# BIRKBECK

(University of London)

MSc EXAMINATION FOR INTERNAL STUDENTS

*MSc Computer Science*

Department of Computer Science and Information Systems

# Programming in Java

BUCI033S7

**DATE OF EXAMINATION:** Wednesday, 11th June 2014
**TIME OF EXAMINATION:** 10:00–13:00
**DURATION OF PAPER:** Three hours

**WITH OUTLINE SOLUTIONS**

RUBRIC:

1. Candidates should attempt ALL 9 questions on this paper.
2. You are advised to look through the entire examination paper before getting started, in order to plan your strategy.
3. Simplicity and clarity of expression in your answers is important.
4. All programming questions should be answered using the Java programming language unless stated to the contrary.
5. Electronic calculators are NOT allowed.
6. **Start each question on a new page**.

| Question: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| Marks: | 5 | 12 | 16 | 10 | 14 | 8 | 10 | 15 | 10 | 100 |

Question 1 .......................................................... Total: 5 marks

What are the values of the elements of the `ArrayList v` when the following code completes execution (as indicated by the output statement)?

```
import java.util.ArrayList;

public class ArrayListExample {
    public static void main(String[] args) {
        ArrayList<Integer> v = new ArrayList<>();
        v.add(1);
        v.add(2);
        v.add(3);
        v.add(4);
        v.add(5);
        for (int i = 0; i < v.size(); i += 1) {
            int x = v.remove(i);
            v.add(x);
        }

        int count = 0;
        for (int i : v) {
            System.out.println("v[" + count + "] = " + i);
            count++;
        }
    }
}
```

**Solution:**

```
v[0] = 2
v[1] = 4
v[2] = 1
v[3] = 5
v[4] = 3
```

Question 2 .......................................................... Total: 12 marks

You have a `List<Character>` that contains a simple mathematical expression involving positive integers with one digit (0, 1, ..., 9), plus signs, and minus signs, with no parentheses. For example, the list might contain `[0, +, 1, - 9, - 5]` representing the expression `0 + 1 - 9 - 5`.

Write a recursive method that returns `true` if the list contains a valid expression and `false` otherwise.

**Solution:** There are at least two ways to solve this problem, so alternative solutions are acceptable. Assume for now that the original list always holds a valid expression. The shortest valid expression is a single digit (which establishes the base case). Every other valid expression always starts with a digit followed by a plus or minus sign (which establishes the recursive case for the first solution).

This solution recursively examines the first two elements of the list. If the first element is a digit and the second element is a plus or minus sign then the method removes the first two elements from the list and recursively examines the rest of the list.

```
// only the recursive method required for the answer.
// rest of the code for testing and completeness.

package samples.recursion;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

public class MathList {
    public static boolean isDigit(Character c) {
        String s = c.toString();
        return s.matches("\\d");
    }

    public static boolean isOperator(Character c) {
        String s = c.toString();
        return s.equals("+") || s.equals("-");
    }

    public static boolean isValid(List<Character> s) {
        boolean result = false;
        // base case: list is empty (or null)
        if (s == null || s.isEmpty()) return result;

        // base case: list of one element
        if (s.size() == 1) {
            result = MathList.isDigit(s.get(0));
        } else {
            Character first = s.get(0);
            Character second = s.get(1);
            if (MathList.isDigit(first) && MathList.isOperator(second)) {
                LinkedList<Character> t = new LinkedList<>(s);
                t.remove();
                t.remove();
                result = MathList.isValid(t);
            }
        }
        return result;
    }

    public static void main(String[] args) {
        // valid
        List<Character> s1 = Arrays.asList('1');
        List<Character> s2 = Arrays.asList('2', '+', '3');
        List<Character> s3 = Arrays.asList('0', '-', '1', '+', '2', '-', '5');
        // invalid
        List<Character> s4 = Arrays.asList();
        List<Character> s5 = Arrays.asList('-');
        List<Character> s6 = Arrays.asList('9', '+', '3', '-');
        List<Character> s7 = Arrays.asList('+', '7', '-');

        System.out.println(MathList.isValid(s1));
        System.out.println(MathList.isValid(s2));
        System.out.println(MathList.isValid(s3));
        System.out.println(MathList.isValid(s4));
        System.out.println(MathList.isValid(s5));
        System.out.println(MathList.isValid(s6));
        System.out.println(MathList.isValid(s7));
    }
}
```

Really the tests should use an appropriate framework, e.g., junit, but that would expand the answer even further.

