Java: Practice & Assignments

Software Engineering Lab
Dept. of CSE, IIT Kharagpur

Instructions

1. Use gedit or any other text editor for this lab. Do not use NetBeans/Eclipse or any other IDE.
2. Compile and execute your code from the terminal using javac and java, respectively.
3. Please store the files for each problem in a separate directory.
4. When compressing the directories, please use a `.tar.gz` or `.zip` format.
5. Marks division:
   - Practice: 25%
   - Assignments: 75%

1 Problems for Practice

Expected Time Required to Complete: 50 minute

1.1 Classes and Objects

- **Sample code**: See Figure 1. (Please rename the `getPoint` method as `setPoint`.)
- **Objective**: Get familiar with defining classes containing fields/attributes/members and behaviors/methods. Subsequently, instantiate objects of the class so defined.
- **Duration**: 10 minute

```
class Point
{
  int x,y;
  void getPoint ( int a, int b ) {
    x = a;
    y = b;
  }
}

// definition of another class. This is a main class
class PointsDemo
{
  float distance;
  public static void main (String args[ ] {
    Point p1 = new Point( );
    Point p2 = p1;
    Point p3 = new Point ( );
    Point p4 = new Point ( );
    p1.getPoint (5, 10 );
    p2.getPoint (15, 20);
    p3.getPoint (20, 30);
    p4.getPoint (30, 40);

    System.out.println (" X1 = " + p1.x +  "Y1  = " + p1.y );
    System.out.println ("X2=" + p2.x + "Y2 = " +p2.y );
    int dx = p3.x - p4. x;                    // X2 - X1
    int dy = p3.y - p4. y;              // y2 - y1
    distance = Math.sqrt (dx * dx + dy * dy );// (X2-X1)2 + (Y2-Y1)2
    System.out.println ( " Distance = "+ distance );
  }
}
```

Figure 1: Code for practice problem 1.1.

1.2 2D Arrays

- **Sample code**: See Figure 2.
- **Objective**: Get familiar 2D arrays in Java.
- **Duration**: 5 minute

1.3 Working with Stack

- **Sample code**: See Figure 3.
- **Objective**: Get familiar with the `Stack` class provided by Java library.
- **Duration**: 5 minute

```
class TwoDArray {
  public static void main(String args[]) {
    int twoD[][]= new int[4][5];
    int i, j, k = 0;
    for(i=0; i<4; i++)
    for(j=0; j<5; j++) {
      twoD[i][j] = k;
      k++;
    }
    for(i=0; i<4; i++) {
      for(j=0; j<5; j++)
        System.out.print(twoD[i][j] + " ");
      System.out.println();
    }
  }
}
```

Figure 2: Code for practice problem 1.2.

Fix any error that might be present in the given code.

```
import java.util.*;

public class StackDemo{
  public static void main(String[] args) {
    Stack stack=new Stack();
    stack.push(new Integer(10));
    stack.push("a");
    System.out.println("The contents of Stack is" + stack);
    System.out.println("The size of an Stack is" + stack.size());
    System.out.println("The number poped out is" + stack.pop());
    System.out.println("The number poped out is " + stack.pop());
    //System.out.println("The number poped out is" + stack.pop());
    System.out.println("The contents of stack is" + stack);
    System.out.println("The size of an stack is" + stack.size());
  }
}
```

Figure 3: Code for practice problem 1.3.

1.4 Method Overloading

- **Sample code**: See Figure 4.
- **Objective**: Get familiar with method overloading, where multiple methods exist in a given class with the same name, but different signatures.
- **Duration**: 10 minute

Fix any error that might be present in the given code.

```
class OverloadDemo {
  void test() {
    System.out.println("No parameters");
  }

  // Overload test for one integer parameter.
  void test(int a) {
    System.out.println("a: " + a);
  }

  // Overload test for two integer parameters.
  void test(int a, int b) {
    System.out.println("a and b: " + a + " " + b);
  }

  // overload test for a double parameter
  double test(double a) {
    System.out.println("double a: " + a);
    return a * a;
  }
}

class Overload {
  public static void main(String args[]) {
    OverloadDemo ob = new OverloadDemo();
    double result;
    // call all versions of test()
    ob.test();
    ob.test(10);
    ob.test(10, 20);
    result = ob.test(123.25);
    System.out.println("Result of ob.test(123.25): " + result);
  }
}
```

Figure 4: Code for practice problem 1.4.

