Lesson 3.1
Introduction to Java

Software Analysis and Design

2nd Year, Computer Science

Universidad Autónoma de Madrid

3.1. Introduction to Java

- **Overview, origin, environment**
- Basic elements of the language
- Exercises

Java: quick overview

- Lenguage compiled to *Bytecode,* and then interpreted
- It is "*strongly typed*" and facilitates **modularization**
- Syntactically similar to C/C++, but for **more reliable programs**
- Without explicit pointers, uses automatic memory *garbage collection*
- **100% portable:** "*write once, run everywhere*"
- Supports **concurrency** (`Thread`) and **exception handling**
- With many libraries: standard (I/O, math, …) and **extension**:
  - Graphical User Interfaces (Swing), distributed objects (rmi), XML documents, cryptography, … and many more
- Also executable by web browsers (via `Applet`)

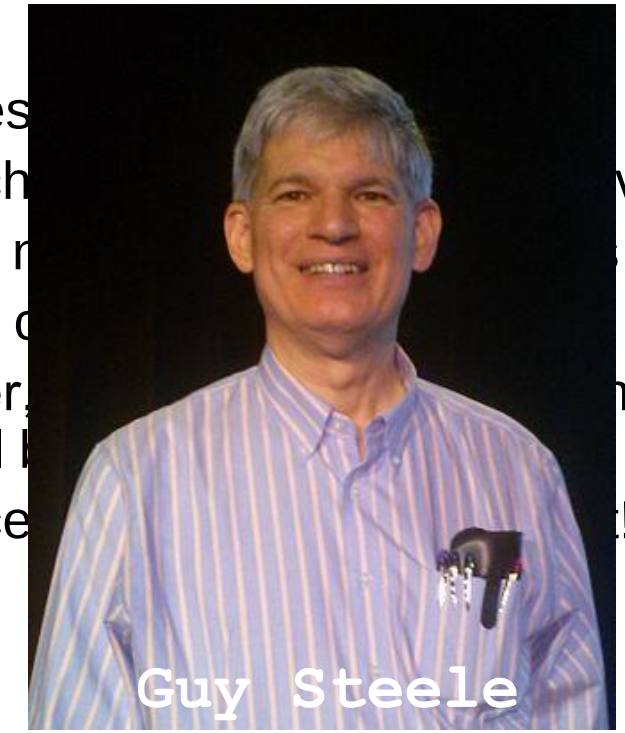
Java: Origin and Editions

- **Java origin** (1991—1995, Sun Microsystems, Inc.)
  - James Gosling
  - Goal: programming home appliances and small equipements
  - Created a new language, Oak, which was then renamed to Java
  - This new *embedded technology* did not succeed as quickly as expected
  - However, the Web and its browsers captured most interest by then
  - Creating a more “intelligent” browser, ***HotJava*** programmed in Java and first to support Applets, was the real booming
  - Now, desk telephones at UAM (since ~2010) use Java. At last!
  - Android apps are written in Java

- **Java Editions**
  - Java SE (Standard Edition), EE (Entreprise Edition), ME (Micro Edition).
  - Distribuited by Oracle, which bought and integrated Sun Microsystems

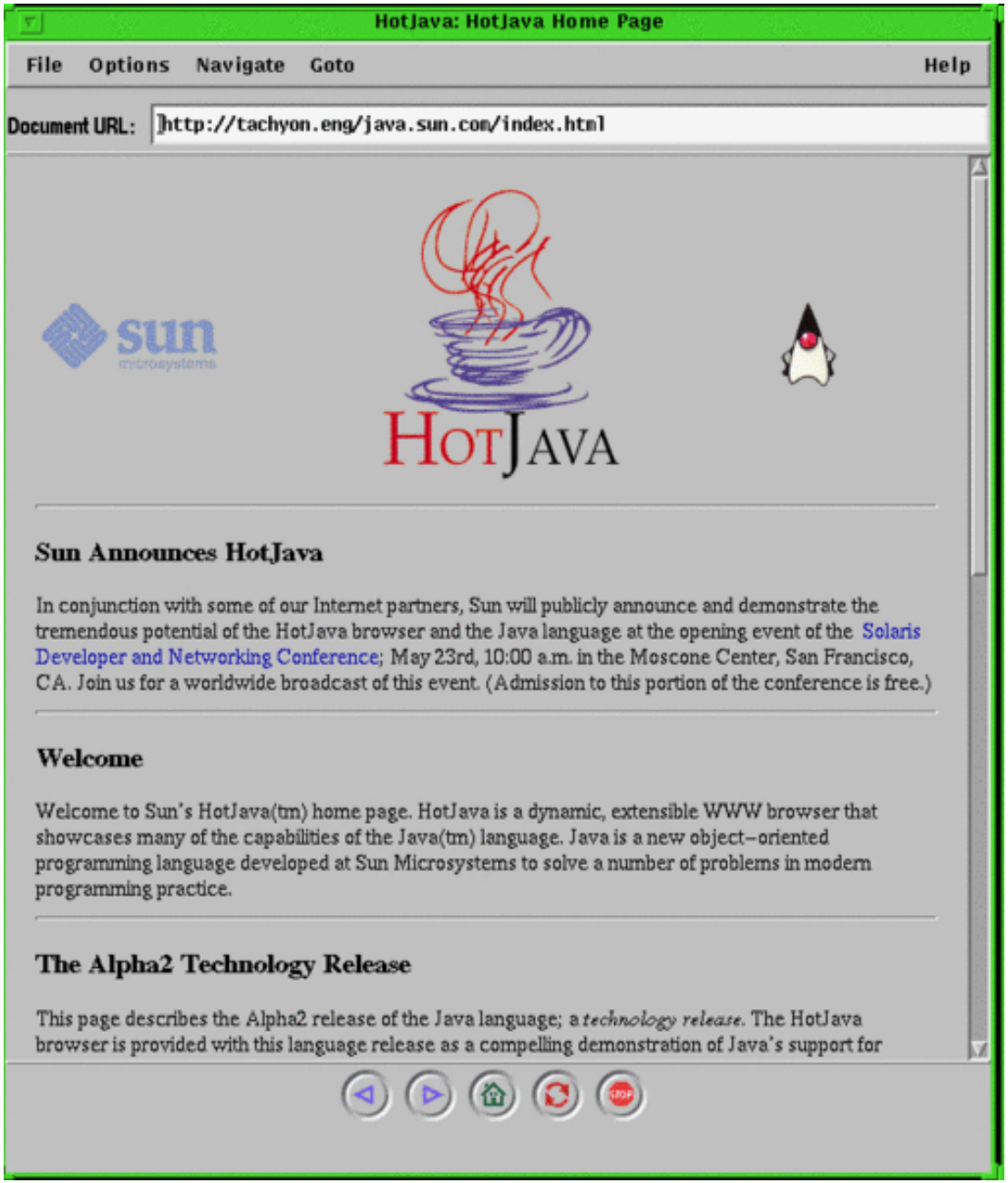
Java: Origin and Editions

- **Java origin** (1991—1995, Sun Microsystems, Inc.)
  - Ja
  - G ances
  - C which va
  - T did r expected
  - H sers then
  - C wser, Java and fi real
  - N since !
  - A a



- **Java Editions**
  - Java SE (Standard Edition), EE (Entreprise Edition), ME (Micro Edition).
  - Distribuited by Oracle, which bought and integrated Sun Microsystems

HotJava


HotJava: HotJava Home Page
File Options Navigate Goto Help
Document URL: http://tachyon.eng/java.sun.com/index.html
sun microsystems
HOTJAVA
Sun Announces HotJava
In conjunction with some of our Internet partners, Sun will publicly announce and demonstrate the tremendous potential of the HotJava browser and the Java language at the opening event of the Solaris Developer and Networking Conference; May 23rd, 10:00 a.m. in the Moscone Center, San Francisco, CA. Join us for a worldwide broadcast of this event. (Admission to this portion of the conference is free.)
Welcome
Welcome to Sun's HotJava(tm) home page. HotJava is a dynamic, extensible WWW browser that showcases many of the capabilities of the Java(tm) language. Java is a new object-oriented programming language developed at Sun Microsystems to solve a number of problems in modern programming practice.
The Alpha2 Technology Release
This page describes the Alpha2 release of the Java language; a technology release. The HotJava browser is provided with this language release as a compelling demonstration of Java's support for

