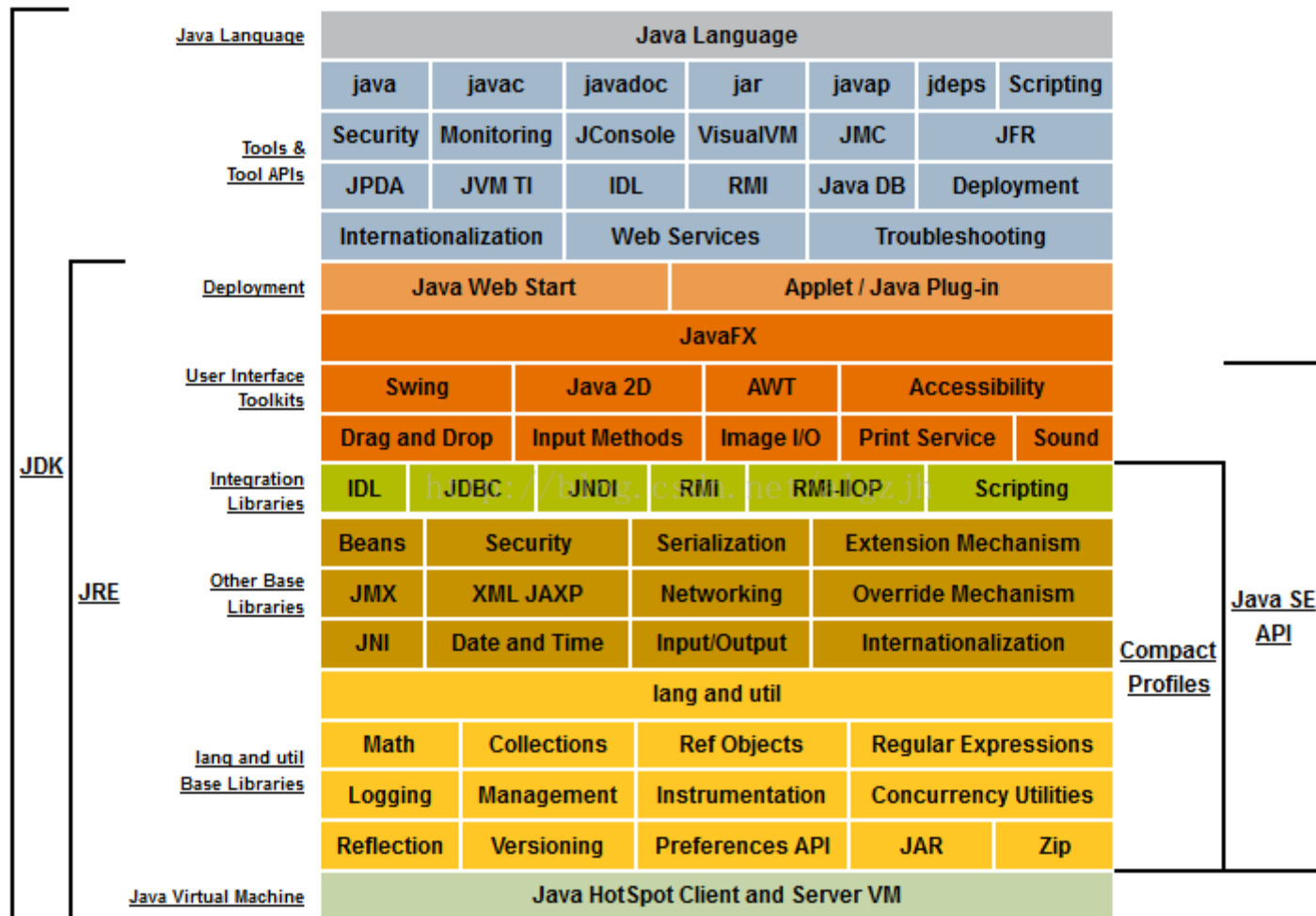







# Specifics of OOP in Java

- All classes are derived from the class Object
- In Java, everything is an object, except for the variables of one of the eight primitive types
- You never have to destroy an object - an automatic garbage collector
- Multiple inheritance prohibited - interfaces are used
- The type can be a class, an interface, or one of eight primitives. These are the only possibilities. Only classes can be used to create new objects
- Java philosophy - "write once, run anywhere" (WORA)

