Excepciones y E/S

Java y Servicios Web I
Master en Ingeniería Matemática

Manuel Montenegro
Dpto. Sistemas Informáticos y Computación

Desp. 467 (Mat)

montenegro@fdi.ucm.es

Contenidos

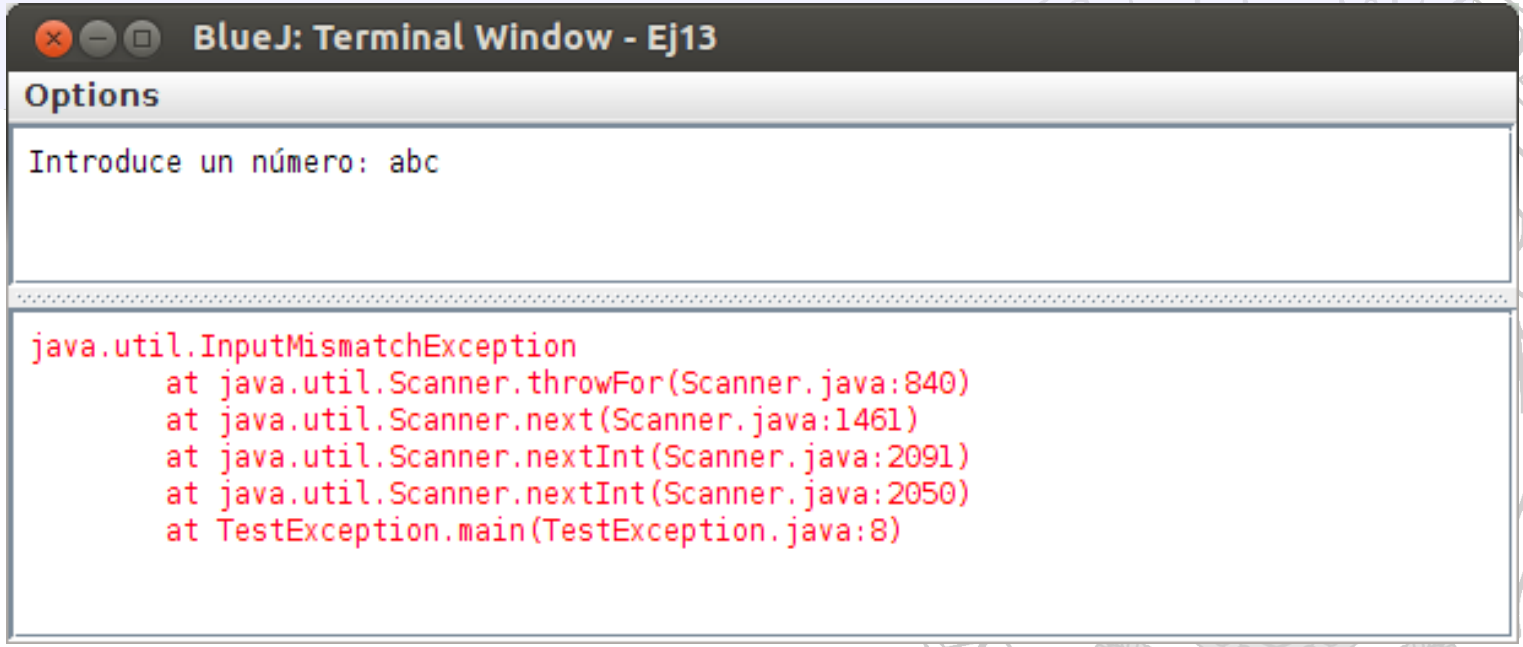
- Generación de excepciones.
- Captura de excepciones.
- Generación de excepciones.
- Excepciones definidas por el programador.
- Bloques try/catch/finally.
- Entrada/Salida en Java.
- Serialización.