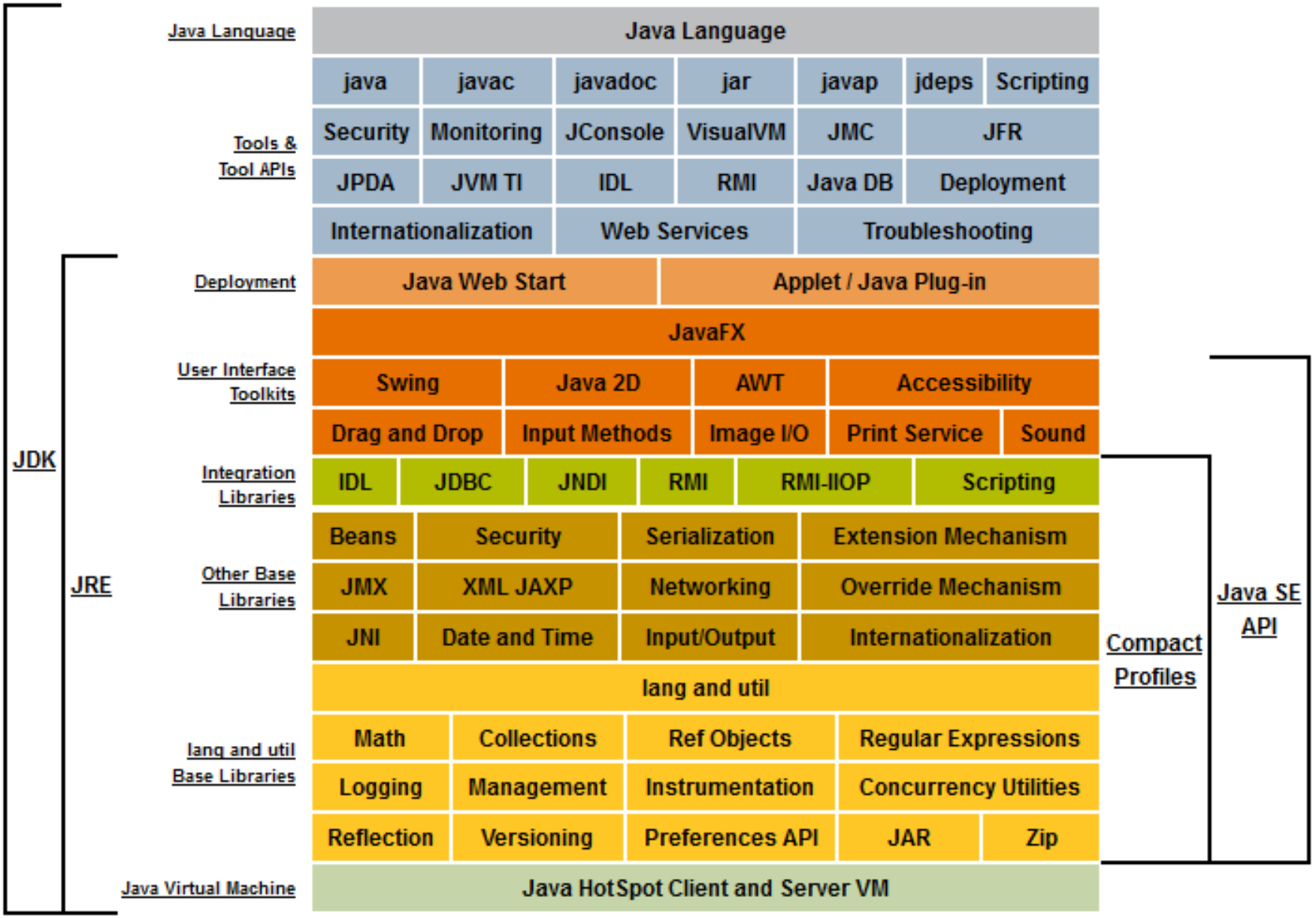
Java Versions

- Java 1.0 – January 1996 — A few hundreds of classes
- Java 1.1 – February 1997 — JavaBeans, JDBC, RMI…
- J2SE 1.2 – december 1998 — Swing, Java IDL, Collections…
- That versión became **"Java 2"**
- J2SE 1.3 – May 2000 — Compatible RMI/CORBA…
- J2SE 1.4 – February 2002 — XML, JWS, internally improved
- J2SE 5.0 (Java 1.5) – September 2004
  - Generics, enumeration, enhanced "for"…
- Java SE 6 (Java 1.6) – December 2006
  - over 3.000 classes, strongly improved performance…
- Java SE 7 – 2011/12
  - better handling of exceptions, allows String in switch statement, …
- Java SE 8 – March 2014
  - Lambda expressions (closures), streams, improved interfaces …
- Java SE 9 – Sept. 2017 — modules, JShell
- Java SE 10 – March 2018 — Local-variable type inference
- Java SE 11 – Sept. 2018
- Java SE 12 – March 2019
- Java SE 13 – Sept. 2019 — Improved switch, text blocks

Java Development Kit (JDK)

- Java Runtime Environment (JRE)
  - Java Virtual Machine (JVM)
  - Java API (*Application Programming Interface*):
    - Java language support + Java standard libraries
    - **http://java.sun.com/docs/books/jls/**
    - **http://docs.oracle.com/javase/9/docs/api/**
- Compile into *Bytecode* (*javac Example.java*) and run it (*java Example*)
- Other general tools: javadoc, appletviewer, jshell, …
- As well as specific to the IDE (*Interactive Development Environment*): templates, syntax-oriented editors, debuggers, …

Java SE 8 Platform



Example of code organization

File PointTest.java

```
class Point{
   private int x;
   private int y;
   public Point (int x, int y) { this.x = x; this.y = y; }
   public void shift(int dx, int dy) { this.x+=dx; this.y+=dy; }
   public String toString() {
     return "(" + x + "," + y + ")";
   }
}

public class PointTest {
  public static void main(String[] args) {
     Point p = new Point(1,-2);
     System.out.println(p);
     p.shift(10,100);
     System.out.println(p);
  }
}
```

- Class definitions
- Identifier conventions
- The main methods
- Object declaration and creation

- Access to object members (structure)
- Class String, concatenation
- Packages
- Scope of variables
- Instance variables, instance methods
- Class variables and class methods

Example: improved organization of code

File Point.java

```
public class Point {
   private int x;
   private int y;
   public Point (int x, int y) { this.x = x; this.y = y; }
   public void shift(int dx, int dy) { this.x+=dx; this.y+=dy; }
   public String toString() {
     return "(" + x + "," + y + ")";
   }
}
```

File PointTest.java

```
public class PointTest {
  public static void main(String[] args) {
     Point p = new Point(1,-2);
     System.out.println(p);
     p.shift(10,100);
     System.out.println(p);
  }
}
```

Even better organized code

File geometry/Point.java

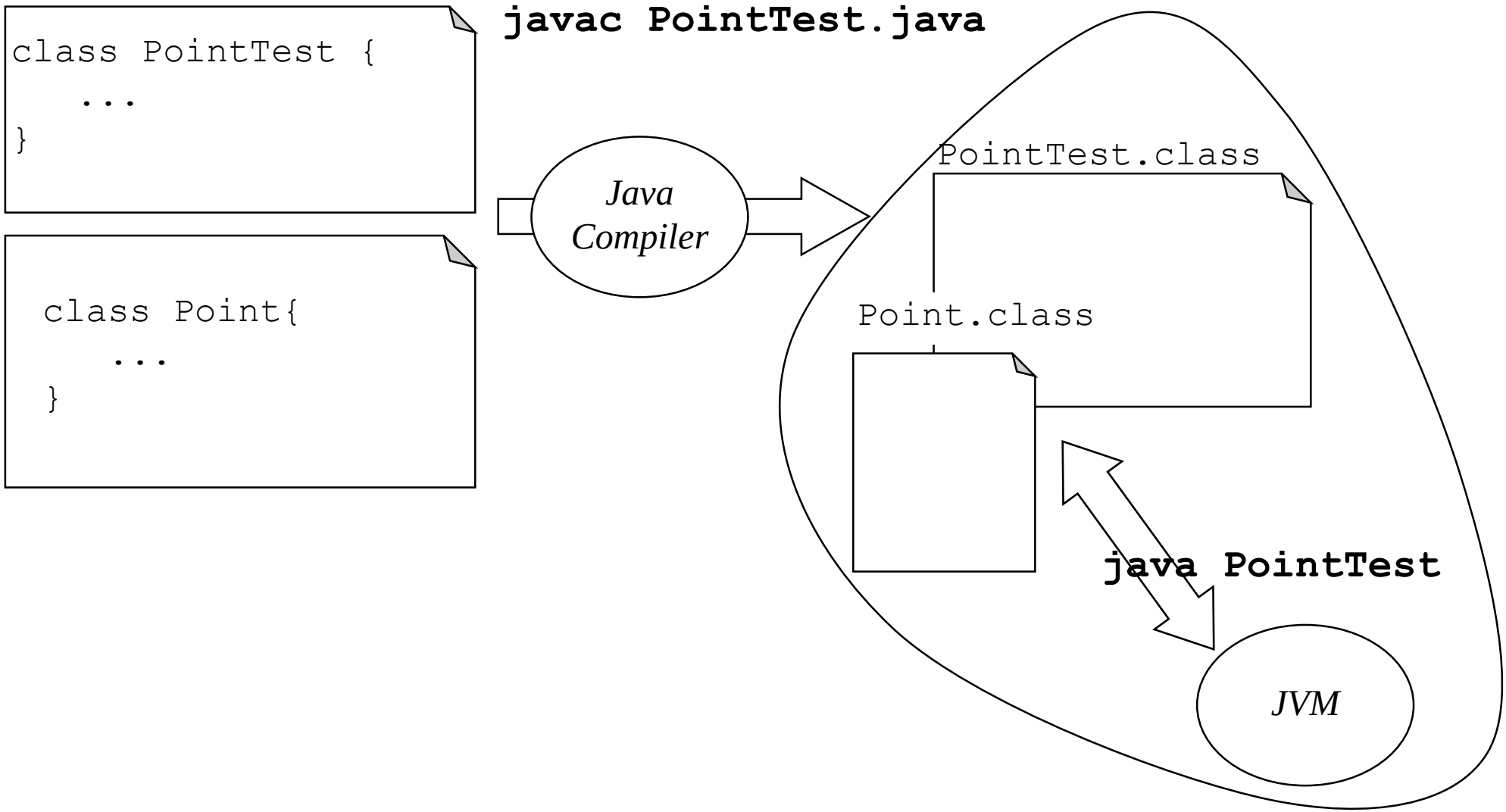
```
package geometry;

public class Point {
   private int y;
   private int y;
   public Point(int x, int y) { this.x = x; this.y = y; }
   public void shift(int dx, int dy) { this.x+=dx; this.y+=dy; }
   public String toString() {
     return "(" + x + "," + y + ")";
   }
}
```

