Birkbeck

(University of London)

BSc EXAMINATION

Department of Computer Science and Information Systems

Software and Programming II (COIY026H6)

Credit value: 15 credits

Date of examination: Thursday, 2nd June 2016
Duration of paper: 10:00 – 13:00 (3 HOURS)

1. Candidates should attempt ALL questions in the paper.
2. There are **9** questions on this paper.
3. The number of marks varies from question to question.

Question:	1	2	3	4	5	6	7	8	9	Total
Marks:	9	6	16	3	8	26	8	16	8	100

4. You are advised to look through the entire examination paper in order to plan your strategy before getting started.
5. Simplicity and clarity of expression in your answers are important.
6. Electronic calculators and supplementary material like notes and text-books are NOT permitted.
7. Answer questions using the Java programming language unless stated otherwise. You may add auxiliary methods unless stated otherwise.
8. Start each question on a new page.
9. An extract of the Java API is given at the end of this exam paper.

1. Consider the following Java classes. (**9** marks)

```
public class Alpha {
    protected int x;

    public Alpha() {  this(10);  }
    public Alpha(int x) {  this.x = x;  }

    public void f() {  x = 20;  }
    public void f(int x) {  x = 25;  }
    public void g(Object a) {  x = 30;  }
    public void h(Object a) {  x = 50;  }
    public void h(Integer a) {  x = 55;  }
}

public class Beta extends Alpha {

    public Beta() {  super(15);  }

    public void g(Object a) {  x = 35;  }
    public void g(Integer a) {  x = 40;  }
    public void h(Object b) {  x = 70;  }
    public void h(Integer b) {  x = 75;  }

    public static void main(String[] args) {
        Alpha a = new Alpha();
        System.out.println("1 : " + a.x);
        a.f();
        System.out.println("2 : " + a.x);
        a.f(100);
        System.out.println("3 : " + a.x);

        Beta b = new Beta();
        System.out.println("4 : " + b.x);
        b.g(200);
        System.out.println("5 : " + b.x);
        b.h(300);
        System.out.println("6 : " + b.x);

        Alpha c = b;
        c.g(400);
        System.out.println("7 : " + c.x);
```



```
        c.h(500);
        System.out.println("8 : " + c.x);
        c.f();
        System.out.println("9 : " + b.x);
    }
}
```

Please write down the output that is produced when `java Beta` is invoked on the command line after the classes have been compiled.

2. (**Total: 3 + 1 + 2 = 6** marks)

Using suitable examples, explain in parts (a) and (b) the prerequisites and consequences of the following modifiers in a method declaration in a Java class, and explain in part (c) why their combination is not allowed.

(a) `abstract` — 3 marks

(b) `final` — 1 mark

(c) `abstract final` — 2 marks

3. Consider the following Java interface. (**Total: 8 + 8 = 16** marks)

```
public interface Transformer<A, B> {
    B transform(A item);
}
```

(a) Write a class `StringDoubler` that implements the above interface. In particular, the class is supposed to have a method — 8 marks

```
public String transform(String item)
```

that behaves as follows: If `item` is the `null` reference, `null` is returned. Otherwise a "doubled version" of `item` is returned. For example, after we execute the line

```
String s = new StringDoubler().transform("abc");
```

the variable `s` will reference `"abcabc"`.

(b) Write a `static` method `transformList` in the interface `Transformer` that takes `Transformer<A,B> trafo` and `List<_> items` as inputs and returns an `ArrayList<B>`. — 8 marks

Instead of `List<_>`, use the most general generic type that is possible here (i.e., replace the "`_`" between the "`< >`" sensibly).

If the input list `items` is [a_1,...,a_n], your method is supposed to return a new `ArrayList` with the following contents:

[trafo.transform(a_1),...,trafo.transform(a_n)]

In your solution you may assume that the parameters of this method are never `null`. Your method is not supposed to modify the structure of the list `items`.

4. Consider the following class. (**3** marks)

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }
}
```

Now consider the following class.