Excepciones

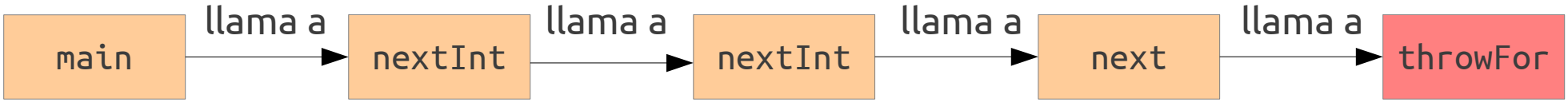
- Una excepción es una situación anómala que ocurre en tiempo de ejecución.
  - Intentar abrir un archivo que no existe.
  - Intentar crear un archivo sin que haya espacio suficiente en el disco.
  - Intentar leer un número del teclado, cuando el usuario ha introducido algo que no es un número.
  - Intento de acceder a un array más allá de sus límites.
  - etc.
- Java permite al programador tomar el control del programa en presencia de estas situaciones anómalas → capturar una excepción.

Generación de excepciones

```
public class TestException
{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduce un número: ");
        int a = sc.nextInt();
        System.out.println("El número que has introducido es " + a);
    }
}
```

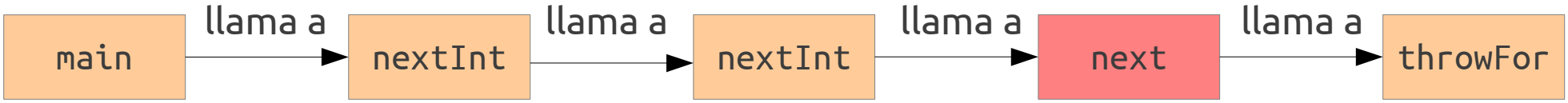


Generación de excepciones



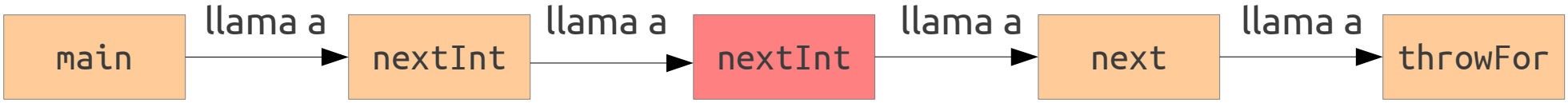
- `throwFor` detectó una situación anómala y decidió reportarlo a next mediante el lanzamiento de una excepción.

Generación de excepciones



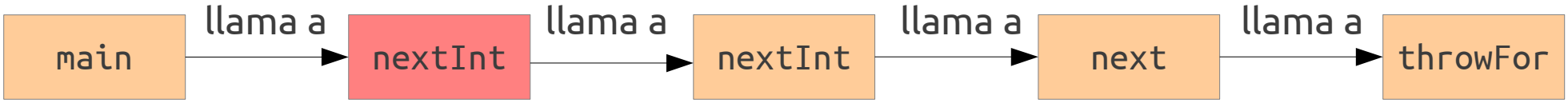
- Cuando `throwFor` lanza una excepción, su ejecución se interrumpe y vuelve al método que llamó a `throwFor` (es decir, `next`).
- En la función `next` no se captura la excepción.

Generación de excepciones



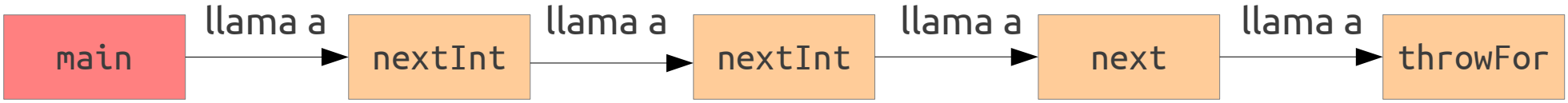
- Como en `next` no se captura la excepción, la ejecución de next se interrumpe y la ejecución vuelve a `nextInt`.

Generación de excepciones



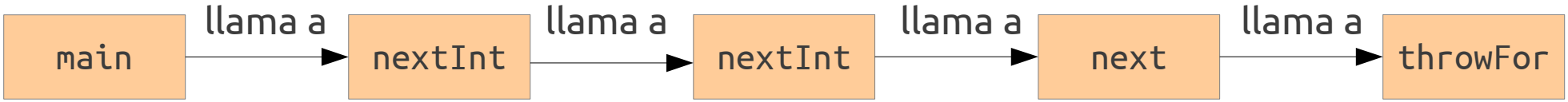
- En `nextInt` tampoco se captura la excepción, y la ejecución vuelve al llamante.

