



Tema 2

Introducción a Java

Análisis y Diseño de Software

2º Ingeniería Informática

Universidad Autónoma de Madrid



2.1. Introducción a Java

- **Presentación, orígenes, entorno**
- Elementos básicos del Lenguaje
- Ejercicios

Java: Un vistazo general

- Lenguaje compilado a *Bytecode*, que se interpreta.
- “Fuertemente tipado” y con facilidades para la **modularización**
- Sintaxis similar a C y C++, pero de **programación más fiable**
- Sin punteros explícitos, con *garbage collection* (recogida de basura)
- **100% portable** (a menos que lo contrario sea el objetivo)
“*write once, run everywhere*”
- Incorpora **conurrencia** (`Thread`) y **manejo de excepciones**
- Integra librerías estándar (I/O, math, ...) y de **extensión**:
 - Interfaces gráficas de usuario (Swing), objetos distribuidos (rmi), documentos XML, criptografía, ... y muchas más
- Ejecutable desde navegadores web (vía `Applet`)

Orígenes y Ediciones de Java

- **Orígenes de Java** (1991—1995, Sun Microsystems, Inc.)
 - James Gosling.
 - Para programar electrodomésticos y equipos pequeños
 - Crearon un lenguaje nuevo, Oak, que renombraron Java
 - No hubo éxito con la nueva tecnología empotrada
 - La Web y los navegadores captaban más interés
 - El boom fue crear un navegador más “inteligente”, **HotJava** programado en Java y el primer browser en soportar Applets.
 - Los teléfonos de la UAM (desde ~2010) usan Java ¡Por fin!
- **Ediciones de Java**
 - **Java SE (Standard Edition)**, EE (Enterprise Edition), ME (Micro Edition).
 - Distribuidas por Oracle, empresa que compró a Sun Microsystems

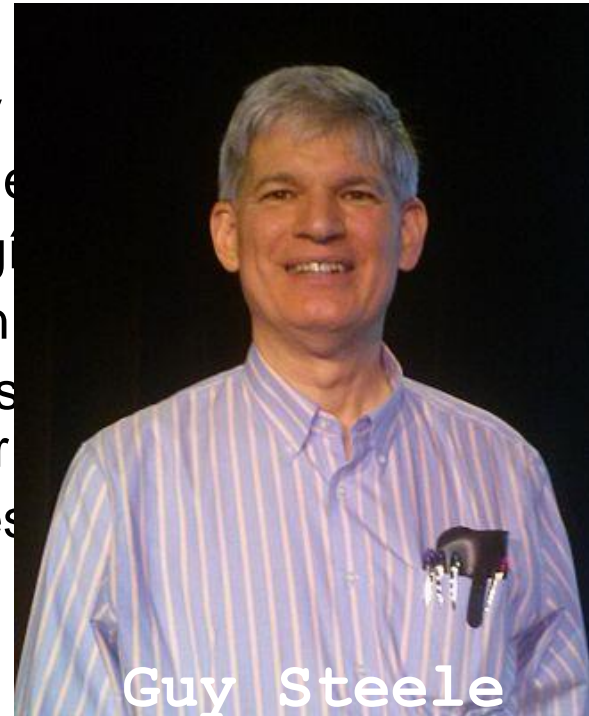
Orígenes y Ediciones de Java

■ Orígenes de Java (1991—1995, Sun Microsystems, Inc.)

- Ja
- P
- C
- N
- L
- E
- J
- L



James Gosling

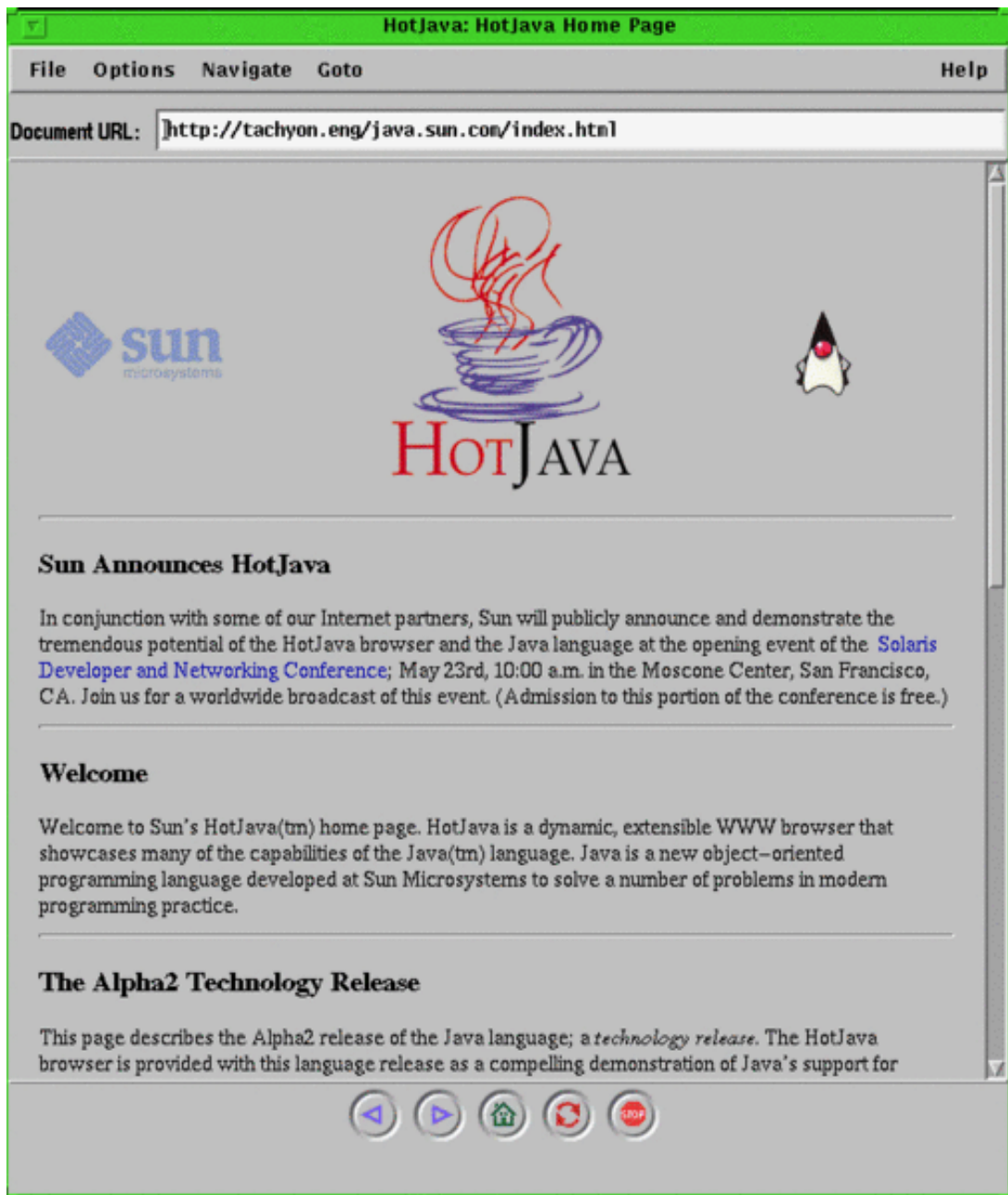


Guy Steele

■ Edic

- **Java SE (Standard Edition)**, EE (Enterprise Edition), ME (Micro Edition).
- Distribuidas por Oracle, empresa que compró a Sun Microsystems

