A08013 *No Calculator permitted in this examination*

UNIVERSITY OF BIRMINGHAM

School of Computer Science

First Year – MSc Computer Science
First Year – MSc Financial Engineering

06 06994

Software Workshop

Summer Examinations 2013

Time Allowed: 3:00 hours

[Answer ALL Questions]

[Use a Separate Answer Book for Each Part]

Turn Over

No calculator

[PART A. USE A SEPARATE ANSWER BOOK.]

Question 1 [Conditional, Loop, Array] Total: **16%**

(a) What does the following program fragment print exactly for the three different values, `x = 1`, `x = -1`, and `x = 0`?

```
if (x > 0) {
    System.out.println(">");
}
if (x <= 0) {
    System.out.println("<=");
}
if (x == 0) {
    System.out.println("=");
}
```

[4%]

(b) Write a program fragment that prints all numbers which are divisible by 2, 3, or 5 starting with 0 up to (inclusively) a positive maximal number m of type `int` (each number in a new line). How can you test the correctness of your program? [6%]

(c) A lottery company needs a method which randomly draws 6 numbers out of the range 1 through 49 (no duplicates, order does not matter).

Implement a method, which first populates an array with the numbers 1 through `n` (`n` being 49 in the example), then uses `Math.random()` to choose one of them. The numbers drawn are to be stored in a second array `result` of length `k` ($k < n$, `k` being 6 in the example). Make sure that your method draws only from the numbers not drawn so far and draws exactly `k` times. [6%]

Question 2 [Classes, Inheritance] Total: **17%**

(a) Declare a class `CreditCard` with fields `name`, `accountNumber`, and `amount` of appropriate types together with a constructor, and a setter and a getter for `amount`. (The full class would contain other methods, but these are not of concern here.) [6%]

(b) A bank wants to offer different types of credit cards, also a `GoldCard` which has the additional field `fee`. Write a class `GoldCard` with a constructor, in which the `fee` is subtracted from the `amount` on the creation of objects. Make use of inheritance to reuse the code from part (a). (The full class would contain methods, which are not of concern here.) [6%]

(c) Suppose the program shown on the next page is in a file `Mystery.java`. Write down the sequence of source lines that get executed when running this program, along with a short justification for each line why it is executed at some particular point, what output is produced (if so) and why. Consider lines inside constructors or methods only. Your answer should start with:

- 37: first line in `main` method
- 29: first line in `B(String, String)` constructor

```
 1 class A {
 2     public  String day;
 3     private String month;
 4     public  String year() {
 5         return "2013";
 6     }
 7     public void month() {
 8         System.out.println(month + " " + year());
 9     }
10     public void day() {
11         System.out.println(day);
12     }
13     public A(String diaryDay, String diaryMonth) {
14         day = "A: " + diaryDay;
15         month = "A: " + diaryMonth;
16     }
17     public A() {
18         day = "25";
19         month = "May";
20     }
21 }
22
23 class B extends A {
24   private String month;
25   public String year() {
26     return "2014";
27   }
28   public B(String diaryDay, String diaryMonth) {
29     super();
30     day = "B: " + diaryDay;
31     month = "B: " + diaryMonth;
32   }
33 }
34
35 public class Mystery {
36   public static void main(String[] args) {
37     B b = new B("20", "June");
38     b.day();
39     b.month();
40   }
41 }
```

[5%]

Question 3 [Exceptions, Patterns, ArrayList] Total: **17%**

(a) Consider the following program.

```
public class DivisionExample {
    public static void main(String[] args) {
        int n =  Integer.parseInt(args[0]);
        int m =  Integer.parseInt(args[1]);

        try {
            System.out.println("n/m: " + (n/m));
        }
        catch (ArithmeticException e) {
            System.out.println("Oops.");
        }
    }
}
```

What output will this program print when invoked with the following commands:

(i) `java DivisionExample 1 2`

(ii) `java DivisionExample 1 0`

(iii) `java DivisionExample 1`

[6%]

(b) Currently issued British number plates consist of two uppercase letters (`A` through `Z`) followed by two digits followed by three uppercase letters, which are neither I, Q, or Z. (There are further restrictions, which we neglect here). Write a concise pattern to exactly match these number plates. [5%]

(c) For a particular day, the actual arrival times of trains are monitored against their scheduled arrival times. A train is considered late when it arrives more than five minutes after the scheduled arrival time. In this case, the number of minutes (rounded to the next integer) it is late is recorded. If trains arrive early, exactly on time or up to five minutes late, they are considered on time, and are recorded as 0 minutes late. Trains that are cancelled are recorded as 120 minutes late.