Generación de excepciones

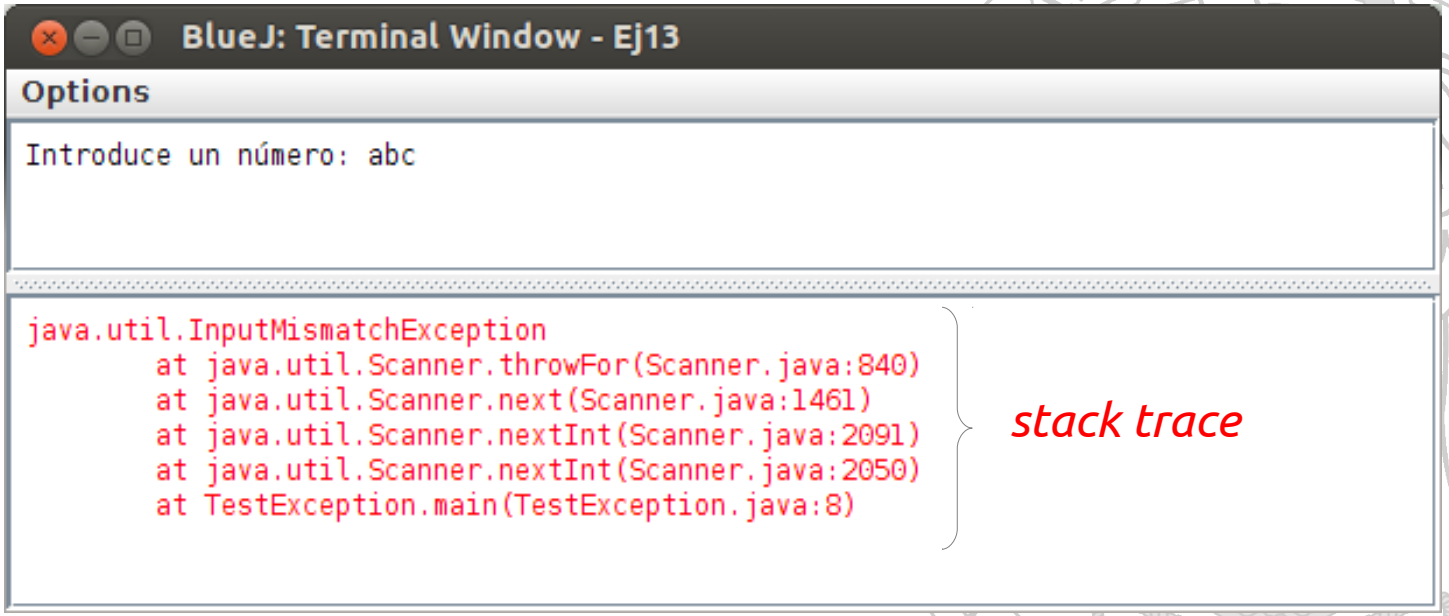


- La ejecución vuelve a nuestro método `main`. Como nosotros no capturamos la excepción, ésta se eleva hacia la máquina virtual de Java, que fue la encargada de ejecutar `main`.

Generación de excepciones



- La máquina virtual de Java, al recibir una excepción, aborta la ejecución del programa, y muestra por pantalla el siguiente mensaje de error:

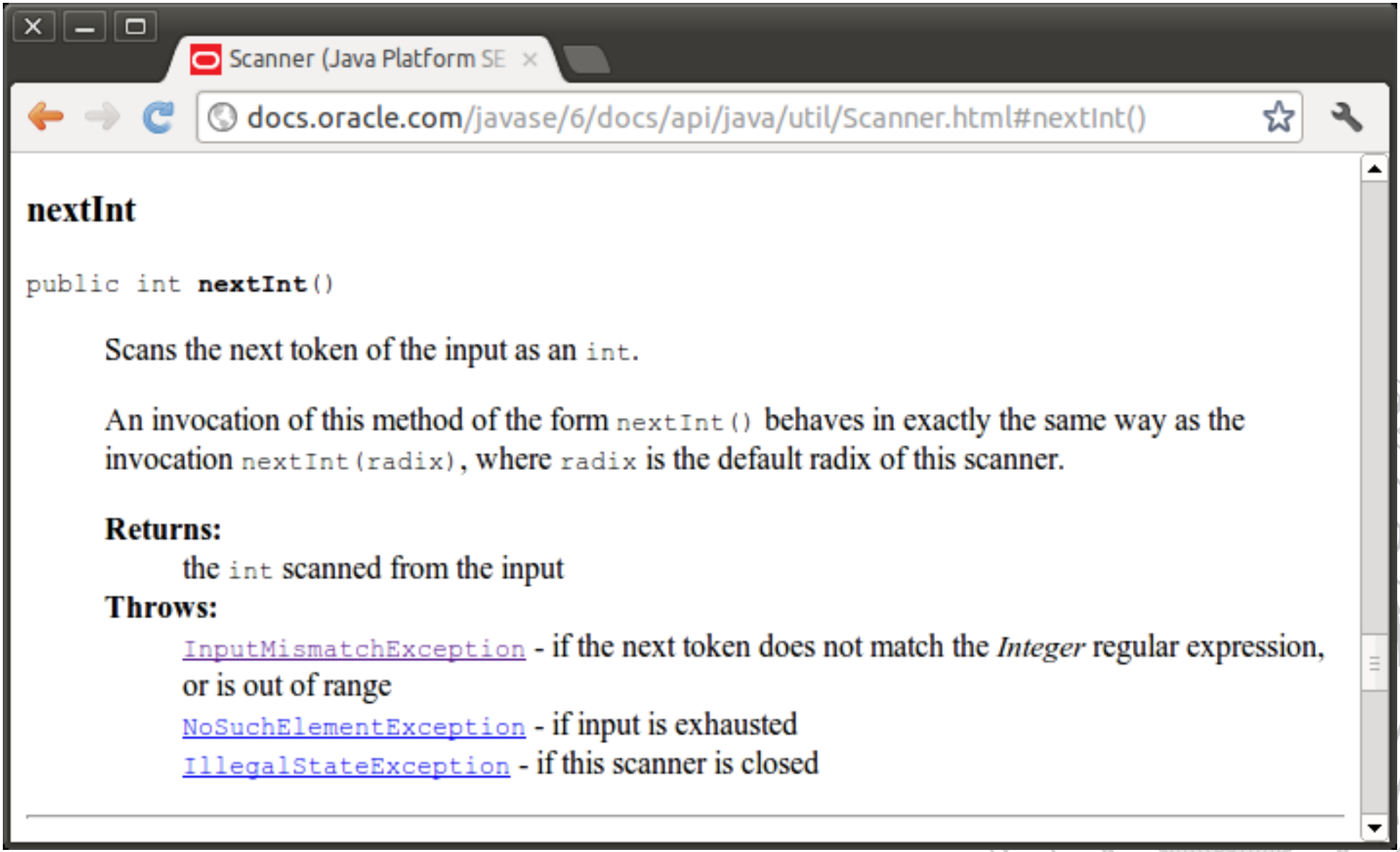


Clases de excepciones

- El mensaje de error generado por la JVM no sólo muestra la secuencia de llamadas en las que se produce la excepción.
- También muestra el tipo de excepción que se ha generado.