HotJava



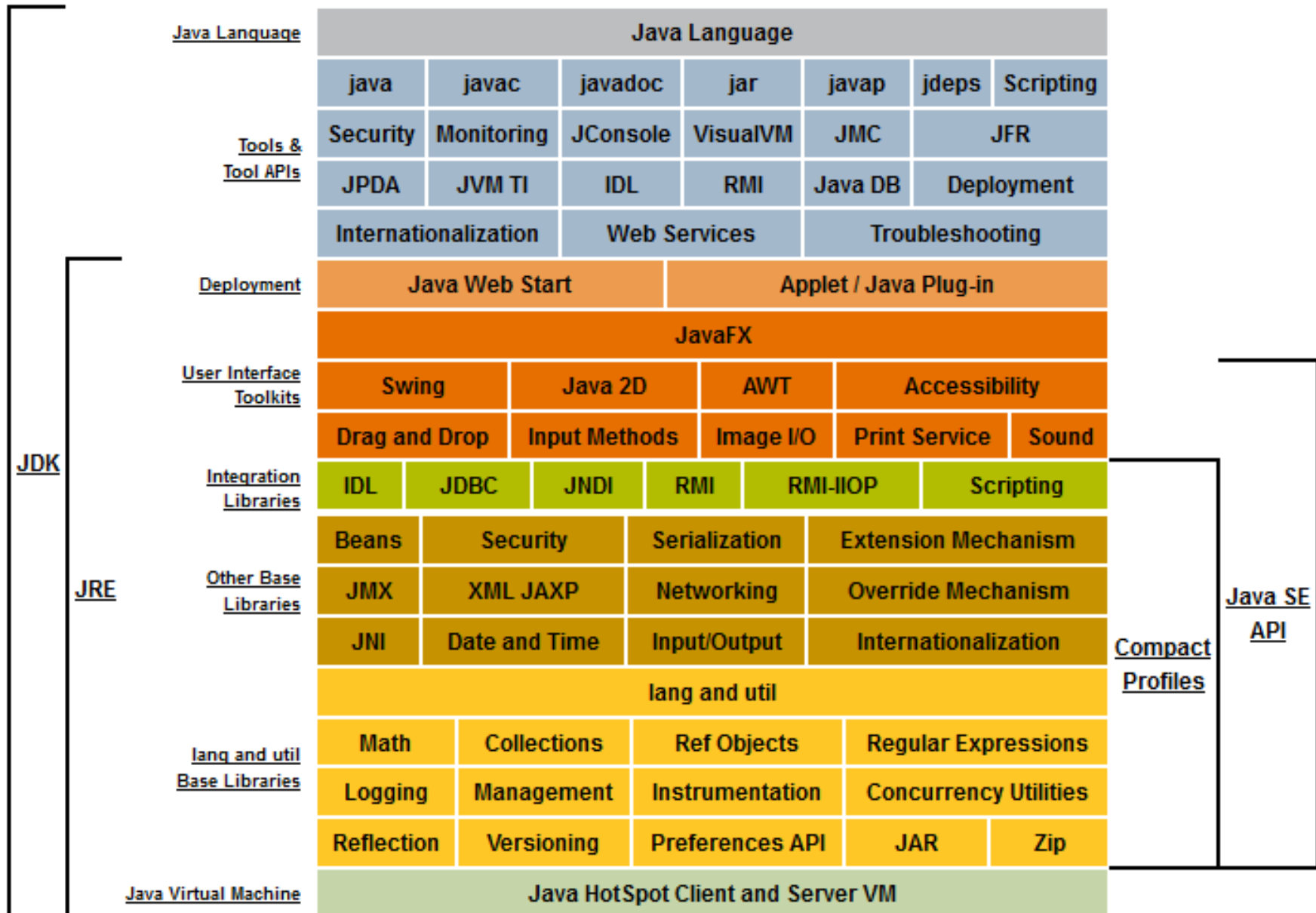
Versiones de Java

- Java 1.0 – Enero 1996 Unos cientos de clases
- Java 1.1 – Febrero 1997 JavaBeans, JDBC, RMI...
- J2SE 1.2 – Diciembre 1998 Swing, Java IDL, Collections...
- Así nació “**Java 2**”
- J2SE 1.3 – Mayo 2000 Compatible RMI/CORBA...
- J2SE 1.4 – Febrero 2002 XML, JWS, mejoras internas...
- J2SE 5.0 (Java 1.5) – septiembre 2004
 - Genéricos, enumeraciones, enhanced “for”...
- Java SE 6 (Java 1.6) – diciembre 2006
 - más de 3.000 clases, mejoras fuertes de rendimiento...
- Java SE 7 – 2011/12
 - mejora de la gestión de excepciones, permite String en switch, ...
- Java SE 8 – marzo 2014
 - Expresiones **lambda** (closures), **streams**, mejora en interfaces...
- Java SE 9 – Sept. 2017 módulos, JShell
- Java SE 10 – Marzo 2018 inferencia de tipos de variables locales
- Java SE 11 – Sept. 2018
- Java SE 12 – Marzo 2019
- Java SE 13 – Sept. 2019 switch mejorado, bloques de texto
- Java SE 14 – Marzo 2020 instance of mejorado, records, switches
- Java SE 15 – Sept. 2020 Sealed classes
- ...
- Java SE 19 – Sept. 2022 (dos versiones al año)

Java Development Kit (JDK)

- Java Runtime Environment (JRE)
 - Java Virtual Machine (JVM)
 - Java API (*Application Programming Interface*):
 - Lenguaje básico Java + Librerías estándar Java
 - <http://java.sun.com/docs/books/jls/>
 - <http://docs.oracle.com/javase/19/docs/api/>
- Compilar a *Bytecode* (*javac Ejemplo.java*) y ejecutar (*java Ejemplo*)
- Otras herramientas generales: javadoc, appletviewer, [jshell](#), ...
- Y específicas del IDE (*Interactive Development Environment*): plantillas, editores orientados a sintaxis, depuradores, ...

Plataforma Java SE



Ejemplo de organización de código

Fichero 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);
    }
}
```

- Definición de clases
- Convenciones de estilo
- Método main
- Declaración y creación de objetos

- Llamada a métodos de objeto
- Clase String, concatenación
- Paquetes
- Ámbito de variables
- Variables y métodos de instancia
- Variables y métodos de clase

Ejemplo: organización mejorada

```
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 + ")";  
    }  
}
```

Fichero Point.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);  
    }  
}
```

Fichero PointTest.java

Todavía mayor organización (paquetes)

```
package geometry;
```

Fichero geometry/Point.java

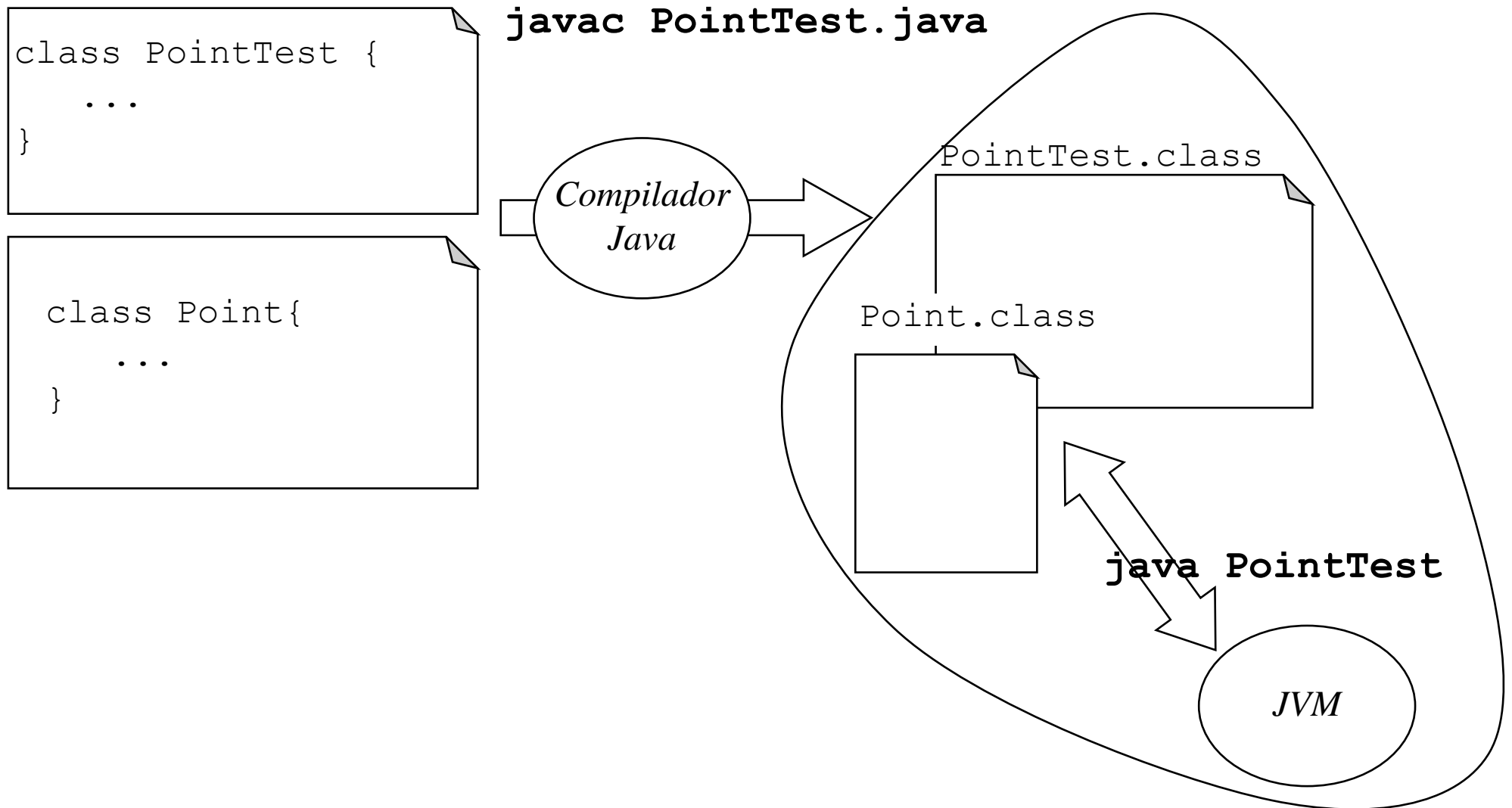
```
public class Point {  
    private int x;  
    private int y;  
    public Point(int x, int y) { this.x = x; this.y = y; }  
    public void shift(int dx, int dy) { this.x+=dx; this.y+=dy; }  
    public String toString() {  
        return "(" + x + "," + y + ")";  
    }  
}
```

```
package geometry.test;  
import geometry.Point;
```