# Conceptual diagram of Java components



# Java Archives - JavaARchive (JAR)

- JAR is a file format typically used to aggregate multiple Java .class files and associated metadata and resources (text, images, etc.) into a single file for distribution
- JAR is based on ZIP format, has a .jar extension
- JAR files allow you to efficiently deliver and run an entire application, including all supporting resources, in one application - applications: web and mobile programming
- Creating .jar from Eclipse: File Export  Java   JAR file   
 selection of preferred classes and resources for inclusion in South Africa
- The contents of the .jar file can be unpacked with any standard decompression tool or using the jar command tool

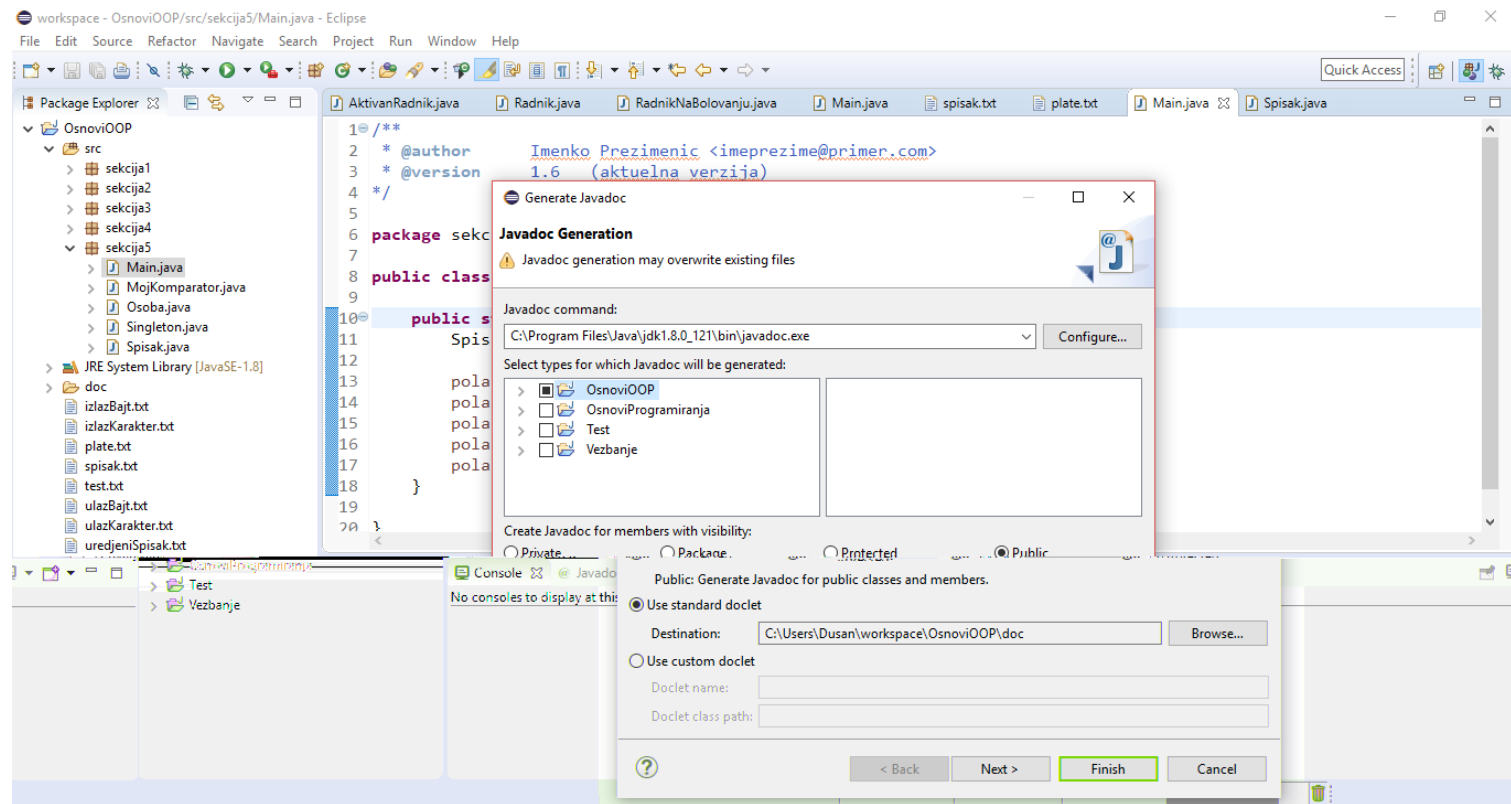
# Javadoc

- Javadoc is a documentation generator used to generate API documentation in HTML format directly from Java source code files
- Shape comments / **\*\* ... \*** /
- Uses and tags - **@author** ,, **@version** ,, **@param.** ..
- De facto industry standard for documentation in Java
- Example of use:

```
/ **  
  * @author          Imenko Last name <first name@example.com >  
  * @version        1.6      (current version)  
* /  
public class Test {  
    // class body  
}
```

