Object-Oriented Programming
Tutorial 3

Exercise 1: Arrays

Question 1.1: Implement a matrix class that allows to manipulate two-dimensional matrices. Your class should implement:

- the initialization of a cell,
- the display of the full matrix,
- the addition of two matrices,
- the multiplication of two matrices (when possible).

Question 1.2: Write a class to manipulate a triangular data structure of the following shape:
#
##
###
A cell is accessed using (row,column). With this data structure, it is possible to access to cells (3,3) and (3,2), but not to cell (2,3). Ensure that you don't allocate unneeded cells.

Exercise 2: Lists

The main class to manipulate lists in Java is java.util.Vector. Among the most used methods of Vector are:

- void add(Object o);
- boolean contains(Object o);
- void remove(Object o);
- int size();

Question 2.1: How to add 1, 2 and 3 in the Vector? How to obtain these integer values from the Vector?

Question 2.2: Implement your own class in order to implement lists such as Vector does.

Question 2.3: Implement a reverse function, that reverses the list. For example, with this function, the list (a,b,c,d) becomes (d,c,b,a).

Exercise 3: Associative arrays

Associative arrays are arrays where the index is not an integer (unlike usual arrays), but an object. The main class in Java to manipulate associative arrays is java.util.Hashtable. A hashtable is implemented using an array of lists. A pair (key, value) is stored within the list of cell hashcode(key). Notice that in Java, there is a method "int hashcode()" in the Object class.

Among the most used functions of Hashtable, there are:

- void put(Object key, Object value);
- boolean contains(Object key);
- Object get(Object key);

Question 3.1: Implement your own class to manipulate associative arrays, such as Hashtable does.

You can use the Vector class.

Question 3.2: Is it possible to modify your class so that there is a method “void put(Object key, int value)”? How to make the difference between an int as a value, and an Integer? Is it possible to have a method “int get(Object key)” used only when the value is an int, but not an Integer?

Exercise 4: Type inheritance

Continuing from tutorial 2, we want to manipulate polygons (such as hexagones). Here are the operations that are allowed:

- to access to the points of the polygon with a single method,
- to perform an homothety by giving the center and the ratio,
- to perform a rotation by giving the center and the angle,
- to display the polygon by showing the coordinates of all its points.

A polygon is represented by a list of points (note that we decided not to use an array here). The main issue with lists is that there are several possible implementations of lists: we have to ensure that our implementation of polygons is not dependent of the implementation of the list.

Here are the operations that we need for the lists:

- to know whether the list is empty or not,
- to add an element at the head of the list,
- to remove the head of the list,
- to access to the first element of the list,
- to access to the next element of the list,
- to modify the current element of the list,
- to remove the current element of the list,
- to access to the current element of the list.

Question 4.1: Propose a class diagram that allows to manipulate polygons of various sizes. We want a diagram that enables to modify the list implementation easily.

Question 4.2: Write the corresponding java classes. Also write a class that allows to test your implementation by testing triangles, quadrangles and hexagons with cartesian points and polar points.

Exercise 5: Abstract classes

Continuing from tutorial 2, we notice that methods toString() and equals() were the same in both Cartesian2DPoint and Polar2DPoint classes. These methods could have been factorized in an abstract class Common2DPoint, with the implementation of these two methods, and other methods declared as abstract methods.

Question 5.1: What is an abstract class? How to declare an abstract class?

Question 5.2: What is an abstract method? How to declare an abstract method?

Question 5.3: Draw the new class diagram and give the implementation of the classes.

Question 5.4: Propose a new class diagram that enables to factorize the code of the geometrical

shapes (from tutorial 2).

Exercise 6: Templates

Let us consider the following code for stacks.

```
public interface Stack {
	public Object top();
	public void clear();
	public void push(Object e);
	public void pop();
}

public class StackImplementation implements Stack {
	private Object[] tab;
	private int top;
	private int numberElements;
	public StackImplementation(int numberElements) {
		tab = new Object[numberElements];
		this.numberElements = numberElements;
		top = -1;
	}
	public boolean empty() {
		return top==-1;
	}
	private boolean full() {
		return top==(numberElements-1);
	}
	public void push(Object e) {
		if (!empty()) {
			tab[++top] = e;
		}
	}
	public void pop() {
		if (!empty()) {
			top--;
		}
	}
	public Object top() {
		if (!empty()) {
			return tab[top];
		}
		else {
			throw new ExecutionException();
		}
	}
}
```

Question 6.1: Is the following code correct?

```
StackImplementation s = new StackImplementation(100);
s.push(“oop”);
s.push(Integer.parseInt(“4”));
s.push(new StringBuffer(“oop2”));
```

Object o = p.top();

Question 6.2: Would it be possible to manipulate objects having the same type, with this stack implementation? Why?

Question 6.3: Propose an implementation that enables to manipulate objects having the same type.

Question 6.4: What is a generic class? Propose an implementation using a generic class. Compare with the previous implementation.

Exercise 7: Method selection

Question 7.1: What does the following program display? Why?

```
public class Parent {
	int i = 1;
	public int f() {
		return i;
	}
}

public class Child extends Parent {
	int i = 5;
	public int f() {
		super.f() + i;
	}
}

public class Test {
	public static void main(String[] args) {
		Parent o = new Child();
		System.out.println(o.i);
		System.out.println(o.f());
	}
}
```

Question 7.2: What does the following program display? Why?

```
public class Figure {
}

public class Circle extends Figure {
}

public class Client1 {
	public void move(Figure f) {
		System.out.println(“I can make this figure move”);
	}
}

public class Client2 extends Client1 {
	public void move(Circle c) {
		System.out.println(“I can make this circle move”);
	}
```

```
}

public class TestMove {
	public static void main(String[] args) {
		Circle c = new Circle();
		Figure f = c;
		Client c1 = new Client1();
		c1.move(c);
		c1.move(f);
		Client c2 = new Client2();
		c2.move(c);
		c2.move(f);
	}
}
```

Question 7.3: How to change the previous program to have the following display?

```
I can make this circle move
I can make this circle move
I can make this circle move
I can make this circle move
```