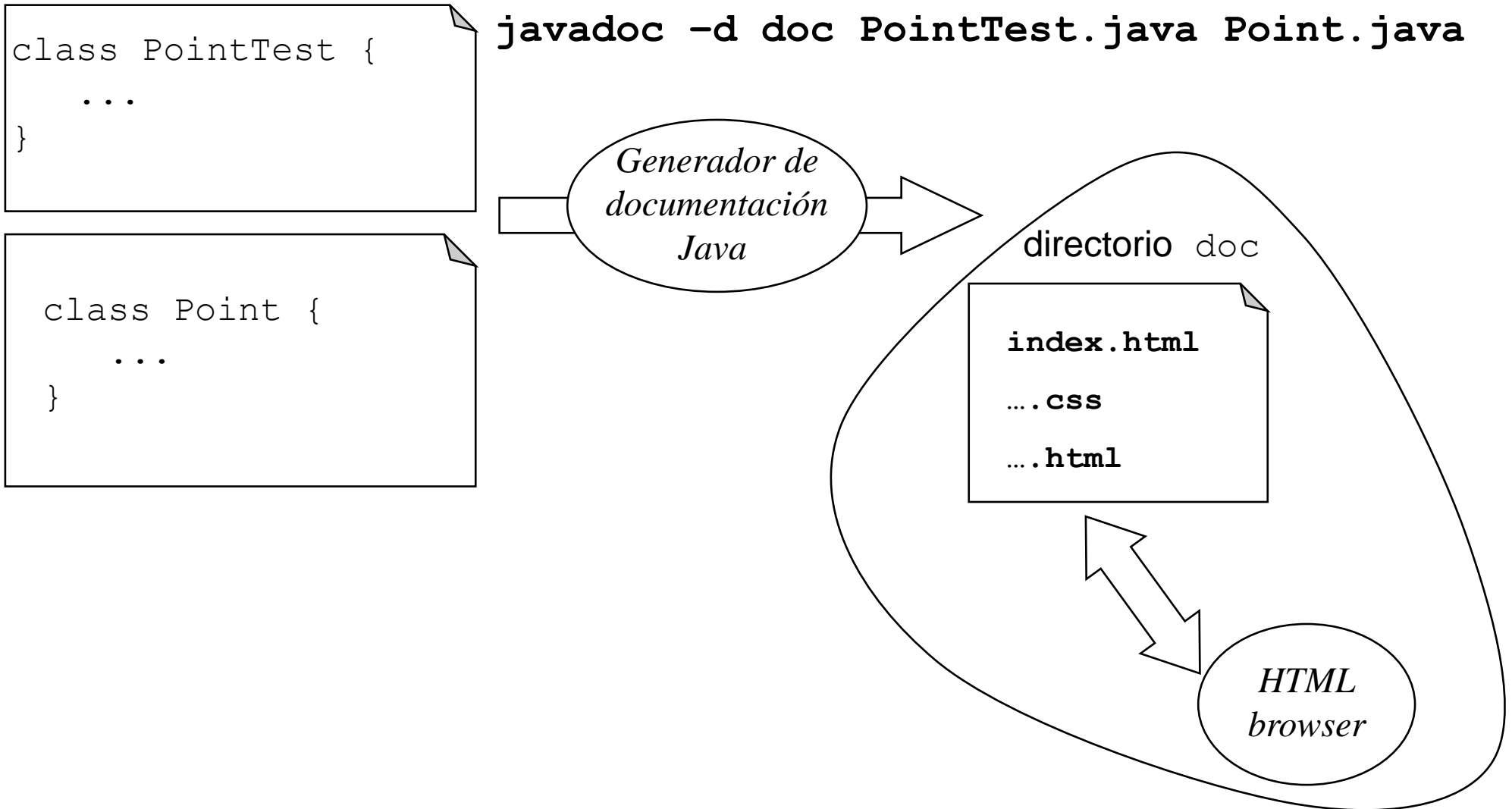
Fichero geometry/test/PointTest.java

```
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);  
    }  
}
```

Compilación y Ejecución



Generación de documentación





Java vs. C

Java es orientado a objetos

Interpretado (bytecode)

Totalmente portable

Memoria dinámica automática
garbage collection

No existen punteros explícitos

Tipos abstractos de datos con
protección real

Mecanismos de modularización

Conceptos modernos de
programación: excepciones, ...

C no es orientado a objetos

Compilado

Aspectos no portables (`sizeof(int)`)

Memoria dinámica gestionada por
el programador

Punteros: difícil programación fiable

Tipos abstractos de datos ficticios
con **struct** y separando ***.h** ***.c**

`#include` es sólo “cortar y pegar”

Es casi como comparar
ensamblador con C

3.1. Introducción a Java

- Presentación, orígenes, entorno

- **Elementos básicos del Lenguaje**

- ☐ Tipos de datos primitivos
- ☐ Tipos de datos no primitivos
- ☐ Instrucciones de control
- ☐ Entrada/Salida
- ☐ Aplicaciones ejecutables vía Web

- Ejercicios

3.1. Introducción a Java

- Presentación, orígenes, entorno
- **Elementos básicos del Lenguaje**
 - (ver apéndices)
- **Introducción mediante ejemplos**
- Ejercicios

Ejemplos elementales

- Elementos sintácticos
 - Guía de estilo (ver Moodle ADSOF, sección Prácticas)
 - Comentarios simples
 - Comentarios *Javadoc*
 - Estructura de un programa principal (aplicación)
-
- Los ejemplos no muestran los comentarios *Javadoc* para facilitar su proyección en la pantalla

Ejemplo Hola Mundo

```
package ejemplos; /* archivo ejemplos/HelloWorld.java */
```

```
public class HelloWorld {
```

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

```
}
```

```
//Guía de estilo:
```

```
// HelloWorld, String y System son clases
```

```
// out es un objeto
```

```
// println es un método (operación sobre un objeto)
```

```
// main también es un método (el método inicial)
```

Ejemplo Feliz 2023

```
package ejemplos; // archivo ejemplos/Happy2023.java
```

```
public class Happy2023 {
```

```
    public static void main(String... args) {  
        System.out.println("Happy 2023!");  
    }
```

```
}
```

```
// String[] indicaba un array de cadenas de caracteres  
// String... indica un número variable de parámetros (todos ellos  
// cadenas de caracteres)
```

Ejemplo Feliz Año Nuevo: importando clases

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

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

```
}
```

```
// antes de codificar algo conviene mirar si ya existe  
// el software reusable se agrupa en paquetes Java  
// cuyos componentes públicos se puede importar
```

Ejemplo Feliz Año Nuevo + "!"

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

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

```
}
```

```
// el operador + está sobrecargado, tiene más de un significado:  
// aquí sirve para concatenar Strings  
// pero también sirve sumar números, e.g.: 2 + 3, x + 0.5, ...
```

Ejemplos: bucles, listas, ...