File geometry/test/PointTest.java

```
package geometry.test;
import geometry.Point;
public class PointTest {
  public static void main(String[] args) {
     Point p = new Point(1,-2);
     System.out.println(p);
     p.shift(10,100);
     System.out.println(p);
  }
}
```

Compilation and Execution

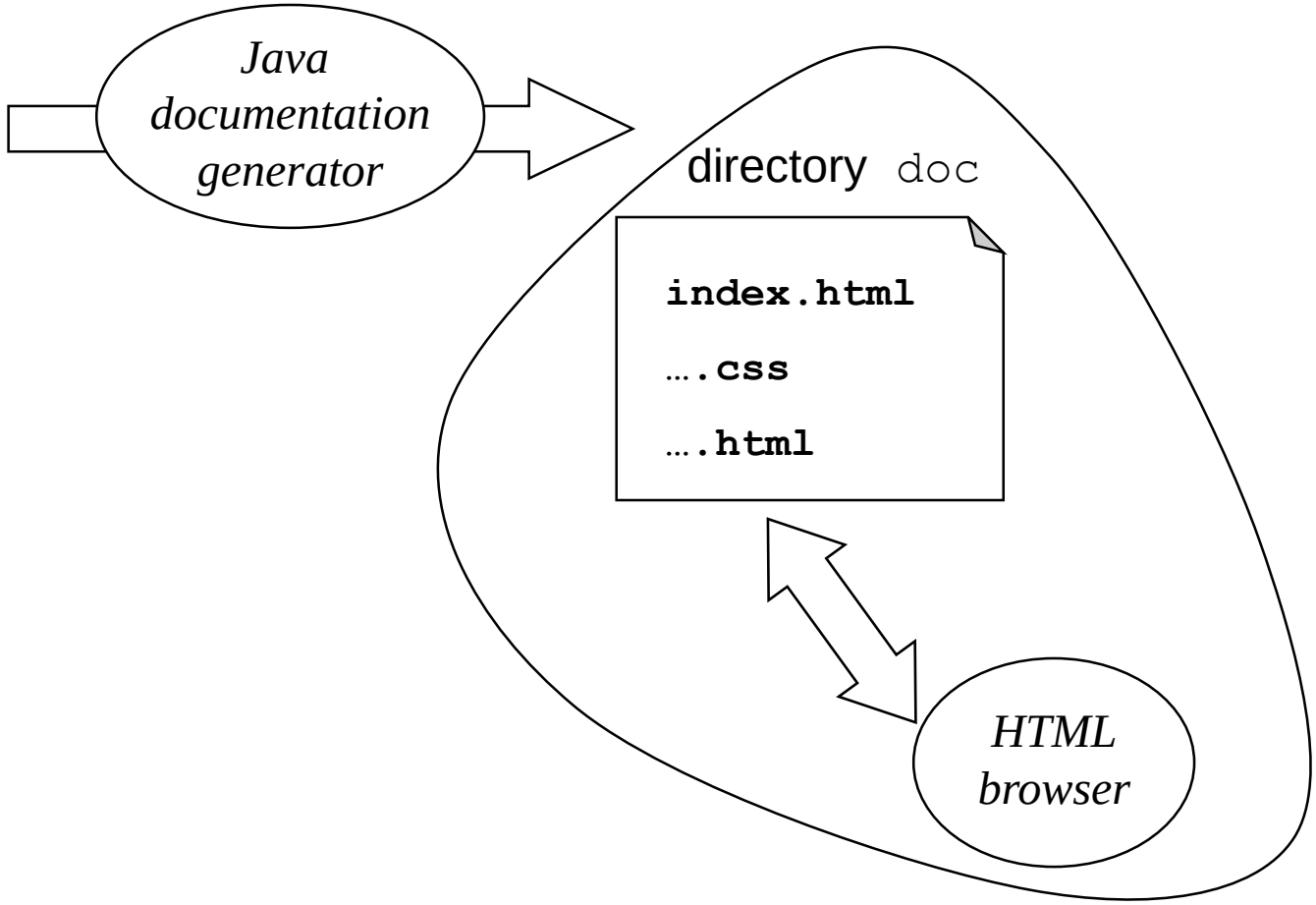


Generating Documentation

```
class PointTest {
   ...
}
```

```
class Point {
   ...
}
```

`javadoc –d doc PointTest.java Point.java`



Java vs. C

Java is object oriented	**C** it is not object oriented
Interpreted (bytecode)	Compiled
Absolutely portable	Not portable features (`sizeof(int)`)
Dynamic memory managed by *garbage collection* Explicit pointers not available	Dynamic memory managed by programmers Pointers: error-prone programming
Abstract data type with full protection	Abstract data types simulated withn **`struct`** and separation in **`*.h *.c`**
Modularity mechanisms	#include is just "*copy and paste*"
Modern programming concepts: exceptions, generics…	It is almost like comparing assembler with C

3.1. Introduction to Java

- Presentation, origin, environment
- **Basic elements of the language**
  - **Primitive data types**
  - **Non primitive (reference) data types**
  - **Control structures**
  - **Input/Output**
  - **Applications executable on Web browers**
- Exercises

3.1. Introduction to Java

- Presentation, origin, environment
- **Basic elements of the language**
  - **(see appendices)**
- **Introduction through examples**
- Exercises

Elementary examples

- Syntactic elements
- Style guide (see Moodle course, Lab Section)
- Simple comments
- Javadoc (advanced) comments
- The main program structure (application)

- Examples shown here do not include Javadoc comments just to facilitate the projection on the screen

Hello World example

```
package examples; /* file examples/HelloWorld.java */

public class HelloWorld {

  public static void main(String[] args)  {
    System.out.println("Hello world, hello class!!");
  }

}
//Style guide:
// HelloWorld, String and System are classes
// out is an object
// println is a method (operation over an object)
// main is also a method (the entry point method)
```

Happy 2020 example

```
package examples; // file examples/Happy2020.java

public class Happy2020 {

  public static void main(String... args) {
    System.out.println("Happy 2020!");
  }

}

// String[] is an array of string of characters
// String... means a variable number of parameters (all strings)
```

Happy New Year example: importing classes

```
package examples;
import java.time.LocalDate;
public class HappyNewYear {

  public static void main(String... args) {
    System.out.println("Happy "
                       + LocalDate.now().getYear() );
  }

}

// before writing your own code, check first if a library exists
// in Java reusable elements are grouped in packages
// whose public components can be imported
```

Happy New Year + "!" example

```
package examples;
import java.time.LocalDate;
public class HappyNewYear2 {

  public static void main(String... args) {
    System.out.println("Happy "
                       + LocalDate.now().getYear()
                       + " to everyone!" );
  }

}
// the operator + is overloaded, it has more than one meaning:
// here it is used to concatenate Strings
// but it can also add numbers, e.g.: 2 + 3, x + 0.5, …
```