(Do not consider trains which arrive after the end of the day or before the beginning of the day.)

Scheduled	Actual	Lateness
9:32	9:34	0
10:32	10:44	12
11:32	11:32	0
12:32	12:37	0
13:32	13:30	0
14:32	15:31	59
15:32	16:44	72
16:32	cancelled	120

In the class `Train` trains are represented by their scheduled and actual arrivals. Cancelled trains are represented by their actual arrival times being recorded as `-1`.

```
public class Train {
    private int timetableHour;
    private int timetableMinute;
    private int actualHour;
    private int actualMinute;

    public Train(int tH, int tM, int aH, int aM) {
        timetableHour = tH;
        timetableMinute = tM;
        actualHour = aH;
        actualMinute = aM;
    }
    public int getActualHour() {
        return actualHour;
    }
    public int differenceInMinutes() {
        return   60 * actualHour + actualMinute
               - 60 * timetableHour - timetableMinute;
    }
}

    import java.util.*;
```

```
public class TrainsTest {
    public static void main(String[] args) {
        ArrayList<Train> trains = new ArrayList<Train>();

        trains.add(new Train(9,32,9,34));
        trains.add(new Train(10,32,10,44));
        trains.add(new Train(11,32,11,32));
        trains.add(new Train(12,32,12,37));
        trains.add(new Train(13,32,13,30));
        trains.add(new Train(14,32,15,31));
        trains.add(new Train(15,32,16,44));
        trains.add(new Train(16,32,-1,-1));

        ArrayList<Integer> l = lateness(trains);
        for (int i : l) {
            System.out.println(i);
        }
        System.out.println("Late proportion " + lateProp(l));
    }
....
}
```

Implement a method `ArrayList<Integer> lateness(ArrayList<Train> trains)` to store the latenesses of trains. Furthermore implement a method `double lateProp (ArrayList<Integer> late)` that returns the proportion `p` of trains that run late. [6%]

[PART B. USE A SEPARATE ANSWER BOOK.]

Question 4 [This question is about recursion and data structures] Total: **20%**

The classes `List` and `Tree` are those used in Lectures/Worksheets for writing recursive functions.

(a) Consider a function for removing duplicate copies of elements in a sorted list.

`static List removeDuplicates (List a)`

The input list *a* is required to be sorted in ascending order. The returned list will be again sorted in ascending order, and contain the same elements as *a* but with no duplicate copies of elements.

Decide 5 test cases to test this function. Make sure that all the "boundary cases" are tested.

Use a table such as the one below to mention (i) what aspect of the function behaviour you are trying to test, (ii) the input list, and (iii) the expected result or effect. [5%]

You may write lists using the square bracket notation, e.g., $[1, 2, 3]$. *Try to make your test cases small*.

no.	behavioural aspect	input list	expected result/effect
1			

(b) Consider the following method for removing duplicates:

```
static List removeDuplicates (List a) {
    /** @param a - a list of integers sorted in asending order
     *  @return a sorted list with the same elements as
     *          a, but with no duplicate copies of elements.
     */
    if (a.isEmpty())
        return empty();
    else if (a.head() == a.tail().head())
        return removeDuplicates(cons(a.head(), a.tail()));
    else
        return cons(a.head(), removeDuplicates(a.tail()));
}
```

For each of the test cases produced in part (a), mention

- whether the method passes the test and,
- if it does not pass, what will be the actual result or the effect seen. [5%]

(c) The above method is known to be buggy. What changes must be made to the method above to produce a correct algorithm? [5%]

(d) Write a method that, given a binary search tree as an argument, produces a list containing its node values in the ascending order.