- Variables locales: inicialización, asignación
- Expresiones aritméticas
- Estructuras de control: for sin índice, if, while, ...
- Creación de objetos (inmutables o no)
- Uso de paquetes Java predefinidos (librerías)
- Manejo básico de listas: `package java.util.*`
 - abstracción de implementación (`ArrayList`, `LinkedList`)
 - recorrido con for
 - acceso por índice
 - búsqueda
 - recorrido con streams (detalles al final del Tema 3)

Selecciona y suma los números que sean pares

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

        System.out.println( numeros );

    }
}
```


Selecciona y suma los números que sean pares

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

        System.out.println( numeros );

        int totalPares = 0;

                                totalPares = totalPares + num;

        System.out.println( "La suma de pares es: " + totalPares);
    }
}
```

Selecciona y suma los números que sean pares

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

        System.out.println( numeros );

        int totalPares = 0;
        for (Integer num : numeros) {
            if (num % 2 == 0) totalPares = totalPares + num;
        }
        System.out.println( "La suma de pares es: " + totalPares);
    }
}
```

Cuenta los números que sean pares

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

        System.out.println( "El número de pares es: "
            + numeros.stream().filter(n -> n % 2 == 0).count());
    }
}
```

```
// este ejemplo usa el estilo de programación funcional
// utiliza los conceptos stream y expresión lambda
// que veremos con más detalle al final del Tema 3
```

Suma los números que sean pares

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

        System.out.println( "La suma de pares es: "
            + numeros.stream().filter(n -> n % 2 == 0)
                .reduce( Integer::sum ));
    }
}
```

```
// este ejemplo usa el estilo de programación funcional
// utiliza los conceptos stream y expresión lambda
// que veremos con más detalle al final del Tema 3
```

Ejemplo básico acceso a listas

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

        System.out.println( numeros );

        System.out.println( numeros.size() + " números" );
        System.out.println( "Primer número es: " + numeros.get(0) );
        System.out.println( "Último número es: "
                               + numeros.get( numeros.size() - 1) );
    }
} // la lista numeros es immutable: ni añadir ni borrar elementos
```

Implementaciones concretas de listas

```
package ejemplos;
import java.util.*; //import java.util.List; import java.util.ArrayList;
public class ListaAmigos1 {
    public static void main(String... args) {
        List<String> amigos = new ArrayList<String>();

        amigos.add("Luis"); amigos.add("Leo"); amigos.add("José");

        System.out.println( amigos );
        System.out.println( amigos.size() + " amigos" );
        System.out.println( "Primer amigo es: " + amigos.get(0) );
        System.out.println( "Último amigo es: "
                               + amigos.get( amigos.size() - 1) );
    }
} // new crea un objeto nuevo de la clase indicada
```

Implementaciones concretas de listas

```
package ejemplos;
import java.util.*; //import java.util.List; import java.util.ArrayList;
public class ListaAmigos1 {
    public static void main(String... args) {
        List<String> amigos = new ArrayList<String>();

        amigos.add("Luis"); amigos.add("Leo"); amigos.add("José");

        System.out.println( amigos ); // salida: [Luis,Leo,José]
        System.out.println( amigos.size() + " amigos" ); // salida: 3
        System.out.println( "Primer amigo es: " + amigos.get(0) );
        System.out.println( "Último amigo es: "
                               + amigos.get( amigos.size() - 1) );
    }
} // new crea un objeto nuevo de la clase indicada
```

Otro ejemplo de recorrido de listas

```
package ejemplos;
import java.util.*;
public class ListaAmigos2 {
    public static void main(String... args) {
        List<String> amigos = new ArrayList<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");

        System.out.println( amigos );

        for (String amigo : amigos) {
            System.out.println( "Hello " + amigo + "!" );
        }
    }
} // la lista amigos NO es inmutable
```


Recorrido de otra implementación de listas

```
package ejemplos;
import java.util.*;
public class ListaAmigos3 {
    public static void main(String... args) {
        List<String> amigos = new LinkedList<>();

        amigos.add("Luis"); amigos.add("Leo"); amigos.add("José");

        System.out.println( amigos );

        for (String amigo : amigos) {
            System.out.println( "Hello " + amigo + "!" );
        }
    }
} // IMPORTANTE:
// el código para recorrerla no depende de la implementación
```

Ejemplo básico: búsqueda en listas

```
package ejemplos;
import java.util.*;
public class ListaAmigos4 {
    public static void main(String[] args) {
        List<String> amigos = new ArrayList<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");

        System.out.println( amigos );

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

Ejemplo básico: uso de argumentos de main

```
package ejemplos;
import java.util.*;
public class ListaAmigos5 {
    public static void main(String[] args) {
        List<String> amigos = new ArrayList<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");

        System.out.println( amigos );

        if ( amigos.contains( args[0] ) ) { // ¿algún error?
            System.out.println( "Hello " + args[0] + "!" );
        }
    }
}
```

Ejemplo básico: if con parte else

```
package ejemplos;
import java.util.*;
public class ListaAmigos6 {
    public static void main(String[] args) {
        List<String> amigos = new ArrayList<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");

        System.out.println( amigos );

        if ( amigos.contains( args[0] ) ) { // ¿algún error?
            System.out.println( "Hello " + args[0] + "!" );
        } else {
            System.out.println( "Hello desconocido!" );
        }
    }
}
```

Ejemplo básico: control de errores

```
package ejemplos;
import java.util.*;
public class ListaAmigos7 {
    public static void main(String[] args) {
        List<String> amigos = new ArrayList<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");

        System.out.println( amigos );

        if ( args.length >= 1 && amigos.contains( args[0] ) ) {
            System.out.println( "Hello " + args[0] + "!" );
        }
    }
} // para evitar excepción no controlada:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 11
```

Recordatorio: abstracción de la implementación

```
package ejemplos;
import java.util.List;
public class ListaAmigos7Bis {
    public static void main(String[] args) {
        List<String> amigos = List.of("Luis", "Leo", "José");

        System.out.println( amigos );

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

Ejercicio: filtrado de listas

- Escribe una clase de utilidad FilterList con un método filter que:
 - reciba una lista de Strings y
 - devuelva la misma lista, borrando los elementos de tamaño menor que 3
- Mejora la clase para que podamos parametrizarla con la función de filtrado
 - Generalizar la función de filtrado para poder filtrar Strings de longitud menor que X
- ¿Cómo hacer para que el método admita cualquier tipo de lista, no necesariamente de Strings?

Ejemplos con colecciones de datos

- Más ejemplos con el paquete `java.util.*`
- Manejo básico de colecciones:
 - Listas: `addAll` (operaciones avanzadas, bulk)
 - Conjuntos, `Set`, con varias implementaciones
 - `HashSet`, `TreeSet`, ...
 - Mapas (tablas hash), `Map`, varias implementaciones
 - `HashMap`, `TreeMap`, ...
- Construir colecciones a partir de otras
- Colecciones inmutables
- Ordenación de colecciones

Juntando listas

```
package ejemplos;
import java.util.*;
public class ListaAmigos8 {
    public static void main(String... args) {
        List<String> amigos = List.of("Luis", "Leo", "José");
        List<String> tusAmigos = new LinkedList<>();

        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Mar");

        tusAmigos.addAll( amigos );

        System.out.println( tusAmigos );

    }
}
```

Juntando listas

```
package ejemplos;
import java.util.*;
public class ListaAmigos8 {
    public static void main(String... args) {
        List<String> amigos = List.of("Luis", "Leo", "José");
        List<String> tusAmigos = new LinkedList<>();

        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Mar");

        tusAmigos.addAll( amigos );

        System.out.println( tusAmigos );//salida:[Pi,Ed,Mar,Luis,Leo,José]

    }
}
```

Juntando listas

```
package ejemplos;
import java.util.*;
public class ListaAmigos8 {
    public static void main(String... args) {
        List<String> amigos = List.of("Luis", "Leo", "José");
        List<String> tusAmigos = new LinkedList<>();

        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Mar");

        tusAmigos.addAll( amigos );

        System.out.println( tusAmigos );//salida:[Pi,Ed,Mar,Luis,Leo,José]

        // Error al ejecutar: amigos.addAll( tusAmigos );
    }
} // la lista amigos es inmutable: ni añadir ni borrar elementos
```