```
public class Employee extends Person {
    private int salary;
}
```

Write a constructor for the class `Employee` with the following signature:

```
public Employee(String name, int salary)
```

Your constructor is supposed to set the attributes of the classes `Person` and `Employee` that correspond to the parameters of the constructor. In this question, you may assume that the constructor will only be called with sensible parameter values. **Do not modify the class `Person`.**

5. Write a `static` Java method that does the following: (**8** marks)

- It first asks the user on the command line to enter a `double` value.
- Then it reads the input from the command line.
- If this input can be converted to a `double` value, the `double` value is returned.
- Otherwise the method throws an `IOException` (from the package `java.io`).

In your solution you may assume that `java.io.IOException` has already been imported.

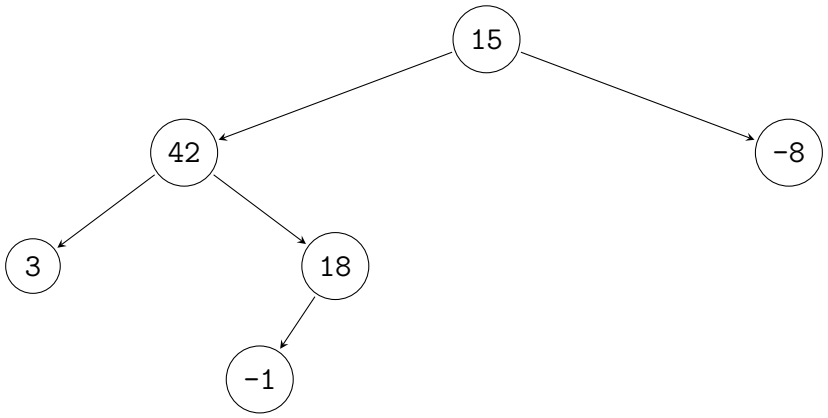
6. (**Total: 7 + 7 + 12 = 26** marks)

Consider the following Java class `Tree`, which is used to implement a data structure for binary trees. Here every node has a data value of type `int` and two references to its children (a reference to `null` indicates that there is no child in the corresponding direction).

```
public class Tree {
    private Tree left;
    private Tree right;
    private int value;

    public Tree(Tree left, Tree right, int value) {
        this.left = left;
        this.right = right;
        this.value = value;
    }
}
```

For example, such a binary tree could look as follows:



(a) In the class `Tree`, implement a method 7 marks

```
public ArrayList<Integer> asList()
```

that returns a list of the values stored in this `Tree` in *pre-order* (i.e., in the list, the values taken first from the left subtree and then from the right subtree come before the value at the current node).

For example, if `mytree` is the `Tree` depicted above, the code snippet

```
System.out.println(mytree.asList());
```

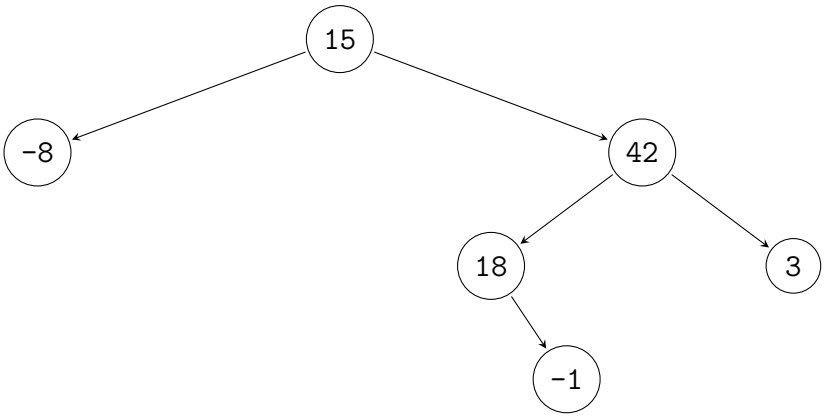
should lead to the following output:

```
[3, -1, 18, 42, -8, 15]
```

The method must not modify the original tree.

(b) In the class `Tree`, implement a method `public Tree getMirrored()` that returns a new `Tree` that is a *mirrored version* of the original tree. This means that for a node in the new tree, the nodes corresponding to the left child and to the right child of the corresponding node in the original tree are now swapped. 7 marks

For example, the mirrored version of the tree depicted above looks like this:



The method must not modify the original tree.

(c) In the class `Tree`, implement a method: 12 marks

```
public int deleteLeavesWith(int val)
```

This method deletes all the leaves of `this` tree that store the value `val` and that are not at the root of `this` tree. Recall that a *leaf* is a node in the tree that does not have any children. The method returns how many leaves it has deleted. Thus, this method may modify the original tree.

7. Consider the following Java method. (**8** marks)