1.5 Access Modifiers

- **Sample code**: See Figures 5 (a) & (b).
- **Objective**: Get familiar with different access modifiers available in Java.
- **Duration**: 15 minute

Some statements in this class may cause compilation error. Find them and comment them out.

```
class BaseClass {
  public int x = 10;
  private int y = 10;
  protected int z = 10;
  int a = 10; //Implicit Default Access Modifier

  public int getX() {
    return x;
  }

  public void setX(int x) {
    this.x = x;
  }

  private int getY() {
    return y;
  }

  private void setY(int y) {
    this.y = y;
  }

  protected int getZ() {
    return z;
  }

  protected void setZ(int z)
    this.z = z;
  }

  int getA() {
    return a;
  }

  void setA(int a) {
    this.a = a;
  }
}
```

(a)

```
public class SubclassInSamePackage extends BaseClass {
  public static void main(String args[]) {
    BaseClass rr = new BaseClass();
    rr.z = 0;
    SubclassInSamePackage subClassObj = new SubclassInSamePackage();
    //Access Modifiers - Public
    System.out.println("Value of x is : " + subClassObj.x);
    subClassObj.setX(20);
    System.out.println("Value of x is : " + subClassObj.x);
       System.out.println("Value of y is : "+subClassObj.y);
    subClassObj.setY(20);
    System.out.println("Value of y is : "+subClassObj.y);

    //Access Modifiers - Protected
    System.out.println("Value of z is : " + subClassObj.z);
    subClassObj.setZ(30);
    System.out.println("Value of z is : " + subClassObj.z);
    //Access Modifiers - Default
    System.out.println("Value of x is : " + subClassObj.a);
    subClassObj.setA(20);
    System.out.println("Value of x is : " + subClassObj.a);
  }
}
```

(b)

Figure 5: Code for problem 1.5.

1.6 Interfaces and Inheritance

- **Sample code**: See Figure 6.
- **Objective**: Get familiar with interfaces and their inheritance.
- **Duration**: 5 minute

Some statements from the class `B1` *have been removed. Complete them so that the resulting code compiles.*

2 Assignments

Expected Time Required to Complete: 115 minute

```
interface I1 {
  void methodI1(); //public static by default
}

interface I2 extends I1 {
  void methodI2(); //public static by default
}



class A1 {
  public String methodA1() {
    String strA1 = "I am in methodC1 of class A1";
    return strA1;
  }

  public String toString() {
    return "toString() method of class A1";
  }
}

class B1 extends A1 implements I2 {

    System.out.println("I am in methodI1 of class B1");
  }


    System.out.println("I am in methodI2 of class B1");
  }
```

Figure 6: Code for practice problem 1.6.

2.1 Factorial Using Recursion

- **Objective**: Define a method `long factorial(int x)()` that returns the factorial of any integer $1 \leq x \leq 10$ computed by recursive call to itself. Pass the value of x as a command line argument to your program.
- **Duration**: 10 minute
- **Difficulty level**: Easy

2.2 Class and Inheritance

- **Objective**:Define two different classes namely, Student and Employee. These classes are derived from a base class Person. Define other two classes Staff and Faculty. Staff and Faculty classes are derived from Employee class. The Person class has name and age data and display method to display the name and age of a person. The Student class has data like rollNo and branch and display method to display name, age, rollNo and branch of the student. Stuff has ecNo and doj(date of joining) data and display method to display name, age, ecNo, doj of the stuff. Faculty has designation data (Assistant Professor, Associate Professor and Professor) and display method to display the name, age, ecNo, doj and designation

of the Faculty. Staff has designation data (Technical and Clerical) and display method to display the name, age, ecNo, doj and designation of the Staff. Each class have their own constructor to initialize the value of each data field. Finally create MainDemoClass and create an object of each class. Print the values of all objects in the MainDemoClass.

- **Duration**: 20 minute
- **Difficulty level**: Easy

2.3 Class Diagram to Code

- **Objective**: Translate the class diagram shown in Figure 7 into Java code.
- **Duration**: 5 minute
- **Difficulty level**: Easy

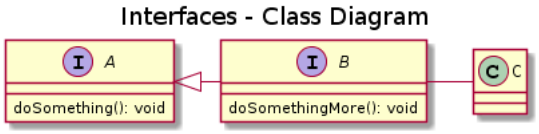

Figure 7: Class diagram for assignment 2.3.

2.4 Code to Class Diagram

- **Objective**: An inner class is one, whose existence is only known to its outer class. Rest of the world does not know about that inner class. In Figure 8, `Kitchen` and `Bathroom` are two inner classes known only to their outer class, `House`.

  Based on the code shown in Figure 8, draw a class diagram relating the `House`, `Kitchen`, and `Bathroom` classes. The specific forms of association should be shown together with multiplicity wherever relevant.

- **Duration**: 10 minute
- **Difficulty level**: Medium

Correction: Line 12 in Figure 8 would be `bathroom1 = new Bathroom();`.