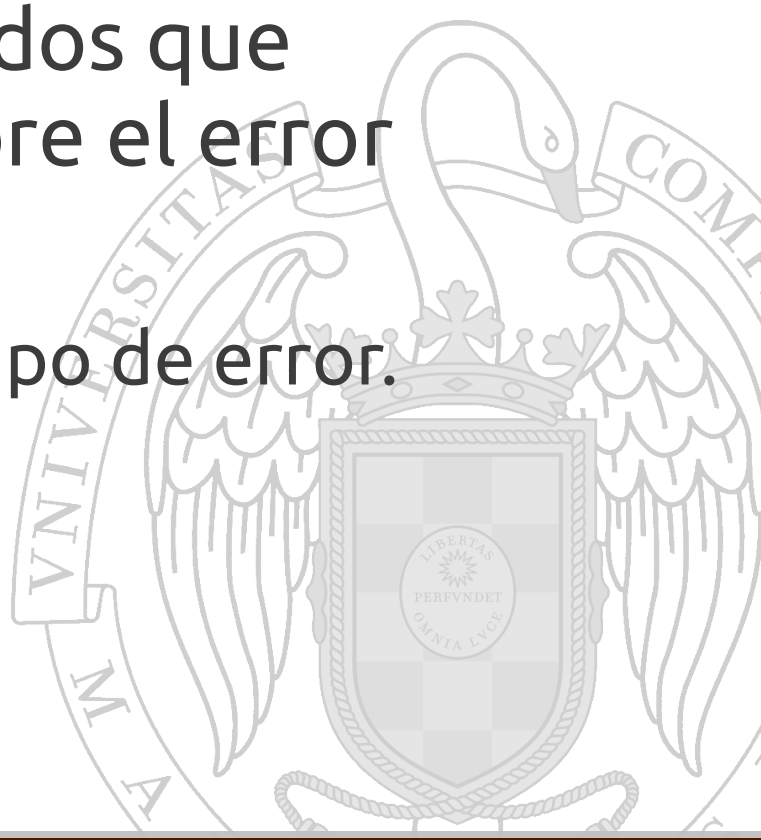
```
java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:840)
        at java.util.Scanner.next(Scanner.java:1461)
        at java.util.Scanner.nextInt(Scanner.java:2091)
        at java.util.Scanner.nextInt(Scanner.java:2050)
        at TestException.main(TestException.java:8)
```

Clases de excepciones



Clases de excepciones

- Las excepciones en Java son objetos.
- La información mostrada por la JVM es la clase a la que pertenece la excepción lanzada.
- Las excepciones poseen métodos que proporcionan información sobre el error producido.
  - Esta información depende del tipo de error.



Contenidos

- Generación de excepciones.
- Captura de excepciones.
- Generación de excepciones.
- Excepciones definidas por el programador.
- Bloques try/catch/finally.
- Entrada/Salida en Java.
- Serialización.

Captura de excepciones

- Una excepción puede capturarse mediante la construcción `try-catch`.

```
public class TestException {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        try {
           System.out.print("Introduce un número: ");
           int a = sc.nextInt();
           System.out.println("El número que has introducido es " + a);
        } catch (InputMismatchException e) {
           System.out.println("La cadena introducida no es un número.");
           System.out.println(e.getMessage());
        }
    }
}
```