Examples: loops, lists, …

- Local variables: initialization, assignment statement
- Arithmetic expressions
- Control structures: `for` without index, `if`, `while`, …
- Object creation (objects can be inmutable or not)
- Using predefined Java packages (libraries)
- Basic list processing: package `java.util.*`

  abstraction from implementation (`ArrayList`, `LinkedList`)

  traversal using for

  access its elements by index

  searching

  processing with a stream (Java 8, details later in Section 3)

Select and add numbers that are even

```
package examples;
import java.util.List;
public class AddEven {
  public static void main(String... args) {
      List<Integer> numbers = List.of(5,2,4,7,8);

      System.out.println( numbers );





  }
}
```

Select and add numbers that are even

```
package examples;
import java.util.List;
public class AddEven {
  public static void main(String... args) {
      List<Integer> numbers = List.of(5,2,4,7,8);

      System.out.println( numbers );

     int totalEven = 0;

                                  totalEven = totalEven + num;

      System.out.println( "Sum of even numbers is: " + totalEven);
  }
}
```

Select and add numbers that are even

```
package examples;
import java.util.List;
public class AddEven {
  public static void main(String... args) {
      List<Integer> numbers = List.of(5,2,4,7,8); // Java 9

      System.out.println( numbers );

      int totalEven = 0;
      for (Integer num : numbers) {
              if (num % 2 == 0) totalEven = totalEven + num;
      }
      System.out.println( "Sum of even numbers is: " + totalEven);
  }
}
```

Select and add numbers that are even

```
package examples;
import java.util.*;
public class AddEvenJava8 {
  public static void main(String... args) {
      // Version Java 8   equivalent to  List.of(5,2,4,7,8);
      List<Integer> nums = new ArrayList<>(Arrays.asList(5,2,4,7,8));
      System.out.println( nums );

     int totalEven = 0;
      for (Integer num : nums) {
              if (num % 2 == 0) totalEven = totalEven + num;
      }
      System.out.println( "Sum of even numbers is: " + totalEven);
  }
}
```

Count numbers that are even

```
package examples;
import java.util.List;
public class CountingEvenWithStream {
  public static void main(String... args) {
      List<Integer> numbers = List.of(5,2,4,7,8);

      System.out.println( "The count of even numbers is: "
            + numbers.stream().filter(n -> n % 2 == 0).count());

  }
}

// this example uses the functional programming style
// it uses the concepts stream and lambda expression new from Java 8
// we will see this in more detail near the end of Section 3
```

Select (*filter*) and add numbers that are even

```
package examples;
import java.util.List;
public class AddEvenUsingStream {
  public static void main(String... args) {
      List<Integer> numbers = List.of(5,2,4,7,8);

      System.out.println( "Sum of even numbers is : "
            + numbers.stream().filter(n -> n % 2 == 0)
                              .reduce(0, Integer::sum));

  }
}

// this example uses the functional programming style
// it uses the concepts stream and expresión lambda new in Java 8
// we will see this in more detail near the end of Section 3
```

Basic example: accessing list elements

```
package examples;
import java.util.List;
public class ListAccess {
  public static void main(String... args) {
      List<Integer> numbers = List.of(5,2,4,7,8);

      System.out.println( numbers );

      System.out.println( numbers.size()   + " numbers" );
      System.out.println( "First number is: " + numbers.get(0) );
      System.out.println( "Last number is: "
                              + numbers.get( numbers.size() - 1) );
  }
} // the List numbers is inmutable: we cannot add/remove elements
```

Concrete implementations of lists

```
package examples;
import java.util.*; //import java.util.List; import java.util.ArrayList;
public class PalsList1 {
  public static void main(String... args) {
      List<String> pals = new ArrayList<String>();

      pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

      System.out.println( pals );
      System.out.println( pals.size()   + " pals" );
      System.out.println( "First pal is: " + pals.get(0) );
      System.out.println( "Last pal is: "
                              + pals.get( pals.size() - 1) );

  }
}  // new creates a new object (instance) from the given class
```

Concrete implementations of lists

```
package examples;
import java.util.*; //import java.util.List; import java.util.ArrayList;
public class PalsList1 {
  public static void main(String... args) {
      List<String> pals = new ArrayList<String>();

      pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

      System.out.println( pals ); // output: [Luca,Leo,Joe]
      System.out.println( pals.size()   + " pals" ); // output: 3
      System.out.println( "First pal is: " + pals.get(0) );
      System.out.println( "Last pal is: "
                              + pals.get( pals.size() - 1) );

  }
}  // new creates a new object (instance) from the given class
```

Another example: list traversal

```
package examples;
import java.util.*;
public class PalsList2 {
  public static void main(String... args) {
      List<String> pals = new ArrayList<>();

      pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

      System.out.println( pals );

      for (String pal : pals) {
        System.out.println( "Hello " + pal + "!" );
      }
  }
} // the List pals is NOT inmutable, it is mutable
```

Traversing another implementation of List

```
package examples;
import java.util.*;
public class PalsList3 {
  public static void main(String... args) {
      List<String> pals = new LinkedList<>();

      pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

      System.out.println( pals );

      for (String pal : pals) {
        System.out.println( "Hello " + pal + "!" );
      }
  }
} // IMPORTANT: our code to traverse a List
  //            does not depend on its implementation details
```

Basic example: searching in lists

```
package examples;
import java.util.*;
public class PalsList4 {
  public static void main(String[] args) {
      List<String> pals = new ArrayList<>();

      pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

      System.out.println( pals );

      if ( pals.contains("Leo") ) {
        System.out.println( "Hello Leo!" );
      }
  }
}
```

Basic example: using command line args in `main`

```
package examples;
import java.util.*;
public class PalsList5 {
  public static void main(String[] args) {
      List<String> pals = new ArrayList<>();

      pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

      System.out.println( pals );

      if ( pals.contains( args[0] ) ) {  // see any error?
        System.out.println( "Hello " + args[0] + "!" );
      }
  }
}
```

Basic example: `if` with and `else` clause

```
package examples;
import java.util.*;
public class PalsList6 {
  public static void main(String[] args) {
      List<String> pals = new ArrayList<>();

      pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

      System.out.println( pals );

      if ( pals.contains( args[0] ) ) {  // see any error?
        System.out.println( "Hello " + args[0] + "!" );
      } else {
        System.out.println( "Hello unknown!" );
      }
  }
}
```

Basic example: handling errors

```
package examples;
import java.util.*;
public class PalsList7 {
  public static void main(String[] args) {
      List<String> pals = new ArrayList<>();

      pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

      System.out.println( pals );

      if ( args.length >= 1 && pals.contains( args[0] ) ) {
        System.out.println( "Hello " + args[0] + "!" );
      }
  }
} // to avoid the uncontrolled (unhandled) exception:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 11
```

Recall: abstraction from the implementation

```
package examples;
import java.util.List;
public class PalsList7Bis {
  public static void main(String[] args) {
    List<String> pals = List.of("Luca","Leo","Joe");




    System.out.println( pals );

    if ( args.length >= 1  &&  pals.contains( args[0] ) ) {
      System.out.println( "Hello " + args[0] + "!" );
    }
  }
}
```

Examples with collections

- More examples using the package `java.util.*`
- Basic handling of collections:

  Lists: `addAll` (advanced operations, bulk operations)

  Sets, `Set`, with various implementations

  `HashSet`, `TreeSet`, ...

  Maps (hash tables), `Map`, several implementations

  `HashMap, TreeMap, ...`
- Create collections from other ones
- Inmutable collections
- Sorting collections

Appending lists

```
package examples;
import java.util.*;
public class PalsList8 {
  public static void main(String... args) {
    List<String> pals = List.of("Luca","Leo","Joe");
    List<String> yourPals = new LinkedList<>();

    yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Mark");

    yourPals.addAll( pals );

    System.out.println( yourPals );


  }
}
```

Appending lists

```
package examples;
import java.util.*;
public class PalsList8 {
  public static void main(String... args) {
    List<String> pals = List.of("Luca","Leo","Joe");
    List<String> yourPals = new LinkedList<>();

    yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Mark");

    yourPals.addAll( pals );

    System.out.println( yourPals );//output:[Pi,Ed,Mark,Luca,Leo,Joe]


  }
}
```

Appending lists

```
package examples;
import java.util.*;
public class PalsList8 {
  public static void main(String... args) {
    List<String> pals = List.of("Luca","Leo","Joe");
    List<String> yourPals = new LinkedList<>();

    yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Mark");

    yourPals.addAll( pals );

    System.out.println( yourPals );//output:[Pi,Ed,Mark,Luca,Leo,Joe]

    // Error when executing: pals.addAll( yourPals );
  }
} // the List pals is inmutable: we cannot add/remove elements
```