Juntando listas

```
package ejemplos;
import java.util.*;
public class ListaAmigos8Java8 {
    public static void main(String... args) {
        List<String> amigos = new ArrayList<>(List.of("Luis", "Leo", "José"));
        List<String> tusAmigos = new LinkedList<>();

        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Mar");

        tusAmigos.addAll( amigos );

        System.out.println( tusAmigos );//salida:[Pi,Ed,Mar,Luis,Leo,José]

        amigos.addAll( tusAmigos );
    }
} // la lista amigos NO es inmutable
```

Juntar listas no elimina duplicados

```
package ejemplos;
import java.util.*;
public class ListaAmigos9 {
    public static void main(String... args) {
        List<String> amigos = new ArrayList<String>();
        List<String> tusAmigos = new LinkedList<String>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");
        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Mar");

        amigos.addAll( tusAmigos );
        tusAmigos.addAll( amigos ); // ¿algún error? Eso depende...
        System.out.println( amigos );
        System.out.println( tusAmigos );
    }
}
```

Juntar listas no elimina duplicados

```
package ejemplos;
import java.util.*;
public class ListaAmigos9 {
    public static void main(String... args) {
        List<String> amigos = new ArrayList<String>();
        List<String> tusAmigos = new LinkedList<String>();

        amigos.add("Luis"); amigos.add("Leo"); amigos.add("José");
        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Mar");

        amigos.addAll( tusAmigos );
        tusAmigos.addAll( amigos ); // ¿algún error? Eso depende...
        System.out.println( amigos );//salida:[Luis,Leo,José,Pi,Ed,Mar]
        System.out.println( tusAmigos ); //salida:
                                           [Pi,Ed,Mar,Luis,Leo,José,Pi,Ed,Mar]
    }
}
```

Para evitar duplicados usamos **conjuntos**

```
package ejemplos;
import java.util.*;
public class ConjuntoAmigos1 {
    public static void main(String... args) {
        Set<String> amigos = new TreeSet<>();
        Set<String> tusAmigos = new TreeSet<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");
        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Leo");

        amigos.addAll( tusAmigos );
        tusAmigos.addAll( amigos );
        System.out.println( amigos );
        System.out.println( tusAmigos );
    }
}
```

Para evitar duplicados usamos **conjuntos**

```
package ejemplos;
import java.util.*;
public class ConjuntoAmigos1 {
    public static void main(String... args) {
        Set<String> amigos = new TreeSet<>();
        Set<String> tusAmigos = new TreeSet<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");
        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Leo");

        amigos.addAll( tusAmigos );
        tusAmigos.addAll( amigos );
        System.out.println( amigos );    // [Ed, José, Leo, Luis, Pi]
        System.out.println( tusAmigos ); // [Ed, José, Leo, Luis, Pi]
    }
}
```


Y con otra implementación de conjuntos

```
package ejemplos;
import java.util.*;
public class ConjuntoAmigos2 {
    public static void main(String... args) {
        Set<String> amigos = new HashSet<>();
        Set<String> tusAmigos = new HashSet<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");
        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Leo");

        tusAmigos.addAll( amigos );
        amigos.addAll( tusAmigos );
        System.out.println( amigos );
        System.out.println( tusAmigos );
    }
}
```

Y con otra implementación de conjuntos

```
package ejemplos;
import java.util.*;
public class ConjuntoAmigos2 {
    public static void main(String... args) {
        Set<String> amigos = new HashSet<>();
        Set<String> tusAmigos = new HashSet<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");
        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Leo");

        tusAmigos.addAll( amigos );
        amigos.addAll( tusAmigos );
        System.out.println( amigos );    // Leo, Luis, José, Pi, Ed
        System.out.println( tusAmigos ); // Luis, Leo, José, Pi, Ed
    }
} // el orden es irrelevante, excepto en SortedSet como TreeSet
```

Todas las implementaciones de conjuntos son conjuntos

```
package ejemplos;
import java.util.*;
public class ConjuntoAmigos3 {
    public static void main(String... args) {
        Set<String> amigos = new HashSet<>();
        Set<String> tusAmigos = new TreeSet<>();

        amigos.add("Luis");  amigos.add("Leo");  amigos.add("José");

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

        tusAmigos.addAll( amigos );
        amigos.addAll( tusAmigos );

        System.out.println( tusAmigos.equals( amigos ) ); // true
    }
}
```

También hay conjuntos inmutables

```
package ejemplos;
import java.util.*;
public class ConjuntoAmigos4 {
    public static void main(String... args) {
        Set<String> amigos = Set.of("Luis", "Leo", "José");
        Set<String> tusAmigos = new TreeSet<>();

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

        // amigos.addAll( tusAmigos ); // error: amigos es immutable
        tusAmigos.addAll( amigos );

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

También hay conjuntos inmutables

```
package ejemplos;
import java.util.*;
public class ConjuntoAmigos4Java8 {
    public static void main(String... args) {
        Set<String> amigos= new HashSet<>(List.of("Luis","Leo","José"));
        Set<String> tusAmigos = new TreeSet<>();

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

        amigos.addAll( tusAmigos ); // amigos NO es inmutable
        tusAmigos.addAll( amigos );

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

Ejercicio

- Crea una clase SortInput que:
 - recibe uno o más números como parámetros por la línea de comandos
 - Los imprime en orden ascendente sin repetición
 - Imprime el más pequeño
 - Imprime el mayor

Los “mapas” más potentes que listas y conjuntos

```
package ejemplos;
import java.util.Map;
public class EdadesAmigos1 {
    public static void main(String... args) {
        Map<String, Integer> edades =
            Map.of("Luis",23,"Leo",28,"José",25);

        System.out.println(edades); // {Luis=23, Leo=28, José=25}

        System.out.println("Edad de José es: " + edades.get("José") );

        System.out.println("Edad de Ana es: " + edades.get("Ana") );

    }
} // El mapa edades es immutable
```

Los “mapas” más potentes que listas y conjuntos

```
package ejemplos;
import java.util.Map;
public class EdadesAmigos1 {
    public static void main(String... args) {
        Map<String, Integer> edades =
            Map.of("Luis",23,"Leo",28,"José",25);

        System.out.println(edades); // {Leo=28, José=25, Luis=23}

        System.out.println("Edad de José es: " + edades.get("José") );
                                   // Edad de José es: 25
        System.out.println("Edad de Ana es: " + edades.get("Ana") );
                                   // Edad de Ana es: null
    }
} // El mapa edades es inmutable
```


Consultar si existe clave, antes de obtener el valor

```
package ejemplos;
import java.util.Map;
public class EdadesAmigos2 {
    public static void main(String... args) {
        Map<String, Integer> edades =
            Map.of("Luis",23,"Leo",28,"José",25);

        System.out.println(edades);

        if (! edades.containsKey( "Ana" ))
            System.out.println( "Ana no tiene edad." );
        else System.out.println("Edad de Ana es: " + edades.get("Ana"));
            // Ana no tiene edad.
    }
} // El mapa edades es inmutable
```

Actualización de un valor en el mapa

```
package ejemplos;
import java.util.*;
public class EdadesAmigos3 {
    public static void main(String... args) {
        Map<String, Integer> edades = new HashMap<>();
        edades.put("Luis", 23);
        edades.put("Leo", 28);
        edades.put("José", 25);

        System.out.println(edades); // {Luis=23, Leo=28, José=25}
        // al cumplir años:
        edades.put("José", edades.get("José")+1 );
        System.out.println(edades); // {Luis=23, Leo=28, José=26}

    } // pero error si José no tiene edad
}
```

Contar frecuencia de palabras en argumentos

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

        for (String palabra : args) {
            if (frecuencia.containsKey(palabra))
                frecuencia.put( palabra, frecuencia.get(palabra)+1 );
            else frecuencia.put( palabra, 1 );
        }
        System.out.println(frecuencia);
        // Con args: la hora de la verdad es la hora de la muerte
        // salida: {de=2, verdad=1, la=4, hora=2, muerte=1, es=1}
    }
}
```

Contar frecuencia de palabras en una línea

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

        String linea="la hora de la verdad es la hora de la muerte";
        for (String palabra : linea.split("\\s+")) {
            if (frecuencia.containsKey(palabra))
                frecuencia.put( palabra, frecuencia.get(palabra)+1 );
            else frecuencia.put( palabra, 1 );
        }
        System.out.println(frecuencia);
        //
        // salida: {de=2, es=1, hora=2, la=4, muerte=1, verdad=1}
    }
}
```

TreeMap y TreeSet se autoordenan

pero no existe TreeList

```
package ejemplos;
import java.util.*;
public class OrdenarListaAmigos {
    public static void main(String... args) {
        List<String> amigos = new ArrayList<String>();
        List<String> tusAmigos = new LinkedList<String>();

        amigos.add("Luis"); amigos.add("Leo"); amigos.add("José");
        tusAmigos.add("Pi"); tusAmigos.add("Ed"); tusAmigos.add("Mar");

        Collections.sort(amigos);    Collections.sort(tusAmigos);

        System.out.println(amigos);    // [José, Leo, Luis]
        System.out.println(tusAmigos); // [Ed, Mar, Pi]
    }
}
```

Ejercicio

Crea una clase RecordsTable que permita almacenar los tiempos asociados a pruebas atléticas, de tal manera que el siguiente programa de la salida de más abajo:

```
public class AthleticsRecords {  
    public static void main(String[] args) {  
        RecordsTable menRecords = new RecordsTable("Athletics world records");  
        menRecords.set("100 meters", 9.58);  
        menRecords.set("400 meters", 43.03);  
        menRecords.set("200 meters", 19.19);  
        if (!menRecords.set("50 meters", -0.3))  
            System.out.println("Invalid value!");  
        System.out.println(menRecords); // orden alfabético de pruebas  
    }  
}
```

Invalid value!