# Javadoc

- Code method set `@ param` i `@ return`
- Launch: Project ➡ Generate Javadoc (previously set the path to javadoc.exe)



# Java platform

The Java class library can be divided into two basic groups of packages:

1. The first group consists of standard packages with classes necessary for programming in Java

Examples:

`java.lang`

`java.io`

`java.util`

2. The second group consists of additional packages with classes for creating applets, working with the network, etc.

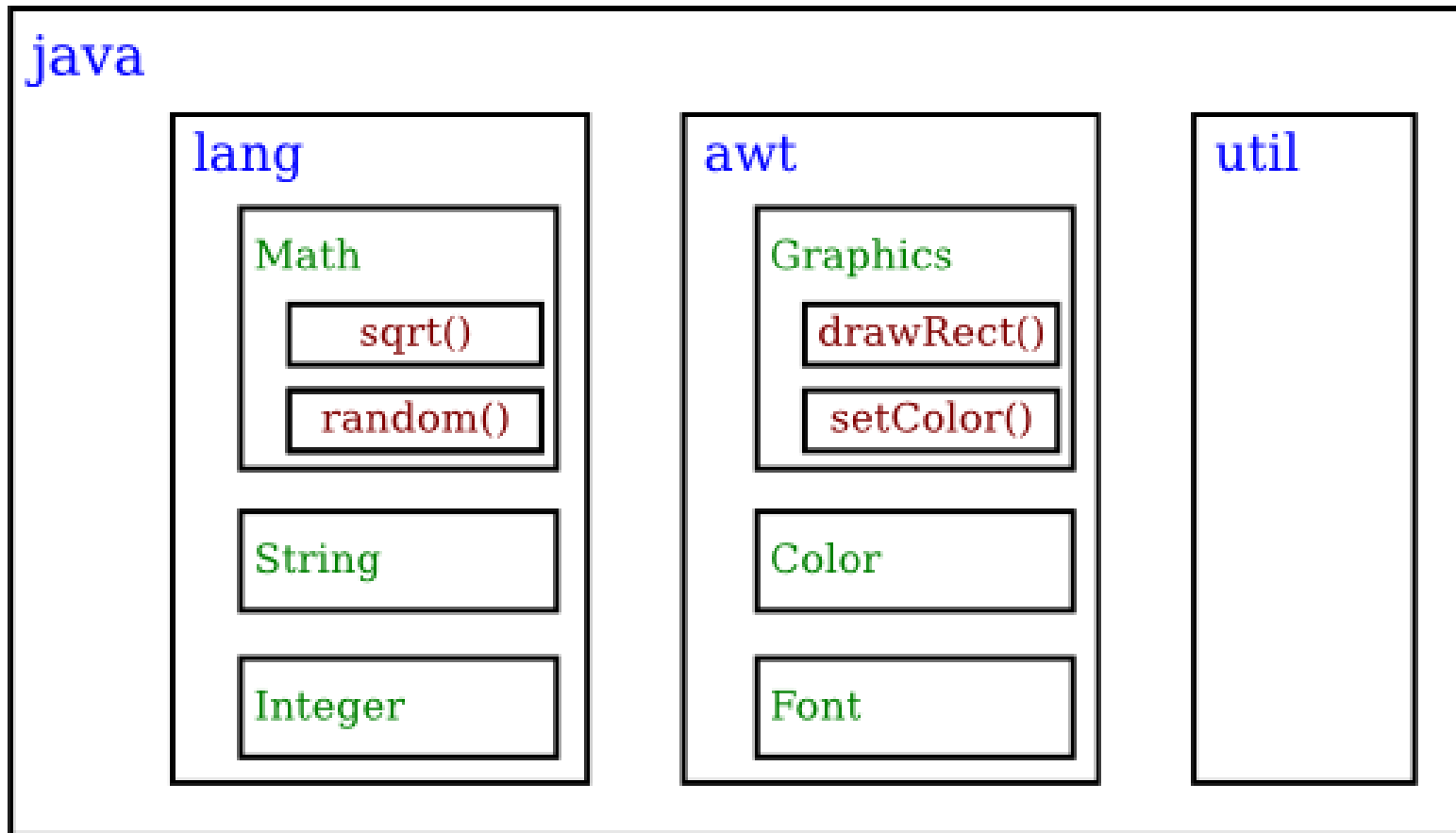
Examples:

`java.applet`

`java.net`

...