Captura de excepciones

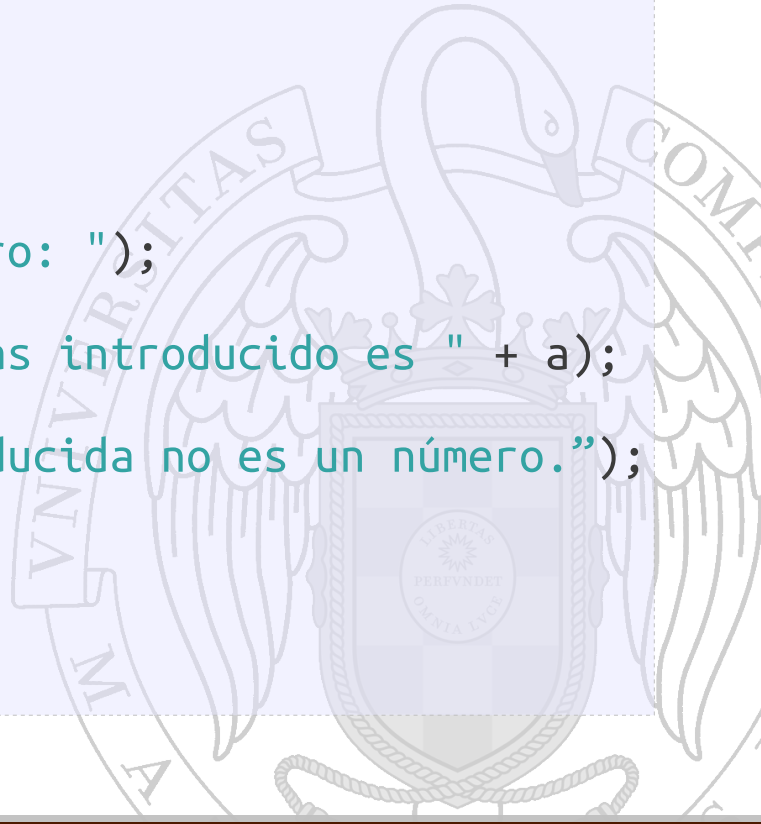
- Dentro del bloque `try` se coloca el código que es susceptible de generar una excepción.

```
public class TestException {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        try {
           System.out.print("Introduce un número: ");
           int a = sc.nextInt();
           System.out.println("El número que has introducido es " + a);
        } catch (InputMismatchException e) {
           System.out.println("La cadena introducida no es un número.");
           System.out.println(e.getMessage());
        }
    }
}
```

Captura de excepciones

- Si el código dentro del bloque `try` produce una excepción, se abandona dicho bloque y se comprueba si la clase de la misma se corresponde con (o es subclase de) la clase indicada tras el `catch`.

```
public class TestException {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        try {
           System.out.print("Introduce un número: ");
           int a = sc.nextInt();
           System.out.println("El número que has introducido es " + a);
        } catch (InputMismatchException e) {
           System.out.println("La cadena introducida no es un número.");
           System.out.println(e.getMessage());
        }
    }
}
```