Athletics world records: {100 meters=9.58, 200 meters=19.19, 400 meters=43.03}

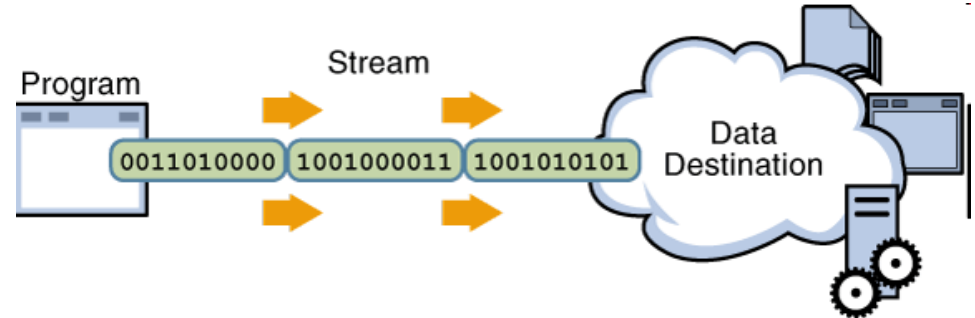
Proceso de archivos de texto

- Toda entrada/salida debe tener en cuenta excepciones
- Usamos paquete predefinido `java.io`
- Usamos una jerarquía de objetos desde el más bajo (byte) hasta los más altos (líneas, objetos)
- *InputStream* lee bytes
- *Reader* lee caracteres (2 bytes), *Writer* los escribe
- `BufferedReader` forma líneas con los caracteres leídos mediante un `InputStreamReader` (que a su vez necesita un `FileInputStream` para leer byte a byte)
- A veces también usamos `java.nio` (**nueva io**)
- En Eclipse los nombres de archivos (de datos, no los fuentes) son relativos al directorio del proyecto Eclipse

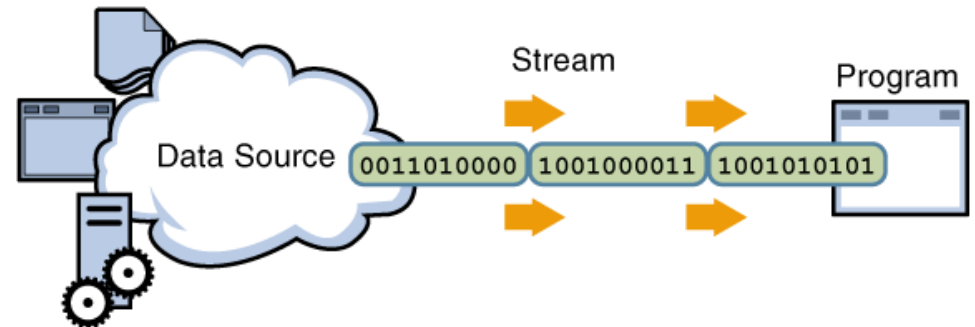
Streams de I/O (entrada/salida)

- Los **streams de I/O** representan fuentes o destinos de datos

- Encapsulan el tipo de fuente:
 - ☐ fichero en disco
 - ☐ otra aplicación
 - ☐ un dispositivo
 - ☐ un array en memoria
 - ☐ un puerto de comunicación
 - ☐ etc.