```
/**
 * Computes the signum function.
 *
 * @param x an int value whose signum we want to compute
 * @return the signum function of x (1 if x is positive,
 *  0 if x is 0, -1 if x is negative)
 */
public static int signum(int x) {
    if (x > 0) {
        return 1;
    }
    if (x < 0) {
        return -1;
    }
    return 0;
}
```

Write JUnit test cases that together achieve *test coverage* for this method. In your solution you may assume that all necessary imports from `org.junit....` have already been performed.

Hint: Remember to use suitable Java annotations to label methods as JUnit test cases.

8. (**Total: 8 + 8 = 16** marks)

In this question we want to design a data structure for managing heat sources. An initial analysis has revealed the following properties of the different kinds of heat sources.

- Every heat source has its thermal power as an attribute.
- A candle is a heat source for which the colour is an important property.
- A star is a heat source whose distance in light-years from the Earth is relevant.
- An electric heat source is a heat source that generates heat via the conversion of electric energy to thermal energy. For electric heat sources, the voltage is important.
- A radiator is an electric heat source with a certain number of fins.
- A fan heater is an electric heat source with a fan that has a certain number of revolutions per minute.
- An electric stove is an electric heat source for which the diameter of the heating elements is relevant.
- Both candles and electric heat sources can be activated by humans. Therefore, they provide a method to switch them on.

(a) Devise a suitable inheritance hierarchy for the heat sources listed above. Common features should be represented in (possibly abstract) superclasses. Introduce interfaces where appropriate. Use the UML class diagram notation from the lecture to draw your design. 8 marks

In particular, mark interfaces and abstract classes as such. For each *class*, you only need to mention the data type and the name of the attributes. You do not need to mention any methods in your classes. For each *interface*, only mention its name and for each of its methods the name, input types, and output type.

(b) Implement a Java method `heatUp`. The method takes an array of heat sources as parameters. The method activates all heat sources in the array for which this is possible. It returns the number of heat sources that have been activated. 8 marks

If the `null` array reference is passed to the method, your implementation is supposed to throw an `IllegalArgumentException` to inform the user of your method that they have passed an illegal parameter to your method. Your method should be able to handle the case that some of the entries of the array may be the `null` reference.

Use the keyword `static` for the method if and only if this is appropriate.

9. (**Total: 4 + 4 = 8** marks)

(a) Consider the following Java class. 4 marks

```
import java.util.HashSet;

public class Point {

    private int x;
    private int y;

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

    @Override
    public boolean equals(Object other) {
        if (! (other instanceof Point)) {
            return false;
        }
        Point p = (Point) other;
        return this.x == p.x && this.y == p.y;
    }

    public static void main(String[] args) {
        Point p = new Point(3, 4);
        Point q = new Point(3, 4);
        HashSet<Point> set = new HashSet<Point>();
        set.add(p);
        System.out.println( p.equals(q) );
        System.out.println( set.contains(p) );
        System.out.println( set.contains(q) );
    }
}
```

Compiling and running this Java program leads to the following output:

```
true
true
false
```

Provide a short explanation for each of the lines of this output.

(b) Consider the following fragment of a Java class. 4 marks

```
import java.util.ArrayList;

[...]

    public static void printFor(ArrayList<?> list) {
        if (list != null) {
            for (Object obj : list) {
                System.out.println( obj );
            }
        }
    }

[...]
```

Write a method

```
public static void printWhile(ArrayList<?> list)
```

that has the same behaviour as the above method `printFor`, but **does not use any `for`-loops**. However, `while`-loops or recursion are explicitly allowed.

Extracts from the Java API (incomplete)

All the following constructors and methods are `public`.

`java.util.List<E>`

`boolean add(E e)` Appends the specified element to the end of this list.

`E get(int index)` Returns the element at the specified position in this list.

`int size()` Returns the number of elements in this list.

`java.util.Scanner`

`Scanner(InputStream source)` Constructs a new `Scanner` that produces values scanned from the specified input stream.

`boolean hasNextDouble()` Returns true if the next token in this scanner's input can be interpreted as a double value using the `nextDouble()` method.

`double nextDouble()` Scans the next token of the input as a `double`. This method will throw `InputMismatchException` if the next token cannot be translated into a valid `double` value. If the translation is successful, the scanner advances past the input that matched.

