Programare avansata pe obiecte - laborator 3 (231)

Butan Silvia
silvia.butan@endava.com
butan.silvia@gmail.com

Object oriented programming - following lab 2

The Arrays Class:

- ➢ java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements:
  - **sort(Object[] a)** - Sorts the specified array of objects into an ascending order, according to the natural ordering of its elements. This method could be used by all primitive data types.

```
package com.paolabs.lab3.exemplul1;

import java.util.Arrays;

public class PracticeArrays {

   public static void main(String[] args) {

       float[] values = new float[3];

       values[0] = 10.0f;
       values[1] = 20.0f;
       values[2] = 15.0f;

       Arrays.sort(values);

       System.out.print("The values sorted in ascending order: ");

       for (int i = 0; i < values.length; i++) {
           System.out.print(values[i]);
           System.out.print(",");
       }

   }
}
```

- **equals(long[] a, long[] a2) -** Returns true if the two specified arrays of longs are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. This method could be used by all primitive data types.

```
package com.paolabs.lab3.exemplul1;

import java.util.Arrays;

public class PracticeArrays {

   public static void main(String[] args) {

       long[] array1 = {12L, 56L, 9999L};
       long[] array2 = {1L, 2L, 3L};

       boolean equals = Arrays.equals(array1, array2);
       System.out.println("The arrays are equal? " + equals);

       boolean equals1 = Arrays.equals(array1, array1);
       System.out.println("The arrays are equal? " + equals1);

   }
}
```

- **binarySearch(Object[] a, Object key)** - Searches the specified array of Object (Byte, Int , double, etc.) for the specified value using the binary search algorithm. The array must be sorted prior to making this call. This returns index of the search key, if it is contained in the list; otherwise, it returns (– (insertion point + 1))

```
package com.paolabs.lab3.exemplul1;

import java.util.Arrays;

public class PracticeArrays {
```

```
    public static void main(String[] args) {

        float[] values = new float[3];

        values[0] = 10.0f;
        values[1] = 20.0f;
        values[2] = 15.0f;

        Arrays.sort(values);

        int valSearched1 = Arrays.binarySearch(values, 10.0f);
        int valSearched2 = Arrays.binarySearch(values, 11.0f);

        System.out.println("First value searched: " + valSearched1);
        System.out.println("Second value searched: " + valSearched2);

    }
}
```

- **fill(int[] a, int val)** - Assigns the specified int value to each element of the specified array of ints. This method could be used by all primitive data types.

```
package com.paolabs.lab3.exemplul1;

import java.util.Arrays;

public class PracticeArrays {

    public static void main(String[] args) {

        int[] arraytoBeFilled = new int[5];
        Arrays.fill(arraytoBeFilled, 7);

        for (int i = 0; i < arraytoBeFilled.length; i++) {
            System.out.print(arraytoBeFilled[i]);
        }

    }
}
```

- ➢ Other methods from the Arrays class you can find in the java documentation: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html

---

Object oriented programming - lab 3

Parameter Immutability:

- ➢ When we call a method and we pass parameters into it, those parameter values are actually ***copied*** down into the parameter => this is something we call **passing "by-value"**.

```
package com.paolabs.lab3.exemplul2;

public class PracticeParameterImmutability {

   public static void main(String[] args) {
       int val1 = 10;
       int val2 = 20;

       System.out.println("val1 = "+ val1);
       System.out.println("val2 = "+ val2);

       swap(val1, val2);

       System.out.println("val1 = "+ val1);
       System.out.println("val2 = "+ val2);
   }

   public static void swap(int i, int j) {
       System.out.println("i = "+ i);
       System.out.println("j = "+ j);

       int k = i;
       i = j;
       j = k;

       System.out.println("i = "+ i);
       System.out.println("j = "+ j);
   }
}
```

- ➢ We know objects behave a little differently than primitive types. Remember that **classes** work with **references**:

- Changes made to **passed value** are **not visible outside of method**:

```
package com.paolabs.lab3.exemplul2;

public class PracticeParameterImmutability {

   public static void main(String[] args) {

       Flight flight1 = new Flight(10);
       Flight flight2 = new Flight(15);

       // flight 1 and flight 2 are not the objects themselves but
references to the Flight objects
       System.out.println("flight1 = "+ flight1.getFlightNumber());
       System.out.println("flight2 = "+ flight2.getFlightNumber());

       swap(flight1, flight2);

       System.out.println("flight1 = "+ flight1.getFlightNumber());
       System.out.println("flight2 = "+ flight2.getFlightNumber());
       // changes made to passed value are not visible outside of method
   }

   public static void swap(Flight f1, Flight f2) {
       System.out.println("f1 = "+ f1.getFlightNumber());
       System.out.println("f2 = "+ f2.getFlightNumber());

       Flight k = f1;
       f1 = f2;
       f2 = k;

       System.out.println("f1 = "+ f1.getFlightNumber());
       System.out.println("f2 = "+ f2.getFlightNumber());
   }
}
```