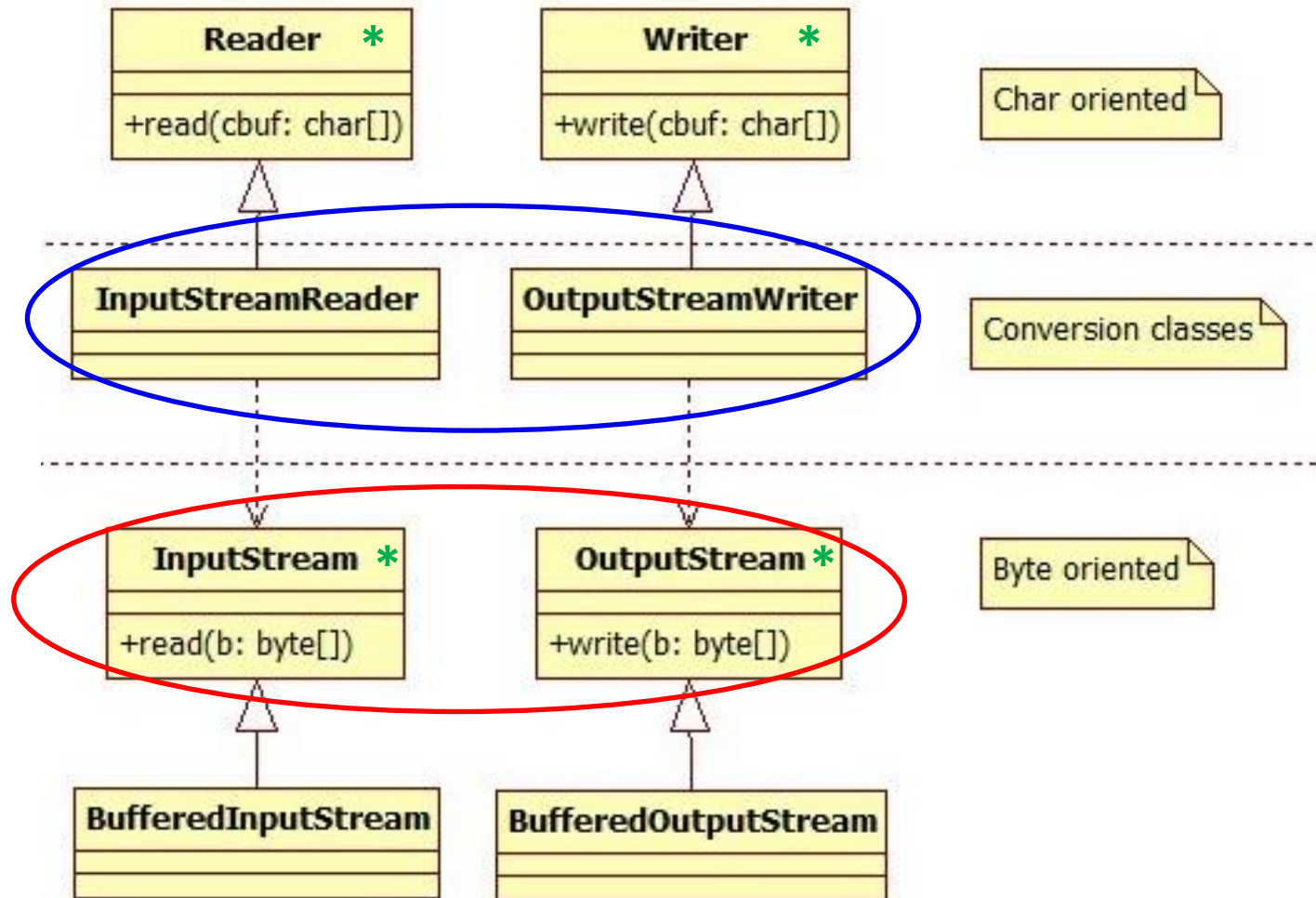
Output Stream



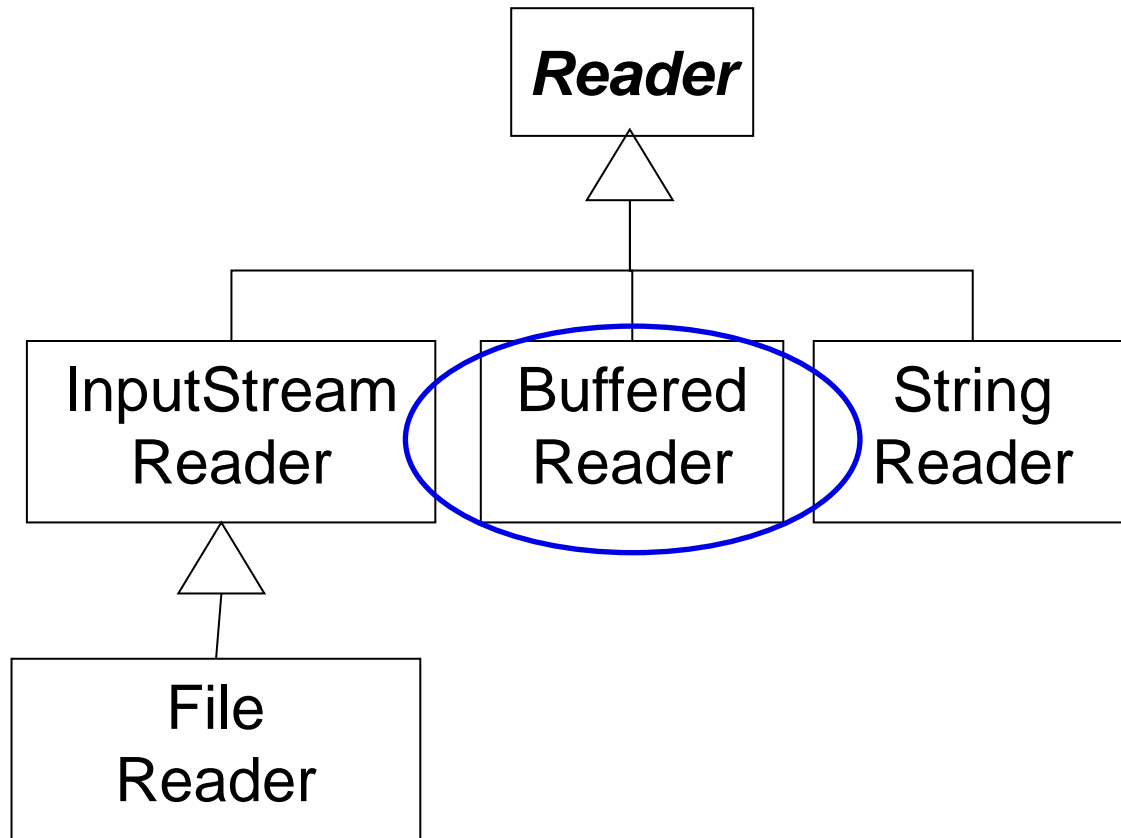
Input Stream

Clases Básicas I/O

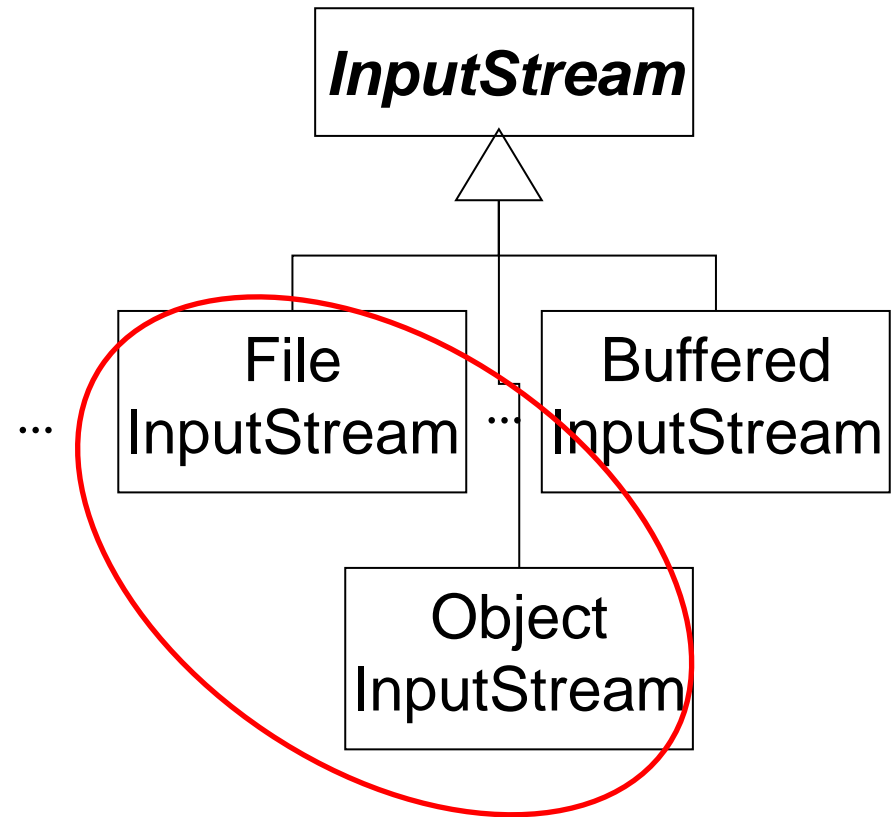
(*) Clases abstractas



Clases Básicas I/O



Lectura de caracteres



Lectura de bytes

Leer y procesar un archivo de texto (falta algo)

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

        String linea;
        while ((linea = buffer.readLine()) != null) {
            System.out.println( "Linea leida: " + linea);
        }

        buffer.close();
    }
}
```

Leer y procesar un archivo de texto (completo)

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

        String linea;
        while ((linea = buffer.readLine()) != null) {
            System.out.println( "Linea leida: " + linea);
        }

        buffer.close();
    }
}
```

Leer y procesar un archivo de texto (más simple)

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

        String linea;
        while ((linea = buffer.readLine()) != null) {
            System.out.println( "Linea leida: " + linea);
        }

        buffer.close();
    }
}
```

Frecuencia de palabras leídas de archivo de texto

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

Frecuencia de palabras leídas de archivo de texto

```
package ejemplos;
import java.io.IOException; import java.nio.file.*;import java.util.*;
public class FrecuenciaPalabrasArchivo2 {
    public static void main(String[] args) throws IOException {

        List<String> lineas = Files.readAllLines(Paths.get("texto.txt"));

        Map<String, Integer> frecuencia = new TreeMap<>();
        for (String linea : lineas){
            for (String palabra : linea.split(" "))
                if (frecuencia.containsKey(palabra))
                    frecuencia.put(palabra, frecuencia.get(palabra)+1 );
                else frecuencia.put(palabra, 1 );
        }
        System.out.println(frecuencia);
    }
}
```

Salida en archivos de texto con formato

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

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

        FileOutputStream stream = new FileOutputStream("numeros.txt");
        PrintWriter salida = new PrintWriter(stream);

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


Persistencia (entrada/salida) de objetos

- Con un `FileOutputStream` creamos el `ObjectOutputStream` en el que se guardan los objetos en formato “binario”
- El mismo archivo se trata ahora como `FileInputStream`
- Con un `FileInputStream` creamos el `ObjectInputStream`
- Desde él leemos los mismos objetos que se guardaron, pero se leen como objetos (es necesario un *casting*).
- Es responsabilidad del programador saber qué objetos guardó (en qué orden) y leerlos del mismo modo
- Cada objeto leído se asigna (con un casting) a una variable del tipo de dato adecuado para ese objeto
- En caso contrario, error `ClassCastException`
- **La clase de los objetos debe ser `Serializable`**

Ejemplo de clase pública: Punto

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

```
public class Punto /* Falta algo, no es Serializable */ {  
    private int x, y; // componentes privados  
  
    public Punto(int x, int y) { // constructor  
        this.x = x;  
        this.y = y;  
    }  
  
    @Override  
    public String toString( ) { return "(" + x + "," + y + ")"; }  
  
    // sin el siguiente método serían puntos inmutables  
    public void desplazar(int dx, int dy) { x += dx; y += dy; }  
}
```

Clase Punto usada en clase pública: Poligono

```
package es.uam.eps.ads.geometria;  
import java.util.*; // no hay import es.uam.eps.geometria.Punto;
```