The method declaration should be:

```
static List sortedList(Tree t)
```

If the search tree contains multiple copies of any value, all of them should be retained in the output list. [5%]

API Reference:

class List:

```
static List    empty     ();
static List    cons      (int a, List l);
static boolean isEmpty   (List l);
static int     head      (List l);
static List    tail      (List l);
```

class Tree:

```
Tree             ();                       -- constructor
Tree             (int a, Tree l, Tree r); -- constructor
boolean isEmpty  ();                       -- instance method
int     value    ();                       -- instance method
Tree    left     ();                       -- instance method
Tree    right    ();                       -- instance method
```

Question 5 [This question is about the Java type system and collection classes] Total: **15%**

(a) The following Java method is expected to create and return the set of integers $\{1, 4, 6\}$. Fill in the blanks. Use an implementation of sets of your choice. [3%]

```
public static ____________________ mySet() {

    ____________________ result = new ________________________();

    result.add(1); result.add(4); result.add(6);
    return result;
}
```

(b) Given below is a class called `Name` for names of people (including their last name and first name), and a program to insert two names into a `TreeSet` of Names.

```
public class Name {
    public String lastName, firstName;
    public Name(String last, String first) {
        lastName = last; firstName = first;
    }
}

public class NameTest {
    public static void main (String[] args) {
        Collection<Name> c = new TreeSet<Name>();
        c.add(new Name("Reddy", "Uday"));
        c.add(new Name("Kerber", "Manfred"));
    }
}
```

When an attempt is made to run the `main` method, we encounter an error:

```
Exception in thread "main" java.lang.ClassCastException:
   Name cannot be cast to java.lang.Comparable
```

- Why does the error arise?
- What changes should be made to the `Name` class to correct the problem? [6%]

(c) A type *A* is said to be a *subtype* of another type *B* if, in *every context* where a *B*-typed value is needed, an *A*-typed value can be used in its place. ("Liskov substitution principle")

Java types include *classes* and *interfaces*. What kind of subtype relations are possible between them? [3%]

(d) Even if *A* is a subtype of *B* (e.g., `Car` subtype of `Vehicle`), Java does not permit `Collection<Car>` as a subtype of `Collection<Vehicle>`. Explain why this is a reasonable choice made by Java designers, by appealing to the Liskov substitution principle. [3%]

Question 6 [This question is about Swing user interfaces and threads]

Total: **15%**

(a) Given below is the skeleton for a simple `JPanel` class displaying a counter.

```
public class CounterPanel extends JPanel {
    int count = 0;
    JTextField countDisplay = new JTextField("     0");
    JButton upButton = new JButton("Up");
    JButton downButton = new JButton("Down");

    public CounterPanel() {
        setPreferredSize(new Dimension(300, 100));
        add(countDisplay);
        add(upButton);
        add(downButton);
        setButtonActions();
    }
}
```

The button labelled "Up" is meant to increase the value of the counter displayed by 1 and the button labelled "Down" is meant to decrease it by 1. However, the buttons do not yet have any actions defined.

- Define actions for the button objects using inner classes, and
- define the method `setButtonActions` to link the actions to buttons. [6%]

Shown on the following page is a Java program using threads. Answer the remaining questions with respect to this program.

```
public class Worker extends Thread {
    Object toolA;
    Object toolB;

    public Worker (Object toolA, Object toolB) {
        this.toolA = toolA;
        this.toolB = toolB;
    }
    private void pretendWork() throws InterruptedException {
        sleep(1000);
    }
    public void run() {
        try {
           while (true) {
               synchronized (toolA) {
                   synchronized (toolB) {
                       pretendWork();
                   }
               }
           }
        }
        catch (InterruptedException e) { // clean up
        }
    }
}

public class Test {
    public static void main(String [] args) {
        Object hammer = new Object();
        Object screwDriver = new Object();

        Worker worker1 = new Worker(hammer, screwDriver);
        Worker worker2 = new Worker(screwDriver, hammer);

        worker1.start();
        worker2.start();
    }
}
```

(b) Explain the role of the `start` method, mentioning parallel threads. [3%]

(c) Explain the role of the `synchronized` statement, mentioning locks. [3%]

(d) The program may deadlock. Explain why, giving an example situation. [3%]