Lecture 5: class activities

prof. dr. Irma Ravkic

Start a new project in your workspace or add a package for Activity 5.

Activity 5.1 Recursion 1: sum

You: Write an **iterative** method that will find a sum of elements in an array.
Together: Write a **recursive** method doing the same.
(*You can find the demonstration of this solution in the textbook*)

Activity 5.2 A bit more about references

Read the following blog to see how Java passes objects around and try the activities below to practice it a bit.

Activity 5.2.1 Passing a wrapper class object to a function

What would you expect to be printed in line 9? Type this code in a class in your project. Is the result what you expected? Why yes/not? How would you change this code to produce the output you expected without changing the type of variable **b**.

```
 1  public class ReferenceDemo{
    public static void incrementIntegerByOne(Integer c){
 3      c = c + 1;
    }
 5
    public static void main(String[] args){
 7      Integer b = new Integer(3);
        incrementIntegerByOne(b);
 9      System.out.println(b);
    }
11  }
```

Activity 5.2.2 Swapping an element in Java not so simple

Create a method called **swap** that will swap two Integers in Java. Test it with the following main method:

```
 public static void main(String[] args) {
    Integer a = new Integer(3);
    Integer b = new Integer(4);
    System.out.println("Before swap: " + a + " and " + b);
    swap(a, b);
    System.out.println("After swap: " + a + " and " + b);
}
```

Create another swap method but now taking two primitive integers as parameters. Change the main method to swap two primitive integers.

Activity 5.2.3 Swapping custom made objects

Create a very simple class called *MutableInteger* that has a single property: a primitive integer variable representing an integer number. Create a swap method that will swap two *IntegerNumber* objects. Test your method with the following main method:

```
public static void main(String[] args) {
    MutableInteger a = new MutableInteger(3);
    MutableInteger b = new MutableInteger(4);
    System.out.println("Before swap: " + a + " and " + b);
    swap(a,b);
    System.out.println("After swap: " + a + " and " + b);
}
```

Your console should show:

```
Before swap: 3 and 4
After swap: 4 and 3
```

Activity 5.3 Recursion 2: traversing a linked list

For this activity you need your LLNode⟨T⟩ code from **support** package I already shared with you. Please re-type it or copy paste it into your project first. And then solve the exercise.
Create a small linked list in your main method and do the following:

You: Write **iterative** methods that will a) traverse a linked list iteratively to print its values, b) calculate the number of elements in a linked list.

Together: Write **recursive** methods doing the same as above. **Note: you can implement all above in a single class by making the methods static. Doing this is not good for the OOP purposes and overusing static methods is wrong, but in this context of learning it is justifiable.**

Test your methods with the following main method (feel free to change the names of methods to fit your implementation):

```
public static void main(String[] args) {
 LLNode<Integer> top = new LLNode<Integer>(3);
 LLNode<Integer> node1 = new LLNode<Integer>(4);
 LLNode<Integer> node2 = new LLNode<Integer>(5);
 LLNode<Integer> node3 = new LLNode<Integer>(6);
 LLNode<Integer> node4 = new LLNode<Integer>(7);
 top.setLink(node1);
```

```java
  node1.setLink(node2);
  node2.setLink(node3);
  node3.setLink(node4);
  iterativeTraversal(top);
  System.out.println("Iterative num elements: ")
  System.out.println(iterativeNumElements(top));
  recursiveLinkedListTraversal(top);
  System.out.println("Recursive num elements: ")
  System.out.println(recursiveNumElements(top));
}
```

Activity 5.4 Recursion 3: transforming a linked list

Continuing on the previous example, for this activity you need your LLNode⟨T⟩ code. Please re-type it or copy paste it into your project first. And then solve the exercise.

You: For the example above in the main method write an **iterative** method with the following method signature:

```java
public static <T> void addToEnd(LLNode<T> lList,
                                LLNode<T> node);
```

that will add a new node to the end of a linked list provided in the input. **Note: you can make the method above generic as well!**. Use a small example, print it to make sure it works correctly.

Together: Write a **recursive** method doing the same as above. **Test it for an empty linked list!**

Activity 5.5 Recursion 4: Towers of Hanoi Demo

The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. He was inspired by a legend that tells of a Hindu temple where the puzzle was presented to young priests. At the beginning of time, the priests were given three poles and a stack of 64 gold disks, each disk a little smaller than the one beneath it. Their assignment was to transfer all 64 disks from one of the three poles to another, with two important constraints. They could only move one disk at a time, and they could never place a larger disk on top of a smaller one. The priests worked very efficiently, day and night, moving one disk every second. When they finished their work, the legend said, the temple would crumble into dust and the world would vanish. Although the legend is interesting, you need not worry about the world ending any time soon. The number of moves required to correctly move a tower of 64 disks is $2^{64-1} = 18,446,744,073,709,551,615$. At a rate of one move per second, that is $584,942,417,355$ years! Clearly there is more to this puzzle than meets the eye.

We will see how this problem can be solved recursively with 3 disks (so without the world vanishing).

You can see the problem setup in the book or in the slides I shared with you. In your class activity package you have a **Tower.java** file. Add it to your existing activity project under a different package called **demos**.