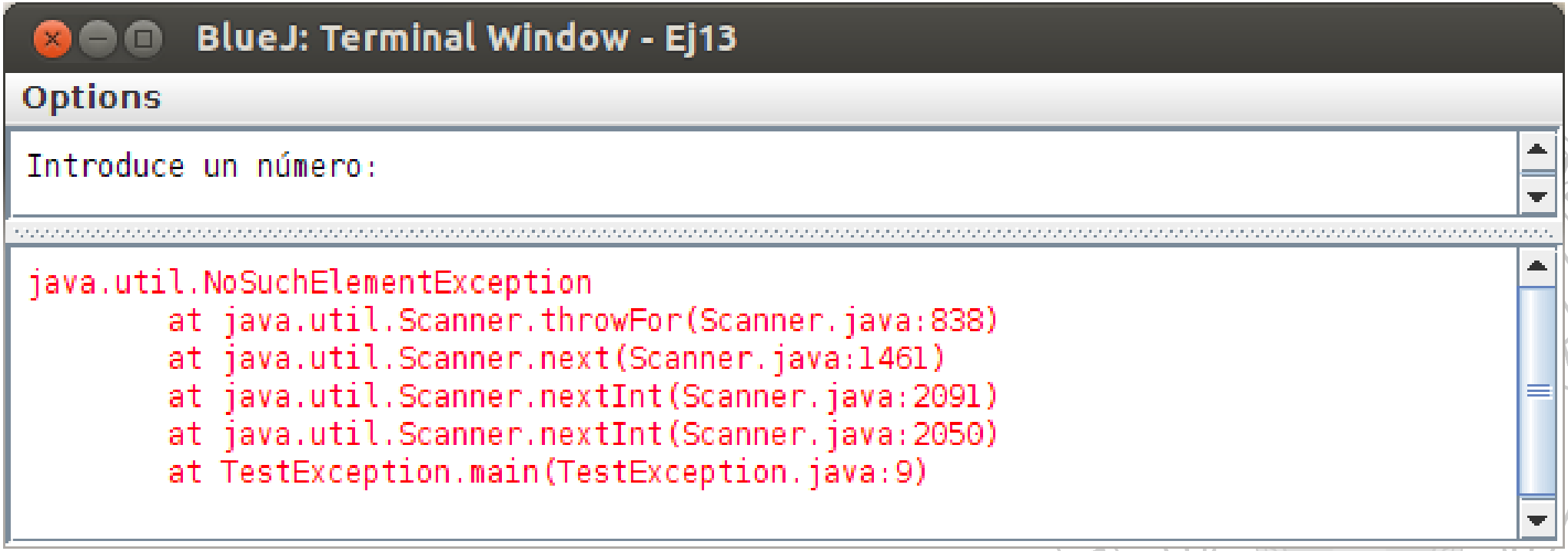
Captura de excepciones

- Si la excepción es compatible con el tipo indicado tras el `catch`, se ejecutará el bloque correspondiente.
- Además, la variable indicada tras el tipo contendrá el objeto que representa la excepción lanzada.

```
public class TestException {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        try {
           System.out.print("Introduce un número: ");
           int a = sc.nextInt();
           System.out.println("El número que has introducido es " + a);
        } catch (InputMismatchException e) {
           System.out.println("La cadena introducida no es un número.");
           System.out.println(e.getMessage());
        }
    }
}
```

Captura de excepciones

- Si la excepción no es compatible con el tipo indicado tras el `catch`, se volverá a lanzar al nivel superior (la JVM).