Appending lists

```
package examples;
import java.util.*;
public class PalsList8Java8 {
  public static void main(String... args) {
List<String> pals = new ArrayList<>(Arrays.asList("Luca","Leo","Joe"));
     List<String> yourPals = new LinkedList<>();

     yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Mark");

     yourPals.addAll( pals );

     System.out.println( yourPals );//output:[Pi,Ed,Mark,Luca,Leo,Joe]

     pals.addAll( yourPals );
  }
} // The List pals IS mutable
```

Appending lists does not remove duplicates

```
package examples;
import java.util.*;
public class PalsList9 {
  public static void main(String... args) {
    List<String> pals = new ArrayList<String>();
    List<String> yourPals = new LinkedList<String>();

    pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

    yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Mark");

    pals.addAll( yourPals );
    yourPals.addAll( pals );   // see any error? It depends …
    System.out.println( pals );
    System.out.println( yourPals );
  }
}
```

Appending lists does not remove duplicates

```
package examples;
import java.util.*;
public class PalsList9 {
  public static void main(String... args) {
    List<String> pals = new ArrayList<String>();
    List<String> yourPals = new LinkedList<String>();

    pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

    yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Mark");

    pals.addAll( yourPals );
    yourPals.addAll( pals );   // see any error? It depends …
    System.out.println( pals );//output:[Luca,Leo,Joe,Pi,Ed,Mark]
    System.out.println( yourPals ); //output:
  }                                 [Pi,Ed,Mark,Luca,Leo,Joe,Pi,Ed,Mark]
}
```

To avoid duplicates we use sets

```
package examples;
import java.util.*;
public class PalsSet1 {
  public static void main(String... args) {
    Set<String> pals = new TreeSet<>();
    Set<String> yourPals = new TreeSet<>();

    pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

    yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Leo");

    pals.addAll( yourPals );
    yourPals.addAll( pals );
    System.out.println( pals );
    System.out.println( yourPals );
  }
}
```

To avoid duplicates we use sets

```
package examples;
import java.util.*;
public class PalsSet1 {
  public static void main(String... args) {
    Set<String> pals = new TreeSet<>();
    Set<String> yourPals = new TreeSet<>();

    pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

    yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Leo");

    pals.addAll( yourPals );
    yourPals.addAll( pals );
    System.out.println( pals );   // [Ed, Joe, Leo, Luca, Pi]
    System.out.println( yourPals );// [Ed, Joe, Leo, Luca, Pi]
  }
}
```

And now with another implementation of sets

```
package examples;
import java.util.*;
public class PalsSet2 {
  public static void main(String... args) {
    Set<String> pals = new HashSet<>();
    Set<String> yourPals = new HashSet<>();

    pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

    yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Leo");

    yourPals.addAll( pals );
    pals.addAll( yourPals );
    System.out.println( pals );
    System.out.println( yourPals );
  }
}
```

And now with another implementation of sets

```
package examples;
import java.util.*;
public class PalsSet2 {
  public static void main(String... args) {
    Set<String> pals = new HashSet<>();
    Set<String> yourPals = new HashSet<>();

    pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

    yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Leo");

    yourPals.addAll( pals );
    pals.addAll( yourPals );
    System.out.println( pals );   // [Leo, Luca, Joe, Pi, Ed]
    System.out.println( yourPals );// [Luca, Leo, Joe, Pi, Ed]
  }
}//the order is irrelevant, except when using a SortedSet as TreeSet
```

Set equality does not mind the order

```
package examples;
import java.util.*;
public class PalsSet3 {
  public static void main(String... args) {
      Set<String> pals = new HashSet<>();
      Set<String> yourPals = new TreeSet<>();

      pals.add("Luca");  pals.add("Leo");  pals.add("Joe");

      yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Leo");

      yourPals.addAll( pals );
      pals.addAll( yourPals );

      System.out.println( yourPals.equals( pals ) ); // true
  }
}
```

There are also inmutable sets

```
package examples;
import java.util.*;
public class PalsSet4 {
  public static void main(String... args) {
      Set<String> pals = Set.of("Luca","Leo","Joe");
      Set<String> yourPals = new TreeSet<>();

      yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Leo");

      // pals.addAll( yourPals ); // error: pals is immutable
      yourPals.addAll( pals );

      System.out.println( yourPals);
  }
}
```

There are also inmutable sets

```
package examples;
import java.util.*;
public class PalsSet4Java8 {
  public static void main(String... args) {
   Set<String> pals = new HashSet<>(Arrays.asList("Luca","Leo","Joe"));
     Set<String> yourPals = new TreeSet<>();

     yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Leo");

     pals.addAll( yourPals ); // pals IS mutable
     yourPals.addAll( pals );

     System.out.println( yourPals);
  }
}
```

Exercise

- Create a class SortInput that:
  - receives 1 or more numbers from the command line
  - Prints them in ascending order, without repetitions
  - Prints the smallest
  - Prints the biggest

Maps are more powerful than lists and sets

```
package examples;
import java.util.Map;
public class PalsAges1 {
  public static void main(String... args) {
      Map<String, Integer> ages =
                                  Map.of("Luca",23,"Leo",28,"Joe",25);



      System.out.println(ages); // {Luca=23, Leo=28, Joe=25}

      System.out.println("Joe’s age is: " + ages.get("Joe") );

      System.out.println("Ann’s age is: " + ages.get("Ann") );

  }
}  // The map ages is immutable
```

Maps are more powerful than lists and sets

```
package examples;
import java.util.Map;
public class PalsAges1 {
  public static void main(String... args) {
      Map<String, Integer> ages =
                                  Map.of("Luca",23,"Leo",28,"Joe",25);




      System.out.println(ages); // {Leo=28, Joe=25, Luca=23}

     System.out.println("Joe's age is: " + ages.get("Joe") );
                                        //  Joe´s age is: 25
     System.out.println("Ann's age is: " + ages.get("Ann") );
                                        //  Ann's age is: null
  }
}  // The map ages is immutable
```

Maps are more powerful than lists and sets

```
package examples;
import java.util.Map;
public class PalsAges1Java8 {
  public static void main(String... args) {
      Map<String, Integer> ages = new HashMap<>();
      ages.put("Luca", 23);
      ages.put("Leo", 28);
      ages.put("Joe", 25);

      System.out.println(ages); // {Leo=28, Luca=23, Joe=25}

      System.out.println("Joe's age is: " + ages.get("Joe") );
                                        //  Joe´s age is: 25
      System.out.println("Ann's age is: " + ages.get("Ann") );
                                        //  Ann´s age is: null
  }
}  // The map ages is mutable
```

Must check existence, before getting it

```
package examples;
import java.util.Map;
public class PalsAges2 {
  public static void main(String... args) {
      Map<String, Integer> ages =
                                  Map.of("Luca",23,"Leo",28,"Joe",25);




      System.out.println(ages);

      if (! ages.containsKey( "Ann" ))
            System.out.println( "Ann has no age." );
      else System.out.println("Ann's age is: " + ages.get("Ann"));
                                  //  Ann has no age.
  }
}  // The map ages is immutable
```

Must check existance, before getting it

```
package examples;
import java.util.Map;
public class PalsAges2Java8 {
  public static void main(String... args) {
      Map<String, Integer> ages = new HashMap<>();
      ages.put("Luca", 23);
      ages.put("Leo", 28);
      ages.put("Joe", 25);

      System.out.println(ages);

      if (! ages.containsKey( "Ann" ))
            System.out.println( "Ann has no age." );
      else System.out.println("Ann's age is: " + ages.get("Ann"));
                                    //  Ann has no age.
  }
}
```

Updating a value stored in the map

```
package examples;
import java.util.*;
public class PalsAges3 {
  public static void main(String... args) {
      Map<String, Integer> ages = new HashMap<>();
      ages.put("Luca", 23);
      ages.put("Leo", 28);
      ages.put("Joe", 25);

      System.out.println(ages); // {Luca=23, Leo=28, Joe=25}
      // Joe's birthday:
      ages.put( "Joe", ages.get("Joe")+1 );
      System.out.println(ages); // {Luca=23, Leo=28, Joe=26}

  } // but error occurs if  Joe has no age
}
```

Compute frequency of words given as arguments

```
package examples;
import java.util.*;
public class WordFrequencyArgs {
  public static void main(String... args) {
      Map<String, Integer> frequency = new HashMap<>();



      for (String word : args) {
         if (frequency.containsKey(word))
              frequency.put( word, frequency.get(word)+1 );
         else frequency.put( word, 1 );
      }
      System.out.println(frequency);
      // With args: the hour of the truth is the hour of the death
      // output: {of=2, truth=1, the=4, hour=2, death=1, is=1}
  }
}
```