# Java platform - standard packages



**Methods** nested in **class** nested in two layers **package** . The full name of the `sqrt()` method is `java.lang.Math.sqrt()` .

# Java platform - java.lang

- Package java.lang contains the basic interfaces and classes necessary for programming in Java. These include class hierarchies, types that are part of the language definition, basic exceptions, mathematical functions, and so on. The classes in this package are **automatically turned on** in each Java source file.
- The most important classes in java.lang are:
  - Object - root class of all classes
  - System - a class that provides system operations
  - Math - a class with basic mathematical functions
  - Throwable, Exception, Error - classes for working with errors and exceptions
  - String - class for working with strings
  - Character, Integer, Float ... - envelope (engl. *wrapper*) classes for primitive types



## Java threads - java.lang

- Java has excellent support for multiprocessing and working with threads, which are very important on modern computers
- Threads are represented by an object that belongs to a class `java.lang.Thread` ( or a subclass of this class) or an object of the class that implements the interface `java.lang.Runnable`
- Purpose of the object `Thread` is to perform a method only once. This method is a task that the thread needs to perform. Multiple threads can be executed in parallel
- Nor can they be programmed by creating a class derived from a class `Thread` or a class that implements an interface `Runnable` and defines the method in it `public void run ()`. The implementation of this method defines the task that the thread will perform

# Java platform - java.io

- Package java.io contains interfaces and classes for working with input and output
- Classes within this package implement work with flows
- The most important classes are:
  - For working with byte streams - abstract classes `InputStream` i `OutputStream`
  - For working with character streams - abstract classes `Reader` i `Writer`
- The class methods of this package generate type exceptions `IOException` in case they cannot be executed - they should be called within try-catch-finally structure
- Package java.io it also contains classes such as `RandomAccessFile` (working with random access files) i `File` ( represents a file or path in the file system)

# Java platform - java.util

- Package java.util contains interfaces and classes with data structures, random number generation, time and date, and other auxiliary tools.
- The most important part of this package is the Collections work environment
  - organized hierarchy of data structures that is designed under the strong influence of project patterns, contains e.g. ArrayList, LinkedList, HashTable, etc.
- This package contains important interfaces Iterator, Comparator, Collection or Map, as well as classes such as Scanner, Vector or Calendar

# JavaAPI documentation

<https://docs.oracle.com/javase/8/docs/api/index.html>

The screenshot shows a web browser displaying the Java Platform, Standard Edition 8 API Specification. The page has a dark blue header with the title "Java™ Platform, Standard Edition 8" and a navigation bar with links: OVERVIEW, PACKAGE, CLASS, USE, TREE, DEPRECATED, INDEX, and HELP. The main content area is titled "Java™ Platform, Standard Edition 8 API Specification" and includes a description: "This document is the API specification for the Java™ Platform, Standard Edition." Below this, there is a section for "Profiles" listing compact1, compact2, and compact3. A "Packages" section is also visible, listing various packages like java.applet, java.awt, java.awt.color, and java.awt.datatransfer. The left sidebar shows a list of "All Classes" and "All Profiles". The bottom of the page features a Windows taskbar with various application icons and a system clock showing 21.10.5.2017.