Question 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Total: 16 marks

(a) Write a generic method to exchange the positions of two different elements in an array. 4 marks

**Solution:**

```
package generics;

public class Exchange {
    public static <T> void swap(T[] a, int i, int j) {
        T temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

(b) If the compiler erases all type parameters at compile time, why should you use generics? 2 marks

**Solution:** You should use generics because:

- The Java compiler enforces tighter type checks on generic code at compile time.
- Generics support programming types as parameters.
- Generics enable you to implement generic algorithms.

(c) What is the following method converted to after *type erasure*? 2 marks

```
public static <T extends Comparable<T>>
    int findFirstGreaterThan(T[] at, T elem) {
    // ...
}
```

**Solution:**

```
public static int findFirstGreaterThan(Comparable[] at, Comparable elem) {
    // ...
}
```

(d) Write a generic method to find the maximum element in an (unsorted) list with `end - 1` elements. You search should start at the element indexed by the parameter `begin`. 8 marks

**Solution:**

```
package generics;

import java.util.List;

public class Maximal {
    public static <T extends Object & Comparable<? super T>>
    T max(List<? extends T> list, int begin, int end) {
        T maxElem = list.get(begin);

        for (++begin; begin < end; ++begin)
            if (maxElem.compareTo(list.get(begin)) < 0)
                maxElem = list.get(begin);
        return maxElem;
    }
}
```

Other solutions are equally acceptable.

Question 4 .......................................................... Total: 10 marks

In Java, boolean assignments are always atomic — there is no need to use `synchronized` even in the presence of concurrent threads. Using this fact a smart Java programmer has written a class `ResourceLock` to control access to one single resource as follows:

```
package concurrent;

class ResourceLock { //for controlling access to one resource only
    private boolean taken = false;

    //acquire the lock
    public void acquireLock() {
        taken = true;
    }

    //release the lock
    public void releaseLock() {
        taken = false;
    }
}
```

Assume that a shared object of the `ResourceLock` class will be used concurrently by many threads.

(a) Explain why the `ResourceLock` class may not work correctly. 4 marks

(b) Correct the code for `ResourceLock` to avoid the access problems. 6 marks

**Solution:** The `ResourceLock` **class** does not have synchronisation so many threads will be able to acquire the lock and use the resource even if we make the methods `synchronized`. We need to block other threads once the lock has been acquired:

```
package concurrent;

public class ResourceLockFixed {
    //for controlling access to one resource only
    private boolean taken = false;

    //acquire the lock
    public synchronized void acquireLock() throws InterruptedException {
        while (taken)
            wait();
        taken = true;
    }

    //release the lock
    public synchronized void releaseLock() {
        taken = false;
        notifyAll();
    }
}
```

A solution using semaphores is also possible but unlikely:

```
package concurrent;
```

```
import java.util.concurrent.Semaphore;

public class ResourceLockSem {
    //for controlling access to one resource only
    private Semaphore rlock = new Semaphore(1);
    //since there is one resource only

    //acquire the lock
    public void acquireLock() throws InterruptedException {
        rlock.acquire();
    }

