)

   (5 points total)

5. The *lucky numbers* are a subset of the positive integers defined as follows. We begin with the complete sequence of positive integers:

$$1, 2, 3, 4, 5, \ldots$$

   We delete every second number, leaving

$$1, 3, 5, 7, 9, 11, 13, 15, \ldots$$

   In this sequence, we delete every third number, leaving

$$1, 3, 7, 9, 13, 15, \ldots$$

   Now we delete every fourth number, and so on indefinitely.

   A positive integer is *lucky* if it is never deleted in the above process. The first four lucky numbers are 1, 3, 7 and 13.

   Write a method `boolean isLucky(int n)` which, given a positive integer `n`, returns `true` if `n` is lucky and `false` if not.

   (**Hint**: You can solve this problem by first writing a recursive method `boolean isLuckyHelper(int p, int k)` which returns `true` if the number in position p after k steps is lucky.)

   (7 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/`.)