Java™ Platform, Standard Edition 8

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES

## Java™ Platform, Standard Edition 8 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

### Profiles

- compact1
- compact2
- compact3

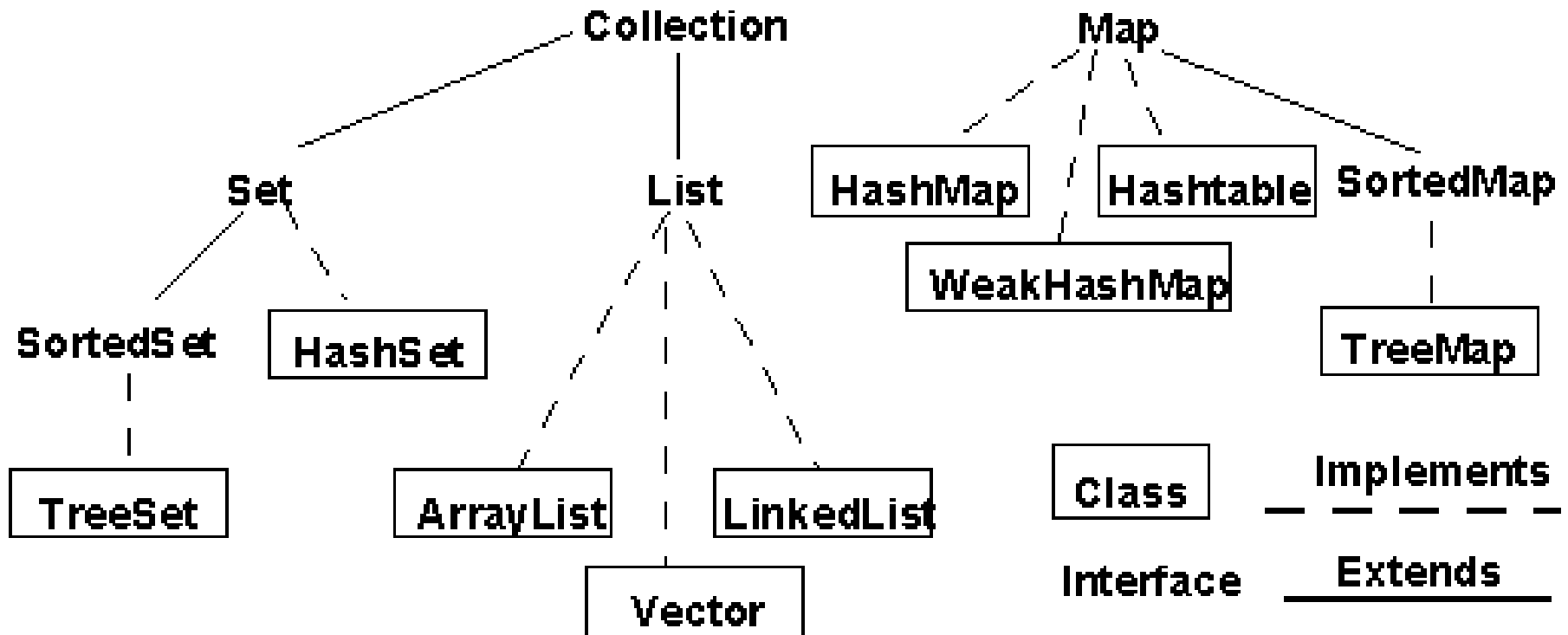
### Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and with various object models.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities located with presentation elements in the GUI.

AbstractAction  
AbstractAnnotationValueVisitor7  
AbstractAnnotationValueVisitor8  
AbstractBorder  
AbstractButton  
AbstractCellEditor  
AbstractChronology  
AbstractCollection  
AbstractColorChooserPanel  
AbstractDocument  
AbstractDocument.AttributeContext  
AbstractDocument.Content  
AbstractDocument.ElementEdit  
AbstractElementVisitor6  
AbstractElementVisitor7  
AbstractElementVisitor8  
AbstractExecutorService  
AbstractInterruptibleChannel  
AbstractLayoutCache  
AbstractLayoutCache.NodeDimensions  
AbstractList

21.10.5.2017

# Java Collections Framework - JCF



# Lists and sets in JCF

- There are two basic types of element collections in Java **list** i  
**expensive**
- **The list consists of a sequence of elements in a linear arrangement.** The list has a specific arrangement, but that does not mean that the values of the elements in it are sorted
- **A set is a collection in which there are no duplicate elements.** Elements of a set may or may not have some arrangement
- The third type of collection is somewhat less frequently used than lists and sets **priority rows** ( engl. priority queues)

## Lists, sets and folders in JCF

- The two standard data structures for presenting lists are **dynamic array** i **chain list**
- Java sets, unlike the mathematical notion of a set, must be finite and contain only elements of the same type
- **Maps are a type of generalized arrays.** They consist of elements in the form of pairs (key, value). The basis for working with maps in Java is the interface Map <K, V>
- Modern systems for working with large data sets, such as Hadoop and Spark, are based on Java and work with maps - MapReduce program model

## JCF lists - ArrayList, LinkedList

- Object type ArrayList <T> represents an ordered sequence of type objects T, placed in a row that can grow as needed - whenever a new element is added
- Object type LinkedList <T> it also represents an ordered sequence of type objects T, but objects are stored in nodes that are interconnected by pointers
- Class LinkedList is more efficient in applications where elements are often added or removed at the beginning or middle of the list, while the class ArrayList more efficient when frequent random access to list items is required
- Both lists implement interface methods Collection, so it is possible to sort them easily ( sort), turning ( reverse), etc.