    //release the lock
    public void releaseLock() {
        rlock.release();
    }
}
```

Question 5 ............................................................ Total: 14 marks

Write unit tests for a class called `Binary` that represents an immutable non-negative binary integer of arbitrary length (possibly more than 64 bits). The API for the class is as follows:

`public class Binary`

| | | |
|---|---|---|
| | `Binary(String s)` | `construct binary integer from string` |
| `String` | `toString()` | `string representation` |
| `int` | `length()` | `number of bits` |
| `Binary` | `not()` | `bitwise not of this binary integer` |
| `Binary` | `xor(Binary b)` | `bitwise xor of two binary integers` |
| `int` | `leadingZeros()` | `number of leading zeros` |
| `boolean` | `isGreaterThan(Binary b)` | `is this binary integer` $>$ `b?` |

A more detailed explanation of each method follows:

**Constructor** The constructor creates a binary integer based on the string argument. It throws a `RuntimeException` if the string argument contains a character other than `'0'` or `'1'`. Leading zeros are permitted.

**String representation** The `toString()` method returns a string representation of the binary integer.

**Length** The `length()` method returns the number of bits (including any leading zeros).

**Bitwise not** The `not()` method returns a binary integer (of the same length) in which all of the bits are flipped.

**Bitwise xor** The `xor()` method returns a binary integer (of the same length) which is the bitwise *exclusive or* of the two integers. Throws a `RuntimeException` if the two binary integers have different lengths.

**Number of leading zeros** The `leadingZeros()` method returns the number of consecutive leading (leftmost) zeros in the binary integer. For example, the number of leading zeros in the binary integer 00011110 is 3.

**Comparison** The `isGreaterThan()` method returns `true` if the invoking binary integer is greater than the argument binary integer, and `false` otherwise. The two binary integers can be of different lengths. Don't worry about negative integers.

You should write one unit test for each method.

The following test client demonstrates some of the desired behaviour of the class:

```java
package binary;

public class BinaryTest {
    public static void main(String[] args) {
        Binary a = new Binary("00011110");
        Binary b = new Binary("01010000");
        System.out.println("a\t\t\t= " + a);
        System.out.println("b\t\t\t= " + b);
        System.out.println("a.length()\t\t= " + a.length());
        System.out.println("a.not()\t\t= " + a.not());
        System.out.println("a.xor(b)\t\t= " + a.xor(b));
        System.out.println("a.leadingZeros()    = " + a.leadingZeros());
        System.out.println("a.isGreaterThan(b)  = " + a.isGreaterThan(b));
    }
}
```

Which produces the following output:

```
a = 00011110
b = 01010000
a.length()         = 8
a.not()            = 11100001
a.xor(b)           = 01001110
a.leadingZeros()   = 3
a.isGreaterThan(b) = false
```

**Solution:** Any tests are acceptable that make use of `@Test`, `@Before`, etc. and the appropriate `assertXXX` forms. The exceptions conditions should also be dealt with.

Question 6 ........................................................................ Total: 8 marks

Consider the following classes:

```
package overload;

public class Test {
    public static void main(String[] args) {
        Base b = new Derived();
        b.methodOne(9);
    }
}

class Base {
    public void methodOne() {
        System.out.println("A");
        methodTwo();
        System.out.println("Z");
    }

    public void methodOne(int a) {
        System.out.println("W");
        methodTwo();
        methodTwo(new Integer(a));
    }

    public void methodTwo() {
        System.out.println("P");
    }

    public void methodTwo(Integer x) {
        System.out.print("B");
    }
}

class Derived extends Base {
    @Override
    public void methodOne(int a) {
        super.methodOne(a);
        System.out.print("X");
    }

    public void methodOne(Integer a) {
        super.methodOne();
        System.out.print("C");
    }

    @Override
    public void methodTwo() {
        super.methodTwo();
        System.out.println("D");
    }

    public void methodTwo(Integer x) {
        System.out.println("!");
    }

}
```

What is printed as a result of the call `b.methodOne(9)` on line 6? You should explain your answer by showing the steps you followed to produce the output.

**Solution:**

```
W
P
D!
X
```

Question 7 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Total: 10 marks

What would be the result of executing the following code? You should show your working.

```
package sample;

public class MaxClass {
    public static void main(String args[]) {
        try {
            CC c = new CC();
            System.out.println(c.max(13, 29));
        } catch (RuntimeException rte) {
            System.out.println(rte);
        } finally {
            System.out.println("In finally of main");
        }
    }
}

class AA {
    int max(int x, int y) {
        try {
            if (x > y)
                x++;
            else
                throw new Exception("Oh Dear!");
            System.out.println("A::max value of x is " + x);
        } catch (Exception ex) {
            System.out.println("In exception " + ex.getMessage());
            System.out.println("x = " + x + " y = " + y);
            return y;
        } finally {
            System.out.println("A::max finally block");
            throw new IllegalArgumentException("A::max Finally x = " + x);
        }
    }
}

class CC extends AA {
    public int max(int x, int y) {
        return super.max(x + 10, y + 10);
    }
}
```

**Solution:**