- Changes **made to members of passed class instances** are **visible outside of method**

```
package com.paolabs.lab3.exemplul2;

public class PracticeParameterImmutability {
```

```
   public static void main(String[] args) {
       // Changes made to members of passed class instances are visible
outside of method
       Flight flight3 = new Flight(10);
       Flight flight4 = new Flight(15);

       System.out.println("flight3 = "+ flight3.getFlightNumber());
       System.out.println("flight4 = "+ flight4.getFlightNumber());

       swapNumbers(flight3, flight4);

       System.out.println("flight3 = "+ flight3.getFlightNumber());
       System.out.println("flight4 = "+ flight4.getFlightNumber());
   }

   public static void swapNumbers(Flight f3, Flight f4) {
       System.out.println("f3 = "+ f3.getFlightNumber());
       System.out.println("f4 = "+ f4.getFlightNumber());

       int k = f3.getFlightNumber();
       f3.setFlightNumber(f4.getFlightNumber());
       f4.setFlightNumber(k);

       System.out.println("f3 = "+ f3.getFlightNumber());
       System.out.println("f4 = "+ f4.getFlightNumber());
   }
}
```

Overloading:

- ➢ A class may have multiple versions of its constructor or methods => known as “overloading” (supraincarcare)
- ➢ Each constructor and method must have a unique signature, made up of 3 parts:
  - Number of parameters
  - Type of each parameter
  - Name

```
package com.paolabs.lab3.exemplul3;

public class Passenger {
```

```
private int freeBags;
private int checkedBags;
private char flightClass;

public Passenger() {
}

public Passenger(int freeBags) {
    this.freeBags = freeBags;
}

public Passenger(char flightClass) {
    this.flightClass = flightClass;
}

public Passenger(int freeBags, int checkedBags) {
    this(freeBags);
    this.checkedBags = checkedBags;
}

public int getFreeBags() {
    return freeBags;
}

public void setFreeBags(int freeBags) {
    this.freeBags = freeBags;
}

public int getCheckedBags() {
    return checkedBags;
}

public void setCheckedBags(int checkedBags) {
    this.checkedBags = checkedBags;
}

public char getFlightClass() {
    return flightClass;
}

public void setFlightClass(char flightClass) {
```

```
        this.flightClass = flightClass;
    }
}
```

Class Inheritance:

- ➢ A class can be declared to inherit from another class - using the keyword “extends”
- ➢ The class derived has the characteristics of the base class
    - Can add specialization
    - Can be assigned to base class typed references
    - If the derived class adds a field with the same name as the field in the base class => the derived class hides the field from the base class
    - **Methods override** base class methods with **same signature**

```
package com.paolabs.lab3.exemplul4;

public class Bus extends Car {

    int seats = 20;
    private int doors = 2;

    public Bus() {
    }

    public int getDoors() {
        return doors;
    }

    public void setDoors(int doors) {
        this.doors = doors;
    }
}
```

```
package com.paolabs.lab3.exemplul4;

public class Car {

    int seats = 4;
    private int doors = 4;

    public Car() {
```

```
    }

    public int getDoors() {
        return doors;
    }

    public void setDoors(int doors) {
        this.doors = doors;
    }
}
```

```
package com.paolabs.lab3.exemplul4;

public class TestVehicle {

    public static void main(String[] args) {

        Car c1 = new Car();
        System.out.println(c1.seats);
        System.out.println(c1.getDoors());

        Bus b1 = new Bus();
        System.out.println(b1.seats);
        System.out.println(b1.getDoors());

        Car c2 =  new Bus();
        System.out.println(c2.seats);
        System.out.println(c2.getDoors());

    }
}
```

- ➢ Every class has the characteristics of the Object class
  - Defines a number of methods that are inherited by all objects:
    - **clone** - create a new object instance that duplicates the current instance
    - **hashCode** - get a hash code for the current instance
    - **getClass** - return type information for the current instance
    - **toString** - return string of characters representing the current instance
    - **equals** - compare another object to the current instance for equality
  - Every class inherits directly or indirectly from the Object class

Special Reference: super

- ➢ Similar to *this, super* is an implicit reference to the current object
  - *super* treats the object as if it is an instance of the base class
  - Useful for accessing base class members that have been overridden

```
package com.paolabs.lab3.exemplul5;

public class Person {

   private int age;
   private String name;

   public Person(int age, String name) {
       this.age = age;
       this.name = name;
   }

   public int getAge() {
       return age;
   }

   public void setAge(int age) {
       this.age = age;
   }

   public String getName() {
       return name;
   }

   public void setName(String name) {
       this.name = name;
   }
}
```

```
package com.paolabs.lab3.exemplul5;

public class Student extends Person {

   private int noOfClasses;

   public Student(int age, String name, int noOfClasses) {
```

```
        super(age, name);
        this.noOfClasses = noOfClasses;
    }

    public int getNoOfClasses() {
        return noOfClasses;
    }

    public void setNoOfClasses(int noOfClasses) {
        this.noOfClasses = noOfClasses;
    }
}
```

Final and Abstract

- ➢ All classes can be extended and derived classes have the option to use or override inherited methods => A class can change these defaults by using the **final** keyword => We use **final** to prevent inheriting and/or overriding
- ➢ We use **abstract** to require inheriting and/or overriding