Compute frequency of words in a single line

```
package examples;
import java.util.*;
public class WordFrequencyLine {
  public static void main(String... args) {
      Map<String, Integer> frequency = new TreeMap<>();

      String line = "the hour of the truth is the hour of the death";
      for (String word : line.split(" ")) {
         if (frequency.containsKey(word))
               frequency.put( word, frequency.get(word)+1 );
         else frequency.put( word, 1 );
      }
      System.out.println(frequency);
      //
      // output: {death=1, hour=2, is=1, of=2, the=4, truth=1}
  }  // in alphabetical order
}
```

Compute frequency of words in a single line

```
package examples;
import java.util.*;
public class WordFrequencyLine {
  public static void main(String... args) {
      Map<String, Integer> frequency = new TreeMap<>();

      String line="the hour of the truth  is  the hour of the death";
      for (String word : line.split("\\s+")) {
         if (frequency.containsKey(word))
              frequency.put( word, frequence.get(word)+1 );
         else frequency.put( word, 1 );
      }
      System.out.println(frequency);
      //
      // output: {death=1, hour=2, is=1, of=2, the=4, truth=1}
  }
}
```

TreeMap and TreeSet are automatically sorted but there is no *Tree*List

```
package examples;
import java.util.*;
public class PalsListSorting {
  public static void main(String... args) {
      List<String> pals = new ArrayList<String>();
      List<String> yourPals = new LinkedList<String>();

      pals.add("Luca");  pals.add("Leo");   pals.add("Joe");
      yourPals.add("Pi"); yourPals.add("Ed"); yourPals.add("Mark");

     Collections.sort(pals );   Collections.sort(yourPals);

     System.out.println(pals );    // [Joe, Leo, Luca]
     System.out.println(yourPals); // [Ed, Mark, Pi]
  }
}
```

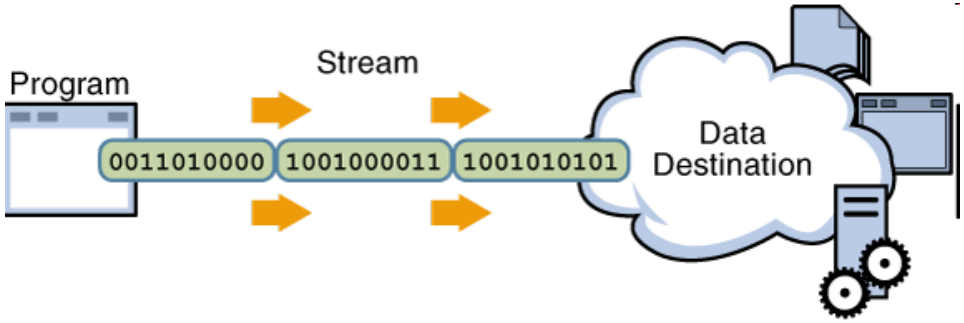
Processing text files

- Use predefined package `java.io`
- All input/output actions must consider exceptions (errors)
- We use an object hierarchy from the lowest level (byte) to higher levels (lines and objects)
- *`InputStream`* reads bytes
- *`Reader`* reads characters (2 bytes), *`Writer`* writes them
- `BufferedReader` forms lines with characters read by an `InputStreamReader` (which in turn requires a `FileInputStream` to first read byte by byte from binary)
- Sometimes we will also use `java.nio` (**n**ew **io** package)
- In Eclipse file names (for data, not for source code) are relative to the directory of the active Eclipse project

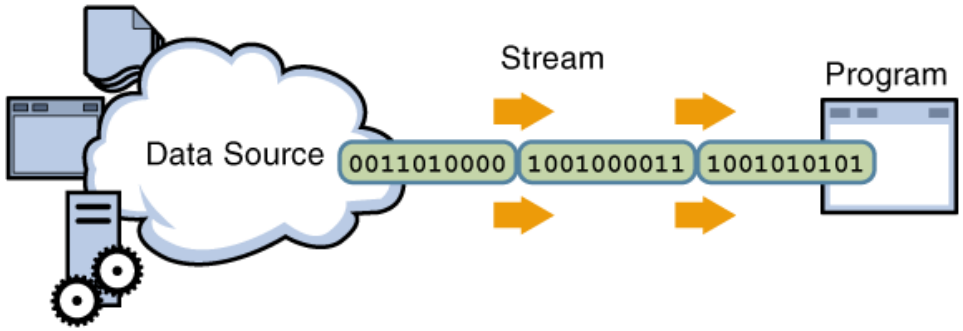
I/O (input/output) Streams

- **I/O streams** represent data sources or data destinations.

- Encapsulate the source/destination type:
  - disk file,
  - application,
  - physical device,
  - memory array,
  - communication port, ...

