University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatics I – EProg HS15

Exercise 11

1 Task: OOP

1.1 Learning Objectives

1. Consolidation of mapping problems to code
2. Identify and implement inheritance hierarchies

1.2 Description

- Ben is a male 21 years old informatics student. Of course he can code and he can model too. His favourite musician is *Santana*.
- Luc is one of Ben's friends and is a 20 years old informatics student too. He's got an own car which he calls *Klopfi* because of the sounds it makes while driving. Luc can code too, but he likes more modelling. Luc has no favourite musician.
- Prof. Harald Gall is a male 30 years young professor and the supervisor of a Ben and Luc during their studies.
- Klopfi is the car of Ben. Its colour is blue and its brand is *Toyota*.

1.3 Assignment

From the four objects described above, identify classes, attributes, and methods that can be used to create those objects in Java. Create a reasonable inheritance hierarchy by shifting common behavior into a *parent class*. Make notes and draw an UML class diagram before implementing the classes in Java.

2 Task: Static and Dynamic Types

2.1 Learning Objectives

1. Repetition of static and dynamic types

2.2 Assignment

The following classes are given:

```java
public class Super {
   public void printHello() {
      System.out.println("Hello World!");
   }

   public void printSomething() {
      System.out.println("Something.");
   }
}
```

```java
public class SubGerman extends Super {
   public void printHello() {
      System.out.println("Hallo Welt!");
   }
   public void printGermanHello() {
      System.out.println("Moin!");
   }
}
```

Decide which of the following statements are allowed within the main method of a TestDriver and provide the console output if applicable.

```java
Super x = new Super();
x.printHello();
```

□ Allowed □ Illegal Output:

```java
Super x = new Super();
x.printGermanHello();
```

□ Allowed □ Illegal Output:

```java
Super y = new SubGerman();
y.printSomething();
```

□ Allowed □ Illegal Output:

```
1 SubGerman z = new Super();
2 z.printHello();
```

□ Allowed □ Illegal Output:

```
1 Super y = new SubGerman();
2 y.printGermanHello();
```

□ Allowed □ Illegal Output:

Solution:

☒ Allowed	□ Illegal	Output: Hello World!
□ Allowed	☒ Illegal	Output:
☒ Allowed	□ Illegal	Output: Something.
□ Allowed	☒ Illegal	Output:
□ Allowed	☒ Illegal	Output:

3 Task: Recursion

3.1 Learning Objectives

1. Practice recursive programming.

3.2 Assignment

Implement a method that takes as an input an array of integer and recursively finds the maximal value which is then returned. Make sure you don't use a for loop to find the maximal value. You may pass more input arguments if you like.

```
1 public int findMaximumRecursive(int[] numbers, ...){
2    your implementation...
3 }
```

Solution:

```java
package task3;

public class RecursionDemo {

   public static void main(String[] args) {
      int[] numbers = {12, 1, -88, 75, 3};
      System.out.println("Max of {12, 1, -88, 75, 3}: " + RecursionDemo.
          findMaximumeRecursive(numbers, 0));
   }

   public static int findMaximumeRecursive(int[] numbers, int index){
      if (numbers.length == 0) {
         System.out.println("There is no integer in the array");
      }
      if (numbers.length == 1) {
         return numbers[0];
      }
      if (index == numbers.length - 2) {
         if (numbers[index] > numbers[index + 1]) {
            return numbers[index];
         }
         else {
            return numbers[index + 1];
         }
      }
      else {
         int recursiveMax = findMaximumeRecursive(numbers, index + 1);
         if (numbers[index] > recursiveMax) {
            return numbers[index];
         } else {
            return recursiveMax;
         }
      }
   }
}
```

Listing 1: RecursionDemo

4 Task: The *Stack* Data Structure

4.1 Learning Objectives

1. Implement a stack data structure in different ways.
2. Practice arrays.
3. Practice ArrayLists.
4. Practice linked data structures.
5. Read javadoc.

