Laboratório 16 – Herança e Polimorfismo IV (24/10/2014)

1.

```
package lab16;

/**
 * Transform is the super class. Here are define all signatures methods and is specified
the executePrint method
 * @author FilipaG
 */
public abstract class Transform {
	private String description;

	public Transform (String d)    // construtor
	{
		this.description = d;
	}

	/**
	 * This method print the inserted string, followed by the initial array and the
final array
	 * with all transformations
	 * @param a
	 * @return
	 */
	public String executePrint(int[] a)
	{
		String msg = this.description + ": [";

		for (int i=0; i<a.length; i++)
		{
			msg = msg + ((i==a.length-1)?(a[i] + "] -> ["):(a[i]+ ", "));
//impressão do array inserido
		}

		int[] res = execute(a);          // executa as tranformações adicionadas.
		for (int i=0; i<res.length; i++)
		{
			msg = msg + ((i==res.length-1)?(res[i] + "]"):(res[i]+ ", "));
//impressão do array com as transformações
		}
		System.out.println(msg);
		return msg;

	}

	public abstract int[] execute(int[] a);
}
```

```
package lab16;
/**
 * SerieTransform allows apply more than one transformation to array
 * @author Filipa
 *
 */
public class SerieTransform extends Transform{
	private int transformationNumber;  // número de transformações inseridas
	private Transform[] transformations;  // array que guardará as transformações
inseridas
```

```
	private int i=0;
	/**
	 * The SerieTransform constructor has the following parameters
	 * @param d
	 * @param transformationNumber
	 */
	public SerieTransform(String d, int transformationNumber) {	// construtor
		super(d);
		transformations = new Transform[transformationNumber];
		this.transformationNumber=transformationNumber;
	}

	/**
	 * This method stores the transformations that will be performed on the array
	 * inserted in an array
	 * @param transformation
	 */
	public void addTransformation(Transform transformation)
	{
		transformations[i]= transformation;
		i++;
	}

	/**
	 * execute, successively performs all entered transformations in the insert array
	 */
	public int[] execute (int[] a)
	{
		int[] b = a;	// executa sucessivamente todas as transformações inseridas
no programa
		for (int i=0; i < transformationNumber;i++)
		{
			b= transformations[i].execute(b);
		}
		return b;
	}
}
```

```
package lab16;

/**
 * The class Inverse obtain the inverse of the array inserted
 * @author Filipa
 *
 */
public class Inverse extends Transform {
	int[] inverseA;
	public Inverse() {
		super("Inverse");
		// TODO Auto-generated constructor stub
	}

	@Override
	public int[] execute(int[] a) {
		inverseA = new int[a.length];
		int i=0;
```

```
            for (int idx = a.length-1; idx>=0; idx--) // coloca o último elemento do
array "a", na primeira posição do array inverseA
            {
                  inverseA[i++]=a[idx];
            }
            return inverseA;   // devolve o array obtido após a transformação
            }
      }
```

```
package lab16;
/**
 * The Unique class features the array without repetition
 * @author Filipa
 *
 */
public class Unique extends Transform{
      private int[] uniqueA;
      public Unique() {
            super("Unique");
      }

      @Override
      public int[] execute(int[] a) {
         int[] aux = new int[a.length];
// como não sabemos à partida quantos elementos serão eliminados por serem repetições
// criamos este array auxiliar com a dimensão do array inicial
// Caso haja repetições e, consequentemente, elementos retirados do array, este array
terá o valor 0 nas últimas posições, tantos quantos os elementos retirados

         int idx=0;
         int countUnique=0; //contagem do número de elementos que fica no array uniqueA

         for (int i=0; i<a.length;i++){
            int count = 0;
            for(int j=0;j<=i; j++){   //compara cada elemento com os que estão antes de
si e a si próprio no array
               if (a[i]==a[j]){    //verificar a existência de repetições
                  count++;
               }
            }
            if (count ==1){
                  aux[idx]= a[i];   // só coloca no novo array os elementos que não
tiverem repetições à sua esquerda
                  idx++;
                  countUnique++;    //conta o número de elementos que foram inseridos
no array auxiliar
            }
         }
         uniqueA = new int[countUnique];
//instanciação de um array com a dimensão countUnique (número de elementos do array
inicial sem números repetidos)

         for (int i=0; i<countUnique;i++)
         {
            uniqueA[i]= aux[i];  //cópia dos elementos válidos do array aux para o array
uniqueA
         }
```

```
        return uniqueA;
        }
}
```

```
package lab16;
/**
 * AscedingSort sorts the array in ascending order
 * @author Filipa
 *
 */
public class AscendingSort extends Transform{

        public AscendingSort() {
                super("AscendingSort");

        }

        /**
         * Sorts the array in ascending order using the BubbleSort
         */
        @Override
        public int[] execute(int[] a) {
                for (int idx=0; idx < a.length; idx++)
// i representa o número de elementos já ordenados
                {
                        for(int j = 1; j < a.length - idx; j++)
// j índice do elemento que está a ser comparado
//trocas até numbers.length - i porque a cada iteração, eu tenhos os i elementos maiores
já ordenados
                        {
                                if(a[j] < a[j - 1])
                                {
                                        int exchangeAux = a [j-1];
                                        a [j-1] = a[j];
                                        a[j] = exchangeAux;
//colocar na variável de apoio o valor da variável que está a ser testada
//colocar no índice da variável que está a ser testada o valor do índice que é mais baixo
                                }
                        }
                }
        return a;
        }
}
```

```
package lab16;

import org.junit.Assert;
import org.junit.Test;

public class TransformTest {

        @Test
        public void testTransformations() {
                // Arrange
                SerieTransform u = new SerieTransform("My Transformation", 3);
                u.addTransformation(new AscendingSort()); //[1,1,4,5,5,6,6,7,8]
```

```
            u.addTransformation(new Unique());     // [1,4,5,6,7,8]
            u.addTransformation(new Inverse());     // [8,7,6,5,4,1]
            int []a = {1,5,6,7,6,8,4,1,5};

            // Act & Assert
            Assert.assertEquals("My Transformation: [1, 5, 6, 7, 6, 8, 4, 1, 5] -> [8,
7, 6, 5, 4, 1]", u.executePrint(a));
      }

}
```

Para verificar o funcionamento do programa (porque me estava a dar erros de compilação principalmente na class Unique) criei a seguinte classe

```
package lab16;

public class App {

      public static void main(String[] args) {
            int []a = {1,5,6,7,6,8,4,1,5};
            SerieTransform u1 = new SerieTransform("My Transformation", 1);
            u1.addTransformation(new Unique());
            u1.executePrint(a);
            SerieTransform u2 = new SerieTransform("My Transformation", 1);
            u2.addTransformation(new AscendingSort());
            u2.executePrint(a);
            SerieTransform u3 = new SerieTransform("My Transformation", 1);
            u3.addTransformation(new Inverse());
            u3.executePrint(a);
      }
}
```

Nota: descobri quais os problemas do programa com a ajuda da Rita.