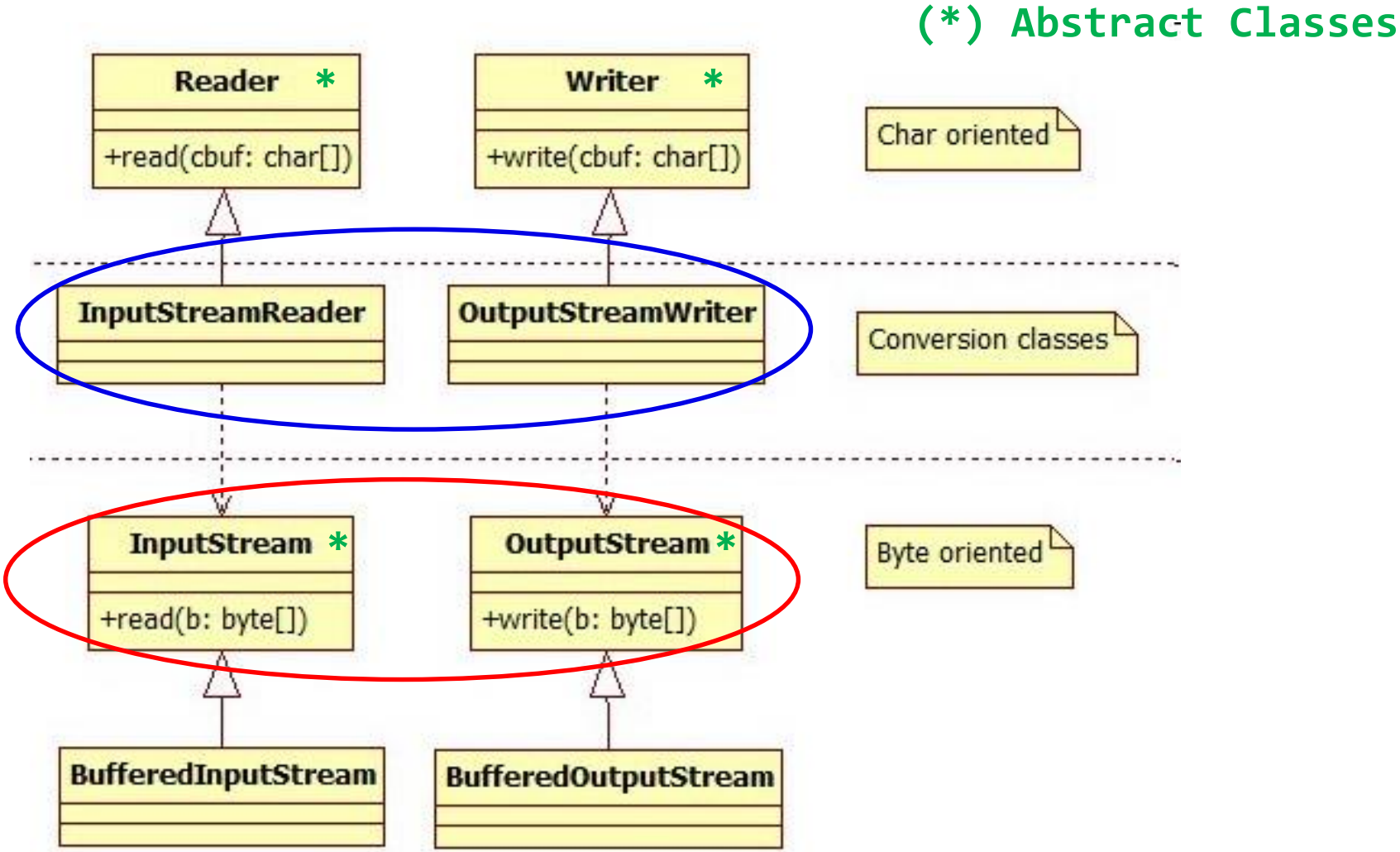


Output Stream

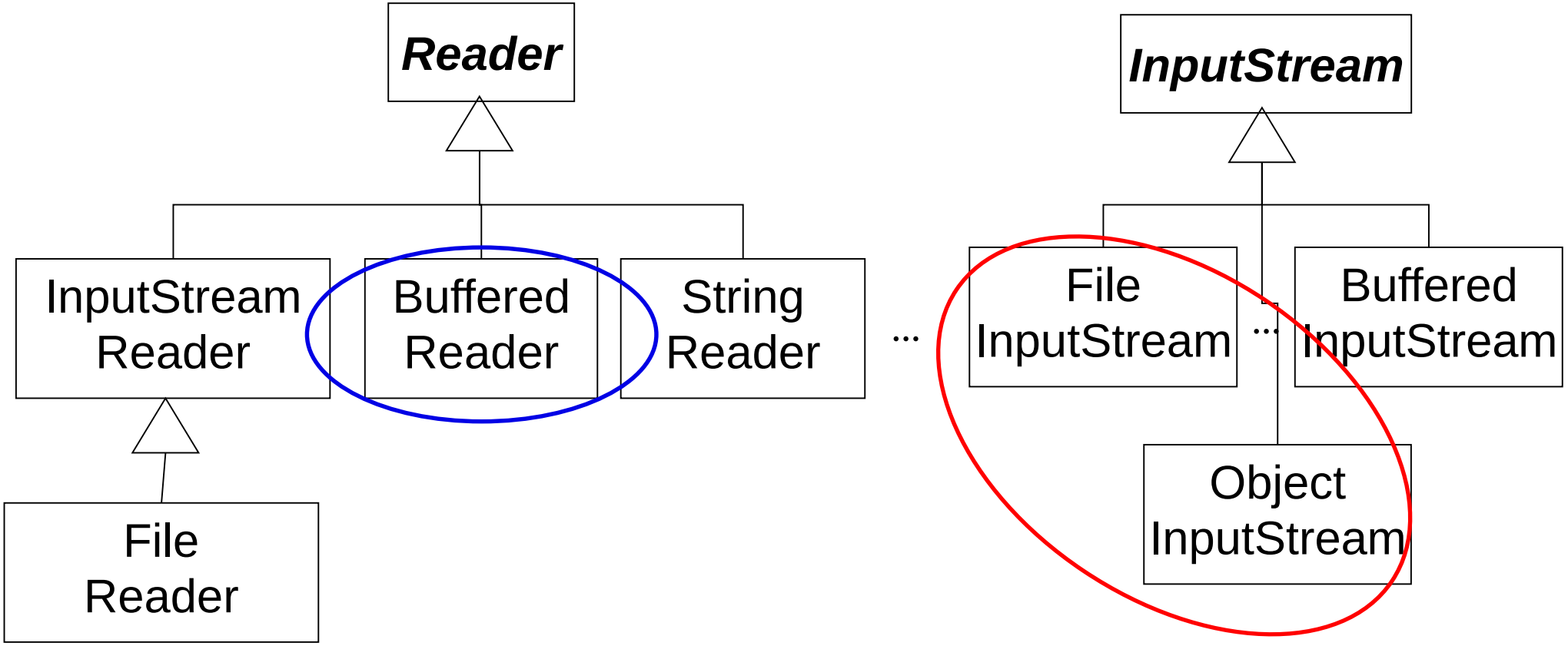


Input Stream

Basic I/O Classes



Basic I/O Classes



Reading characters **Reading bytes**

Read and process a text file (something missing)

```
package examples;
import java.io.*;
public class DemoInput {
  public static void main(String[] args)                    {
    FileInputStream stream = new FileInputStream( "text.txt" );
    InputStreamReader reader = new InputStreamReader(stream);
    BufferedReader buffer = new BufferedReader(reader);

    String line;
    while ((line = buffer.readLine()) != null)
      System.out.println( "Line read is: " + line);


    buffer.close();
 }
}
```

Read and process a text file (complete)

```
package examples;
import java.io.*;
public class DemoInput {
  public static void main(String[] args) throws IOException {
    FileInputStream stream = new FileInputStream( "text.txt" );
    InputStreamReader reader = new InputStreamReader(stream);
    BufferedReader buffer = new BufferedReader(reader);

    String line;
    while ((line = buffer.readLine()) != null)
      System.out.println( "Line read is: " + line);



    buffer.close();
 }
}
```

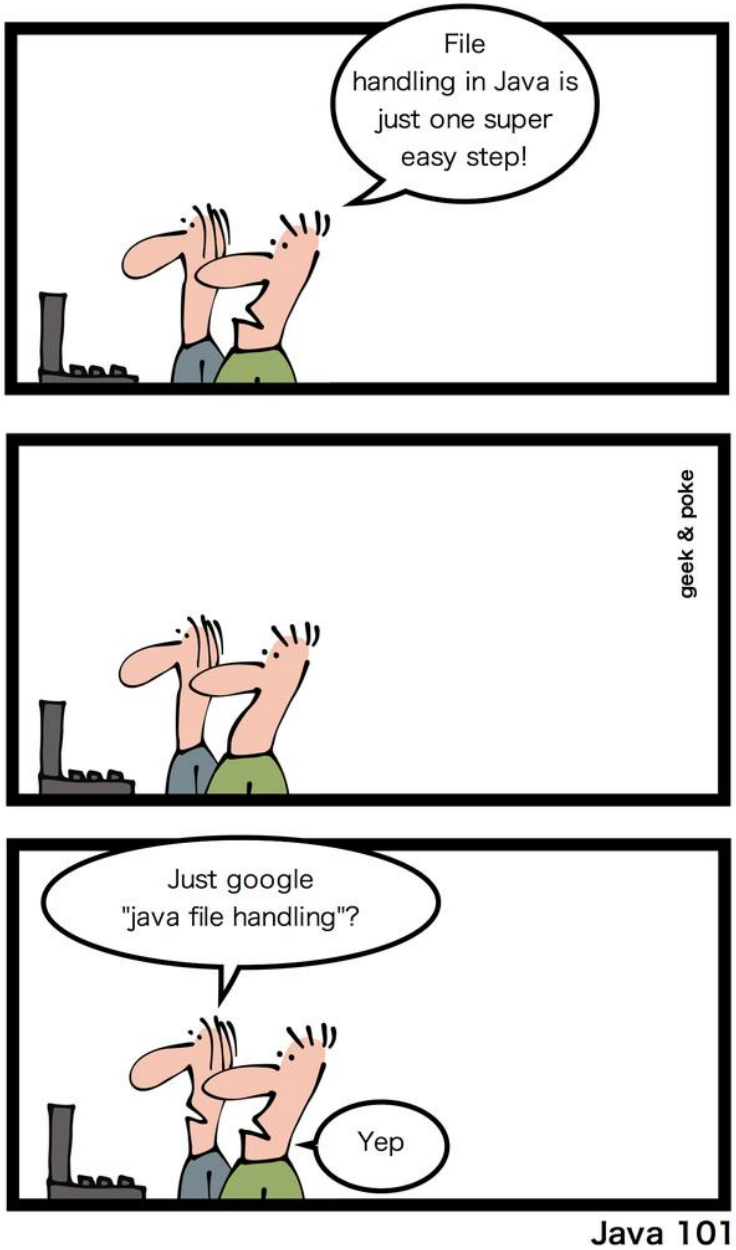
Read and process a text file (even simpler)

```
package examples;
import java.io.*;
public class DemoInput2 {
  public static void main(String[] args) throws IOException {
    BufferedReader buffer = new BufferedReader(
                                new InputStreamReader(
                                  new FileInputStream("text.txt")));

    String line;
    while ((line = buffer.readLine()) != null)
      System.out.println( "Line read is : " + line);



    buffer.close();
 }
}
```


File handling in Java is just one super easy step!
geek & poke
Just google "java file handling"?
Yep
Java 101

Frequency of words read from a text file

```
package examples;
import java.io.*; import java.util.*;
public class WordFrequencyFile {
  public static void main(String[] args) throws IOException {
    BufferedReader buffer = new BufferedReader(
                                new InputStreamReader(
                                  new FileInputStream("text.txt")));
    Map<String, Integer> frequency = new TreeMap<>();
    String line;
    while ((line = buffer.readLine()) != null) {
      for (String word : line.split(" "))
        if (frequency.containsKey(word))
            frequency.put(word, frequency.get(word)+1 );
        else frequency.put(word, 1 );
    }
    System.out.println(frequency);
    buffer.close(); }}
```

Frequence of words read from a text file

```
package examples;
import java.io.IOException; import java.nio.file.*;import java.util.*;
public class WordFrequencyFile2 {
  public static void main(String[] args) throws IOException {

    List<String> lines = Files.readAllLines(Paths.get("text.txt"));

    Map<String, Integer> frequency = new TreeMap<>();
    for (String line : lines){
      for (String word : line.split(" "))
        if (frequency.containsKey(word))
            frequency.put(word, frequency.get(word)+1 );
        else frequency.put(word, 1 );
    }
    System.out.println(frequency);
  }
}
```