4.2 Assignment

The interface `IStack` defines the functionality of a typical stack data structure:

```
1  package task4;
2
3  /**
4   * The <code>IStack</code> interface defines the functionality
5   * of a last-in-first-out (LIFO) stack for numbers of the type
6   * <code>long</code>.
7   */
8  public interface IStack {
9     public static final long ERROR_VALUE = Long.MIN_VALUE;
10
11    /**
12     * Pushes a number onto the top of this stack.
13     *
14     * @param number the number to be pushed onto this stack.
15     * @return the <code>number</code> argument.
16     */
17    public long push(long number);
18
19    /**
20     * Removes the number at the top of this stack and returns that
21     * number as the value of this function.
22     *
23     * @return The number at the top of this stack
24     *    or ERROR_VALUE if the stack is empty.
25     */
26    public long pop();
27
28    /**
29     * Looks at the number at the top of this stack without removing it
30     * from the stack.
31     *
32     * @return the number at the top of this stack
33     *    or ERROR_VALUE if the stack is empty.
34     */
35    public long peek();
```

```
36
37      /**
38       * Tests if this stack is empty.
39       *
40       * @return <code>true</code> if and only if this stack contains
41       * no numbers; <code>false</code> otherwise.
42       */
43      public boolean isEmpty();
44
45      /**
46       * Returns the 1-based position where a number is on this stack.
47       * If the number <tt>number</tt> occurs as a number in this stack,
48       * this method returns the distance from the top of the stack to
49       * the first occurrence of the number in the stack (top to bottom);
50       * the topmost number on the stack is considered to be
51       * at distance <tt>1</tt>.
52       *
53       * @param number the desired number.
54       * @return the 1-based position from the top of the stack where
55       * the number is located; the return value <code>-1</code>
56       * indicates that the number is not on the stack.
57       */
58      public int search(long number);
59  }
```

Listing 2: `IStack` interface

a) ArrayStack

Write a class called `ArrayStack` that implements the `IStack` interface by using Java arrays. Provide a default constructor that initializes the size of the array to 2. Double the size of the array if the array is full. Provide a second constructor that allows to specify the initial size of the stack.

b) ArrayListStack

Write a class called `ArrayListStack` that implements the `IStack` interface by using a Java ArrayList. Hint: Use the class `Long`.

c) LinkedStack

Write a class called `LinkedStack` that implements the `IStack` interface by using a linked data structure. Therefore, you have to create an additional class called `StackElement` that contains the actual value of an element within the stack (i.e. stores the number). Additionally, `StackElement` stores a reference another `StackElement` (i.e. the one below itself). Finally, the `LinkedStack` class must keep track of the topmost `StackElement`. Figure 1 illustrates the proposed solution.

d) Generic ArrayListStack

You should implement the ArrayListStack assignment above for any type of class (not only numbers of type long) by using generics.

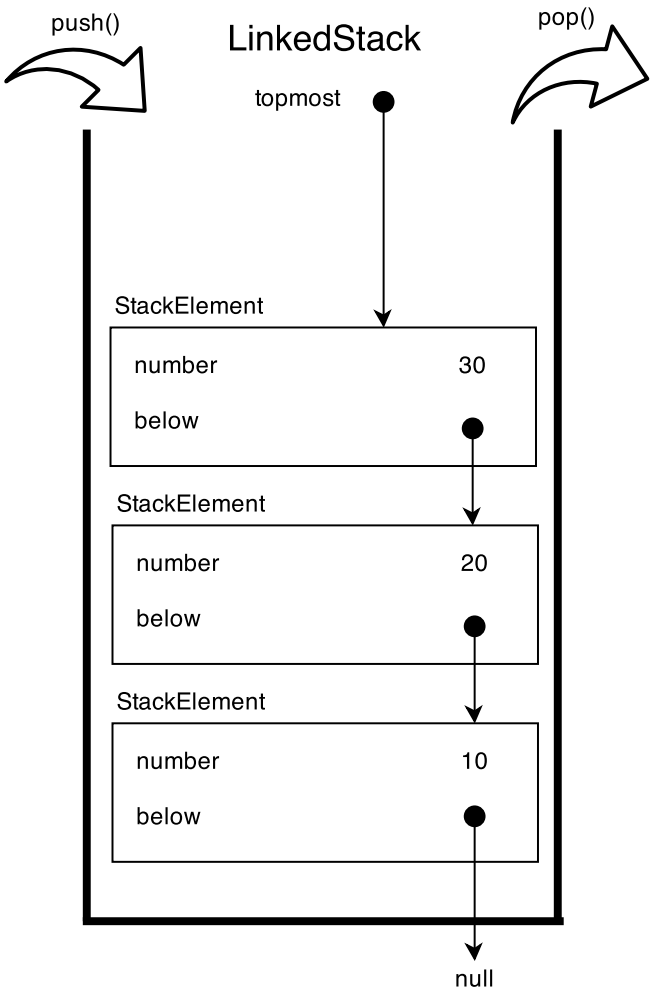

Figure 1: `LinkedStack` conceptual approach