```
In exception Oh Dear!
x = 23 y = 39
A::max finally block
java.lang.IllegalArgumentException: A::max Finally x = 23
In finally of main
```

2 for each line + 2.

Question 8 .......................................................... Total: 15 marks

A word ladder puzzle is one in which you try to connect two given words using a sequence of English words such that each word differs from the previous word in the list in exactly one letter position. For example,

TEST → **B**EST → BE**E**T → BEE**S** → B**Y**ES → **E**YES → E**V**ES → EVE**R** → **O**VER

You are required to write a method `isValidPair` that is used by a program that checks the correctness of a word ladder entered by the user. The program reads in a sequence of words and checks that each word in the sequence follows the rules for word ladders, where each line entered by the user must:

(a) be a legitimate English word,

(b) have the same number of characters as the preceding word, and

(c) differ from the preceding word in exactly one character position.

Implementing the first condition requires that you have some sort of dictionary available and you may therefore assume the existence of a `Dictionary` class that has the following method:

```
    boolean isWord(String str)
```

which takes a word and returns true if that word is in the dictionary. You may assume that you have access to such a dictionary via the following instance variable declaration:

```
    Dictionary dict = new Dictionary("english")
```

All words in the dictionary are in upper case.

The javadoc for `isValidPair` is as follows:

```
/**
 * Checks to see if it is legal to link the two words in a word ladder
 * @param previousWord previous word in the ladder
 * @param currentWord current word in the ladder
 * @return true if the pair of words is a valid adjacent pair, otherwise false
 */
boolean isValidPair(String previousWord, String currentWord)
```

Complete the definition of the method `isValidPair`.

If you wish you may define additional helper methods.

**Solution:**

```
boolean isValidPair(String previousWord, String currentWord) {
    Dictionary dict = new Dictionary(fileName: "english");
    if (!dict.isWord(currentWord)) return false;
    if (previousWord == null) return true;
    if (previousWord.size() != currentWord.size()) return false;
    return countCharacterDifferences(previousWord, currentWord) == 1;
}

int countCharacterDifferences(String s1, String s2) {
    int count = 0;
    (0..<s1.size()).each { i ->
        if (s1[i] != s2[i]) {
            count++
        }
    }
    return count;
}
```

Could provide a recursive solution for `countCharacterDifferences` but the iterative version is more likely (we do not expect the Java 8 features — a simple loop structure will suffice.

Question 9 .......................................................... Total: 10 marks

The interface `MultiSet`, shown below, is used to maintain a collection of `String`s in which duplicates are allowed.

```
package sets;

interface MultiSet {
    // returns the size of the MultiSet
    int size();

    // Return true if s is in the MultiSet
    boolean contains(String s);

    // Return the number of occurrences of s in the MultiSet
    int occurrences(String s);

    // Add a new item s to the MultiSet
    void add(String s);

    // Remove an item s from the MultiSet
    // If s is not present in the MultiSet, return false.
    // If s is present with count 1, remove s from the MultiSet.
    // If s is present with count > 1, reduce the count by 1.
    // If s was present return true.
    boolean remove(String s);
}
```

The class `MultiSetMap` implements `MultiSet` using a map in which the key is a `String` and the value is the number of occurrences of that string.

You are required to provide implementations for the required methods in `MultiSetMap`, namely, `size`, `contains`, `occurrences`, `add`, and `remove`.

*Note:* If, for example, the map contains `[ "hello":2, "word":1, "zoo":1 ]` then the method `size()` should return 4.

**Solution:**

```
package sets;

import java.util.Map;

class MultiSetMap implements MultiSet {
    private Map<String, Integer> counts;

    @Override
    public int size() {
        int total = 0;

        for (int x : counts.values())
            total += x;
        return total;
    }

    @Override
    public boolean contains(String s) {
        return counts.get(s) != null;
    }

    @Override
    public int occurrences(String s) {
        return counts.get(s);
    }

    @Override
    public void add(String s) {
        if (counts.get(s) == null)
            counts.put(s, 1);
        else
            counts.put(s, counts.get(s) + 1);
    }

    @Override
    public boolean remove(String s) {
        Integer count = counts.get(s);

        if (count == null)
            return false;

        if (count == 1)
            counts.remove(s);
        else
            counts.put(s, count - 1);

        return true;
    }
}
```

There are neater solutions possible but we would expect something like the above.