```
public class Poligono /* Falta algo */ {  
    private List<Punto> puntos;  
    public Poligono() { puntos = new ArrayList<>(); }  
  
    public Poligono add(Punto p) { puntos.add(p); return this; }  
  
    public String toString() {  
        String resultado = "<";  
        for (Punto p : puntos) { resultado += p; }; // p.toString();  
        return resultado + ">";  
    }  
    public void desplazar(int dx, int dy) {  
        for (Punto p : puntos) { p.desplazar(dx, dy); };  
    }  
}
```

Persistencia de objetos de clase Punto

```
package ejemplos.persistencia;
import java.io.*;    import java.util.*;
import es.uam.eps.ads.geometria.Punto;

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

        ObjectOutputStream salidaObjetos =
            new ObjectOutputStream(
                new FileOutputStream( "puntos.objectData" ) );

        List<Punto> puntos =
            new LinkedList<>(Arrays.asList(
                new Punto(3,4), new Punto(0,3), new Punto(2,6) ));

        salidaObjetos.writeObject(puntos);
        salidaObjetos.close();
    }
}
```

Clase Punto con persistencia (serializada)

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

Faltaba añadir que la
clase es *serializable*

```
public class Punto implements Serializable {  
    private int x, y; // componentes privados
```

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

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

```
// sin el siguiente método serían puntos inmutables
```

```
public void desplazar(int dx, int dy) { x += dx; y += dy; }  
}
```

Lectura de objetos Punto serializados en archivo

```
package ejemplos.persistencia;
import java.io.*;    import java.util.*;
import es.uam.eps.ads.geometria.Punto; // importante

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

        ObjectInputStream entradaObjetos =
            new ObjectInputStream(
                new FileInputStream( "puntos.objectData" ) );

        List<Punto> puntos = (List<Punto>) entradaObjetos.readObject();

        entradaObjetos.close();
        System.out.println("Leido: " + puntos);
    }
}
```

Lectura de objetos Punto serializados en archivo

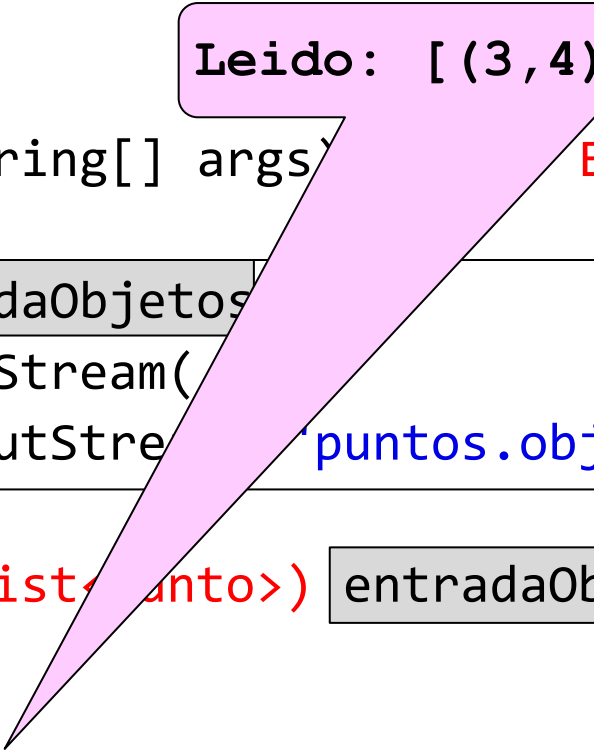
```
package ejemplos.persistencia;
import java.io.*;    import java.util.*;
import es.uam.eps.ads.geometria.Punto; // importante

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

        ObjectInputStream entradaObjetos =
            new ObjectInputStream(
                new FileInputStream("puntos.objectData" ) );

        List<Punto> puntos = (List<Punto>) entradaObjetos.readObject();

        entradaObjetos.close();
        System.out.println("Leido: " + puntos);
    }
}
```



Leido: [(3,4), (0,3), (2,6)]

Lectura de objetos Punto serializados en archivo

```
package ejemplos.persistencia;  
import java.io.*;    import java.util.*;  
import es.uam.eps.ads.geometria.Punto; // importante
```

Precaución: Modificar el contenido de la clase serializada (y su recompilación) *puede* hacer que los objetos previamente serializados ya no sean legibles con la nueva clase.

```
ObjectInputStream entradaObjetos  
    new ObjectInputStream(  
        new FileInputStream("puntos.objectData" ) );
```

```
List<Punto> puntos = (List<Punto>) entradaObjetos.readObject();
```

```
entradaObjetos.close();
```

```
System.out.println("Leido: " + puntos);
```

```
}
```

```
}
```


Persistencia (entrada/salida) de objetos

- Esta forma de serialización es muy simple
- Tiene limitaciones:
 - Si cambia la clase del objeto serializado,
 - el objeto serializado probablemente será ilegible
 - mediante la nueva clase
- Sin duda, es mejor que hacerlo con conversión a texto
- Otras formas de serialización:
 - JavaBeans, XML, JSON, ...

Otro ejemplo: persistencia de clase Poligono (1/2)

```
package ejemplos.persistencia;  
import java.io.*;    import java.util.*;  
import es.uam.eps.ads.geometria.*;  
  
public class PersistenciaPoligonos {  
    public static void main(String[] args) throws IOException {
```

```
        ObjectOutputStream salidaLineas =  
            new ObjectOutputStream(  
                new FileOutputStream( "poligonos.objectData" ));
```

```
        Poligono segmento = new Poligono();  
        Poligono cuadrado = new Poligono();
```

```
        // sigue ...
```

Otro ejemplo: persistencia de clase Poligono (2/2)

```
segmento.add(new Punto(1,1)).add(new Punto(3,2));
```

```
cuadrado.add(new Punto(0,0)).add(new Punto(0,2))  
          .add(new Punto(2,2)).add(new Punto(0,2))  
          .add(new Punto(0,0));
```

```
System.out.println( segmento );
```

```
System.out.println( cuadrado );
```

```
salidaLineas.writeObject(segmento);
```

```
salidaLineas.writeObject(cuadrado);
```

```
salidaLineas.close();
```

```
}
```

```
}
```

Clase Poligono con persistencia (serializada)

```
package es.uam.eps.ads.geometria;
import java.util.*; // no hay import es.uam.eps.geometria.Punto;
import java.io.Serializable;
public class Poligono implements Serializable {
    private List<Punto> puntos;
    public Poligono() { puntos = new ArrayList<>(); }

    public Poligono add(Punto p) { puntos.add(p); return this; }

    public String toString() {
        String resultado = "<";
        for (Punto p : puntos) { resultado += p; };
        return resultado + ">";
    }
    public void desplazar(int dx, int dy) {
        for (Punto p : puntos) { p.desplazar(dx, dy); };
    }
}
```

Leer objetos Poligono serializados en archivo

```
package ejemplos.persistencia;
import java.io.*;    import java.util.*;
import es.uam.eps.ads.geometria.Poligono;

public class LeerPoligonos {
    public static void main(String[] args) throws Exception {
        ObjectInputStream entradaObjetos =
            new ObjectInputStream(
                new FileInputStream( "poligonos.objectData" ) );
        Poligono p1 = (Poligono) entradaObjetos.readObject();
        Poligono p2 = (Poligono) entradaObjetos.readObject();
        entradaObjetos.close();
        System.out.println("Segmento: " + p1);
        System.out.println("Cuadrado: " + p2);
        p2.desplazar(10,100);
        System.out.println("Cuadrado desplazado: " + p2);
    }
}
```

Leer objetos Poligono serializados en archivo

Segmento: $\langle (1,1) (3,2) \rangle$

Cuadrado: $\langle (0,0) (0,2) (2,2) (0,2) (0,0) \rangle$

Cuadrado desplazado: $\langle (10,100) (10,102) (12,102) (10,102) (10,100) \rangle$

```
public static void main(String[] args) throws Exception {  
    ObjectInputStream entradaObjetos =  
        new ObjectInputStream(  
            new FileInputStream("poligonos.objectData" ) );  
    Poligono p1 = (Poligono) entradaObjetos.readObject();  
    Poligono p2 = (Poligono) entradaObjetos.readObject();  
    entradaObjetos.close();  
    System.out.println("Segmento: " + p1);  
    System.out.println("Cuadrado: " + p2);  
    p2.desplazar(10,100);  
    System.out.println("Cuadrado desplazado: " + p2);  
}
```