Frequence of words using a Stream of lines from the file

```
package examples;
import java.io.IOException; import java.nio.file.*;import java.util.*;
public class WordFrequencyFileStream {
  public static void main(String[] args) throws IOException {

    Map<String, Integer> frequency = new TreeMap<>();

    Files.lines(Paths.get( "text.txt" ) )
          .forEach(line -> {
            for (String word : line.split(" "))
              if (frequency.containsKey(word))
                  frequency.put(word, frequency.get(word)+1 );
             else frequency.put(word, 1 );
          } );
    System.out.println(frequency);
  }
} // We'll see the details at the end of Lesson 3
```

Formatting output in text files

```
package examples;
import java.io.*;
import java.time.LocalDate;

public class TextOutput {
  public static void main(String[] args) throws IOException {

    FileOutputStream stream = new FileOutputStream("numbers.txt”);
    PrintWriter output = new PrintWriter(stream);

    for (double i = 0.15; i <= 0.20; i = i + 0.01)
        output.printf("%5.2f\n", i);
    output.printf( "\t%s\n\t==========", LocalDate.now());
    output.flush();
    output.close();
 }
}
```

Object persistence (input/output)

- With a `FileOutputStream` we create an `ObjectOutputStream` in which we can save objects in “binary”
- The same file can then be treated as a `FileInputStream`
- With a `FileInputStream` we create an `ObjectIntputStream` from which we read *the same* objects that we saved before, but they are read as `Objects` (thus, a *casting* is needed)
- The programmer is responsible of knowing what objects where saved (in what sequence) to read them accordingly
- Each read object is assigned (after casting) to a variable of a suitable data type for the saved object
- Otherwise, a `ClassCastException` error occurs
- **This requires the object’s class implement Serializable**

Example with a public class: Point

```
package es.uam.eps.ads.geometry;


public class Point  /* Something missing, not yet Serializable */ {
  private int x, y; // private components

  public Point(int x, int y) {   // constructor
    this.x = x;
    this.y = y;
  }

  @Override
  public String toString( ) { return "(" + x + "," + y + ")"; }

  // without the following method points would immutable
  public void shift(int dx, int dy) { x += dx; y += dy; }
}
```

Class Point used in another public class: Polygon

```
package es.uam.eps.ads.geometry;
import java.util.*;  // without import es.uam.eps.geometry.Point;

public class Polygon /* Something missing here */ {
   private List<Point> points = new ArrayList<>();

   public Polygon add(Point p) { points.add(p);  return this; }

   public String toString() {
      String result = "<";
      for (Point p : points)
         result += p;  // p.toString();
      return result + ">";
   }
   public void shift(int dx, int dy) {
      for (Point p : points) p.shift(dx, dy);
   }
}
```

Persistence of *instances* from class Point

```
package examples.persistence;
import java.io.*;   import java.util.*;
import es.uam.eps.ads.geometry.Point;

public class PointPersistence {
  public static void main(String[] args) throws IOException {

    ObjectOutputStream objectOutput =
         new ObjectOutputStream(
             new FileOutputStream( "points.objectData" ) );
    List<Point> points =
        new LinkedList<>(Arrays.asList(
                new Point(3,4), new Point(0,3), new Point(2,6) ));

    objectOutput.writeObject(points);
    objectOutput.close();
  }
}
```

Class Point with persistence (serialized)

Declare the class as *serializable* was missing before

```
package es.uam.eps.ads.geometry;
import java.io.Serializable;

public class Point  implements  Serializable  {
  private int x, y; // private components

  public Point(int x, int y) {   // constructor
    this.x = x;
    this.y = y;
  }

  public String toString( ) { return "(" + x + "," + y + ")"; }

  // without the following method points would inmutable
  public void shift(int dx, int dy) { x += dx; y += dy; }
}
```

Reading Point objects that were serialized in a file

```
package examples.persistence;
import java.io.*;    import java.util.*;
import es.uam.eps.ads.geometry.Point; // important

public class ReadPoints {
  public static void main(String[] args) throws Exception {

      ObjectInputStream objectInput =
              new ObjectInputStream(
                  new FileInputStream( "points.objectData" ) );

      List<Point> points = (List<Point>) objectInput.readObject();

      objectInput.close();
      System.out.println("Read in: " + points);
  }
}
```

Reading Point objects that were serialized in a file

```
package examples.persistence;
import java.io.*;    import java.util.*;
import es.uam.eps.ads.geometry.Point; // important

public class ReadPoints {
  public static void main(String[] ar        ws Exception {

    ObjectInputStream objectInput
            new ObjectInputStream
                new FileInputSt      ( "points.objectData" ) );

    List<Point> points = (Lis   Point>) objectInput.readObject();

    objectInput.close();
    System.out.println("Read in: " + points);
  }
}
```

```
Read in: [(3,4), (0,3), (2,6)]
```

Reading Point objects that were serialized in a file

```
package examples.persistence;
import java.io.*;    import java.util.*;
import es.uam.eps.ads.geometry.Point; // important

p
```

Warning: Modifying the description of the serialized class (and recompiling it) *may* turn previously serialized objects incompatible to be read using the new class definition.

```
    ObjectInputStream objectInput =
            new ObjectInputStream(
                new FileInputStream        nts.objectData" ) );

    List<Point> points = (List<Point>) objectInput.readObject();

    objectInput.close();
    System.out.println("Read in: " + points);
  }
}
```

Object Persistence (input/output)

- This form of serialization is very simple
- It has limitations:
  - If the class of serialized objects is changed, serialized objects will most likely be unreadable by means of the new class definition
  - Static attributes are not persisted
- However, this is far better than converting objects into a text form to implement persistence in text files
- There are other advanced forms of persistence
  - JavaBeans, XML, JSON, JDBC, …

Other example: persistence of class Polygon (1/2)

```
package examples.persistence;
import java.io.*;   import java.util.*;
import es.uam.eps.ads.geometry.*;

public class PolygonPersistence {
     public static void main(String[] args) throws IOException {

        ObjectOutputStream outputLines =
               new ObjectOutputStream(
                   new FileOutputStream( "polygons.objectData" ));

        Polygon segment = new Polygon();
        Polygon square = new Polygon();

            // continues ...
```

Other example: persistence of class Polygon (2/2)

```
        segment.add(new Point(1,1)).add(new Point(3,2));

        square.add(new Point(0,0)).add(new Point(0,2))
                .add(new Point(2,2)).add(new Point(0,2))
                .add(new Point(0,0));

        System.out.println( segment );
        System.out.println( square );

        outputLines.writeObject(segment);
        outputLines.writeObject(square);

        outputLines.close();
    }
}
```

Class Polygon with persistence (serialized)

```
package es.uam.eps.ads.geometry;
import java.util.*;  // without  import es.uam.eps.geometry.Point;
import java.io.Serializable;
public class Polygon implements Serializable {
  private List<Point> points = new ArrayList<>();

  public Polygon add(Point p) { points.add(p); return this; }

  public String toString() {
     String result = "<";
     for (Point p : points) { result += p; };
     return result + ">";
  }
  public void shift(int dx, int dy) {
     for (Point p : points) { p.shift(dx, dy); };
  }
}
```

Read Polygon objects previously serialized in file

```
package examples.persistence;
import java.io.*;   import java.util.*;
import es.uam.eps.ads.geometry.Polygon;

public class ReadPolygons {
  public static void main(String[] args) throws Exception {
      ObjectInputStream objectInput =
             new ObjectInputStream(
                new FileInputStream( "polygons.objectData" ) );
      Polygon p1 = (Polygon) objectInput.readObject();
      Polygon p2 = (Polygon) objectInput.readObject();
      objectInput.close();
      System.out.println("Segment: " + p1);
      System.out.println("Square: " + p2);
      p2.shift(10,100);
      System.out.println("Square shifted: " + p2);
  }
}
```

Read Polygon objects previously serialized in file

```
Segment: <(1,1)(3,2)>
Square: <(0,0)(0,2)(2,2)(2,0)(0,0)>
Square shifted: <(10,100)(10,102)(12,102)(12,100)(10,100)>
```

```
  public stat                    ing[] args) throws Exception {
      ObjectInpu                  Input =
             new Obj              eam(
                new Fil           eam( "polygons.objectData" ) );
      Polygon p1 = (Polygon)      ctInput.readObject();
      Polygon p2 = (Polygon) ob   tInput.readObject();
      objectInput.close();
      System.out.println("Segment: " + p1);
      System.out.println("Square: " + p2);
      p2.shift(10,100);
      System.out.println("Square shifted: " + p2);
  }
}
```