2.5 Dynamic Arrays

- **Objective**: An array is a static data structure in the sense that, once its size has been specified during initialization, the size cannot be altered. However, pre-specifying a fixed size may cause underuse of space (has less

```java
class Table { }

class Painter {
    void doPaint() { }
}

class House {
    Table[] tables;
    Kitchen kitchen;
    Bathroom bathroom1;
    Bathroom bathroom2;

    House(Table[] tables) {
        this.tables = tables;
        kitchen = new Kitchen();
        bathroom1 = new Bathroom();
        bathroom2 = new Bathroom();
    }

    void paint(Painter painter) {
        painter.doPaint();
    }

    private class Kitchen { }

    private class Bathroom { }
}
```

Figure 8: Code for assignment 2.4.

elements than the desired size) or overuse (does not have enough space to add new elements). Implement a `DynamicArray` (of integers) class to address this issue by supporting the following functionality.

- Begin with a small array (the "original array") of size say, 10. This should be declared as private.
- The following public operations are supported:
  - `add(int x)`: Adds a given element to the end of the existing array.
  - `remove()`: Removes an element, if any, from the end of the array.
  - `size()`: Returns the number of elements currently stored in the array.
  - `print()`: Prints all the elements currently stored in the array.
- When a new integer is to be added, but there isn't enough space available, double the size of the array. To do this, copy the elements into a temporary array, alter the size of the original array, and copy back the elements into the original array. Finally, add the new element.
- If 50% of the array cells are vacant after removing an element, shrink the size of the original array by half.

Illustrate the use by creating a `DynamicArrayTest` class containing the `main` method.

- **Duration**: 30 minute
- **Difficulty level**: Hard

2.6 ArrayList

- **Objective**: The Java library provides the `ArrayList` class, which can be considered as a dynamic array that occupies as much space as the number of elements in it. Figure 9 shows how an `ArrayList` to hold zero or more items of type `String` is created.

  Based on the sample code,

  - Create an `ArrayList` to store `Integer` types.
  - Using a for loop, store the squares of the first 10 natural numbers in the list so created. Use the `add()` method of `ArrayList` for this purpose.
  - Once again, use a for loop to print all the elements.
  - Print the size of the list (you should not count the items individually).
  - Remove all the elements from the list using a single method call.
  - Print whether or not the list is empty using only a single method call (no comparison allowed).

- **Duration**: 20 minute
- **Difficulty level**: Medium

Hint: Find the relevant methods from the API documentation of ArrayList[1].

Note: In Java, `<>` is called the diamond operator, and is used to represent *generics*. For example, an `ArrayList` is a generic type, that can store objects of *any* type – String, Double, Integer, Planet, Star, ArrayList, and so on. In contrast to type casting, generics are *type safe*.

```
1 import java.util.ArrayList;
2
3 public class ArrayListDemo {
4     public static void main(String[] args) {
5         // A list to store items of type String
6         ArrayList<String> alist = new ArrayList<>();
7     }
8 }
```

Figure 9: Sample code for assignment 2.6.

[1] `https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html`

2.7 Sorting Objects

- **Objective**: How to sort objects created from user-defined classes? One way is to have the `Comparable` interface implemented by the class so that the `compareTo()` method is defined. This is illustrated in Figure 10, where we sort an array of objects of user-defined types.

  Based on the given code, define a class `Student`, which has two members – `firstName` and `lastName`. Create an array containing five students. Sort them in ascending order by their first names followed by their last names. Sample sorted output:

```
Aayush Gupta
Ayush Bansal
Ayush Sharma
Siddharth Singh
Yashsvi Dixit
Yashvardhan Singh
```

- **Duration**: 20 minute
- **Difficulty level**: Medium

Hint: The `String` class itself implements the `Comparable` interface.

```java
import java.util.Arrays;

class MyClass implements Comparable<MyClass> {
    private int x;
    public MyClass(int x) { this.x = x; }
    public String toString() { return "" + x; }

    // This method from the Comparable interface has to be implemented;
    // defines how we want to sort two objects of our class.
    public int compareTo(MyClass cls) {
        // Return value: a negative integer, zero, or a positive integer
        // as *this* object is less than, equal to, or greater than the
        // *specified* object.
        // Ascending order
        return this.x - cls.x;
    }
}

public class ObjectSortingDemo {
    public static void main(String[] args) {
        MyClass[] mycls = new MyClass[3];
        mycls[0] = new MyClass(10);
        mycls[1] = new MyClass(1);
        mycls[2] = new MyClass(20);

        Arrays.sort(mycls);

        for (MyClass cls : mycls) {
            System.out.println(cls);
        }
    }
}
```

Figure 10: Sample code for assignment 2.7.