# TND002 Object-oriented programming

## Lecture 5: Useful standard classes and JavaDoc

Mark Eric Dieckmann,
MIT division,
ITN, Linköping University.

January 18, 2023

## Outline of the lecture

- Standard classes (Math, Random, Double, and Integer)
- Static (constant size) arrays
- Generics
- Dynamic arrays (ArrayList)
- JavaDoc documentation

## The wrapper class Integer

Variables of type *int* have a fixed memory requirement and can thus go into the stack memory.

Java provides us with a class **Integer**, which is wrapped around an integer and provides relevant instance and class methods.

Initialization: *Integer exInt = 10;* (*exInt* is a reference variable)

```
String s = "10"; Integer refInt; int primInt;
refInt = Integer.valueOf(s); primInt = Integer.parseInt(s);
```

*valueOf(arg)* converts *arg* into an Integer.

*parseInt(arg)* converts *arg* into an int.

## The wrapper class Double and Math

The class **Double** is a wrapper class for a *double*.

Initialized by autoboxing in the form *Double refDouble = 10.0;*

It comes with the class methods *valueOf(arg)*, which convert *arg* into a Double. *parseDouble(arg)* returns a double.

The class **Math** contains all mathematical functions and several class constants like $\pi$ (*Math.PI*) or *e* (*Math.E*).

It only contains class methods and constants: typing **Math.** lists them all. You use its methods as usual, for example

*double d = Math.sin(Math.PI/2);* gives the result *d=1*.

## Importing the class Random

We will occasionally use a sequence of random numbers.

**Random** provides us with suitable methods.

Random is located in the library *java.util.Random* and has to be imported.

```
package lecture5;
import java.util.Random;
public class Example {
public static void main(String[] args) {
Random r = new Random();
System.out.println(r.nextInt(10)); // random int between 0 and 9
System.out.println(r.nextDouble()); // random double between
0.0 and 1.0. } }
```

## Static arrays for numbers

We have previously used static arrays for Strings (Lesson1b).

```
double[] parray = new double[3]; // reserves space for three doubles in the heap.

Double[] rarray = new Double[3]; // reserves space for three instances of Double in the heap.

rarray[0] = 1.0; rarray[1] = 2.0; rarray[2] = 7.0;

Integer[] oneDarray = {1,2}; // you can initialize at declaration.

int[][] twoDarray = {{1,2},{5,3}};
```

You can not change the array size after you set it. You can reinitialize the array, replacing the old array with a new one.

## An own dynamic array

Some tasks work only if we can change the array size at runtime. We could write a wrapper class for a static array.

```
public class DynamicArray {
private int[] theArray;
public DynamicArray() { theArray = null; }
public int addElement(int arg) {
if (theArray == null) { theArray = new int[1]; theArray[0] = arg; }
else { int[] temp = new int[theArray.length+1];
for (int i=0; i < theArray.length; i++) { temp[i] = theArray[i]; }
temp[theArray.length] = arg; theArray = temp;}
return theArray.length;} }
```

The call *addElement(arg)* adds another slot to *theArray*.

## Dynamic arrays

Better to use the inbuilt one:

Instances of **ArrayList** are dynamic arrays; the number of elements can change at run time.

**ArrayList** is also in the standard library and you must import it.

```
package lecture5;
import java.util.ArrayList;
public class MainClass {
public static void main(String[] args) {
ArrayList<Double> al = new ArrayList<Double>(); } }
```

You initialize it for objects of one class, for example **Double**.

Array lists do not work for primitive variables.

## Generics T

*ArrayList<T>* uses the generic type **T** (the **T** has to be used):

```
public class Generics<T> {
T theVariable;
public Generics(T arg) { theVariable = arg;}
public String toString()
{ return theVariable.getClass().getSimpleName(); } }
```

```
public static void main(String[] args) {
Generics<Double> d = new Generics<Double>(1.0);
System.out.println(d);
Generics<Integer> i = new Generics<Integer>(1);
System.out.println(i); }
```

*getClass()* returns the runtime class and *getSimpleName()* returns the name of the class.

## Adding elements to the ArrayList

*ArrayList<Double> al = new ArrayList<Double>();* starts off with zero elements.

The instance method *al.size()* returns the number of elements.

You add elements with the overloaded method *add(..)*:

```
public static void main(String[] args) {
ArrayList<Double> al = new ArrayList<Double>();
System.out.println(al.size()); // writes 0.
Double d1 = 1.0, d2 = 2.0, d3 = 3.0, d4 = 4.0;
al.add(d1); // adds d1 at the slot 0.
al.add(d2); // adds d2 to the next higher free slot 1.
al.add(d3); // adds d3 to the slot 2.
al.add(1,d4) // adds d4 to slot 1 and shifts d2, d3 by one slot. }
```

## Retrieving elements from the ArrayList

The method *get(int arg)* returns the element at position *arg*.

```
public static void main(String[] args) {
ArrayList<Double> al = new ArrayList<Double>();
// definition and addition of Doubles on previous slide.
System.out.println(al.size()); // writes 4
System.out.println(al.get(0)); // writes 1.0
System.out.println(al.get(1)); // writes 4.0
System.out.println(al.get(2)); // writes 2.0
System.out.println(al.get(3)); // writes 3.0
```

The index 3 is the largest one because numbering starts at 0.

## Replacing and removing elements

We can replace elements of the array list *al* as

```
al.set(2, Double.valueOf(5.0));
```

We have replaced the Double with the index 2 (list position 3) by a new Double.

```
al.remove(4);
```

*remove(arg)* deletes the element with the index *i*.

All elements with indices larger than *i* are shifted one slot down, filling the gap at position *i*.

Dynamic arrays do not have unoccupied slots.

## JavaDoc documentation

One line comments in java start with //

Longer comments are enclosed by /* and */

JavaDoc comments are enclosed by /** and */

You can describe global variables, methods and the class in JavaDoc comments.

Eclipse converts JavaDoc comments into html code.

JavaDoc comments come before the first line of the class, method or variable they describe.

## JavaDoc documentation

You can provide information for a class between the imports and the class declaration.

Java provides you with special fields that follow @.

```
/**
* @author Mark Dieckmann
* @version 1.0
* @since today
*/
```

You can also place a comment before a public variable:

```
/** This class constant corresponds to blah blah.*/
public static final int ACONSTANT = 1;
```

## JavaDoc documentation

We can also comment methods.

```
/**
* Adding two doubles (short description)
* <p> // paragraph symbol
* You can have an extended description after this paragraph.
* @param myArg The method requires this input argument
* @return The method returns twice the input value.*/
public Vector add(double myArg) { return 2*myArg; }
```

You generate JavaDoc by right clicking on the class / package / project name and selecting the Export / Java / Javadoc wizard.

## Summary

We discussed in this lecture

- standard classes (Math, Random, Double and Integer)
- static arrays.
- generics
- dynamic arrays (ArrayList).
- JavaDoc documentation.