## JCF sets - TreeSet, HashSet

- Sets implement all interface methods `Collection`, but in such a way as to ensure that no element can appear twice in a set
- Expensive `TreeSet` has the property that its elements are arranged in ascending order
- Expensive `HashSet` stores its elements in a special data structure known as a hash table
- With hash tables, the operations of finding, adding and deleting elements are very efficient (much faster than with `TreeSets`). Elements `HashSet`- and are not kept in any special arrangement

## Task 5.1 - List of participants

- Create a program that reads from an input text file *spisak.txt* accepts data on participants (name, surname, JMBG) and displays them on the screen. Then the list of participants should be sorted by JMBG, displayed again on the screen and finally written to the output file.

*uredjeniSpisak.txt*

- Test the classes in the main program by creating an object with a list of students and calling the appropriate methods

## Task 5.1 - class A person

```
public class Person {  
    private String name ;  
    private String last name ;  
    private String jmbg ;  
  
    Person () {}  
  
    public Person (String name , String last name , String jmbg ) {  
        this . name = name ;  
        this . last name = last name ;  
        this . jmbg = jmbg ;  
    }  
  
    public String getName () {  
        return this . name ;  
    }  
    ...  
}
```

## Task 5.1 - class A person

...

```
public String obtainSurname () {  
    return this . last name ;  
}
```

```
public String getJMBG () {  
    return this . jmbg ;  
}
```

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

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

...

## Task 5.1 - class A person

...

```
public void setJMBG (String jmbg ) {  
    this . jmbg = jmbg ;  
}  
  
@Override public String toString () {  
    return ( " Name: " + this . obtainName () + "Surname:"  
        + this . obtainSurname () + "JMBG:"  
        + this . getJMBG ());  
}  
  
}
```

## Task 5.1 - class MyComparator

```
import java.util. *;
```

```
class MyComparator implements Comparator <Person> {
```

```
    @Override public int compare (Person o1 , Person o2 ) {  
        int i = o1 .additJMBG (). compareTo ( o2 .pribaviJMBG ())  
        if ( i> 0) {  
            return - 1;  
        }  
        else if ( i <0) {  
            return 1;  
        }  
        return 0;  
    }  
}
```

# Task 5.1 - class The list

```
import java.io. *;
import java.util. *;

public class List {
    ArrayList <Person> list of students ;

    public void loadList (String Filename ) {Scanner s = null ;

        ArrayList <Person> list of students = new ArrayList <Person> ();
        try {
            s = new Scanner ( new File ( Filename ));
            to {
                String name = s .next (); String last name = s .next
                ();
                String jmbg = s .next (); A person newStudent = new Person ( name, surname, jmbg

                list of students .add ( newStudent );
            } while ( s .hasNext ());
        } catch ( IOException e ) {System. out . println ( e .getMessage ());

        } ...
    }
```

## Task 5.1 - class The list

```
...  
    finally {  
        if ( s != null ) {  
            s .close ();  
        }  
    }  
    this . list of students = list of students ;  
}  
  
public void sortList () {  
    Collections. sort ( this . list of students ,, new MyComparator ());  
}  
  
public void printList () {  
    System. out . println (Arrays. toString ( this . list of students .toArray ());  
}  
  
...
```



## Task 5.1 - class The list

```

...
public void writeList (String Filename ) {PrintWriter pw = null ;

    try {

        pw = new PrintWriter ( new FileOutputStream ( Filename ));
        for ( A person student : this . list of students )
            pw .println ( student .get the Name () + "" +
                           student .get a LastName () + "" +
                           student .pribaviJMBG ());
    } catch ( IOException e ) {System. out . println ( e .getMessage ());

    }
    finally {
        if ( pw != null ) {
            pw .close ();
        }
    }
}
}
}

```

## Task 5.1 - class Main

```
public class Main {
```

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

```
        The list students = new List ();
```

```
        students .ucitajListu ( "spisak.txt" );
```

```
        students .printList ();
```

```
        students .sortList ();
```

```
        students .printList ();
```

```
        students .upisiListu ( "uredjeniSpisak.txt" );
```

```
    }
```

```
}
```

# Tasks for class work

- Modify the package employed so it includes the class  
The list. Use a convenient structure from the Java Collections Framework to keep the list of workers.
- Modify classes Institution, Classroom, Employees  
(which inherits the class Person) i Computer so they use ready-made data structures from the Java Collections Framework. Under what conditions is it more efficient to use it to store a sequence of objects ArrayList, and under which Linke

# UNIFIEDMODELING LANGUAGE (UML)

---