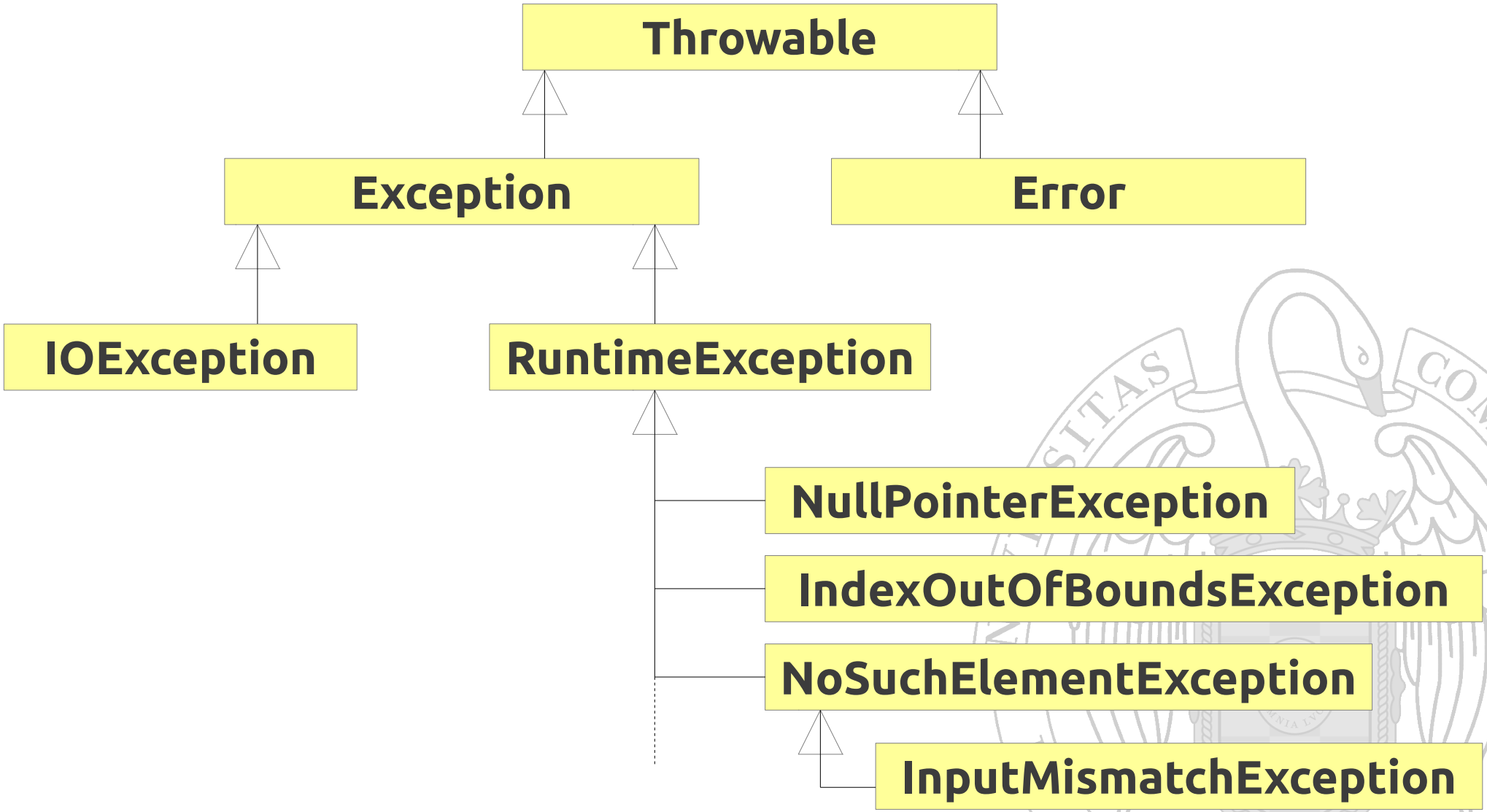


Captura de excepciones

- Es posible tener varias sentencias `catch` en un mismo bloque `try`, con el fin de capturar distintos tipos de excepciones.

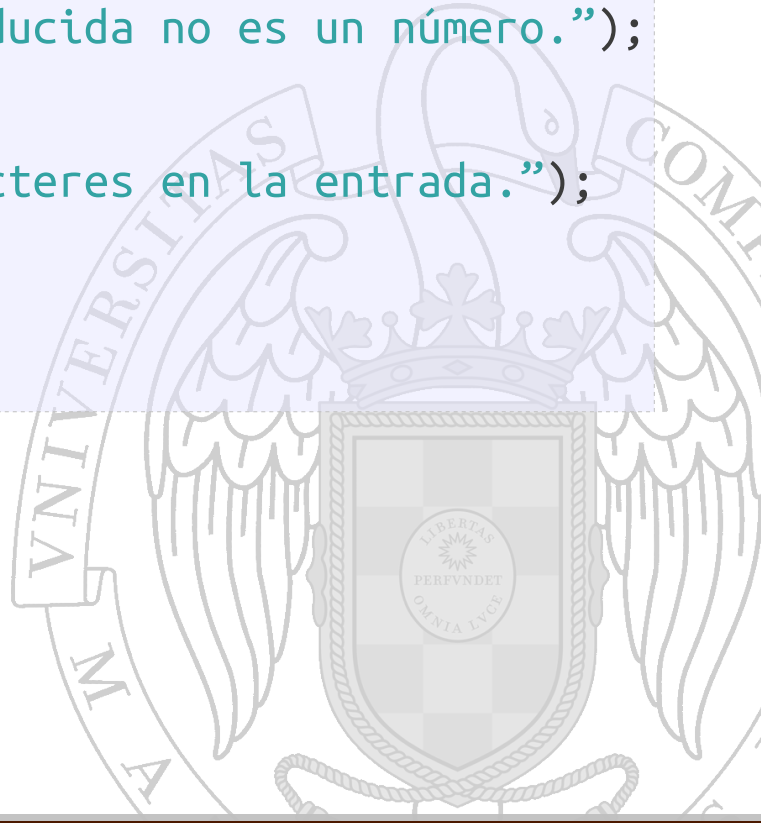
```
Scanner sc = new Scanner(System.in);

try {
   System.out.print("Introduce un número: ");
   int a = sc.nextInt();
   System.out.println("El número que has introducido es " + a);
} catch (InputMismatchException e) {
   System.out.println("La cadena introducida no es un número.");
   System.out.println(e.getMessage());
} catch (NoSuchElementException e) {
   System.out.println("No hay más caracteres en la entrada.");
   System.out.println(e.getMessage());
}
```

Jerarquía de excepciones



Jerarquía de excepciones

```
Scanner sc = new Scanner(System.in);

try {
   System.out.print("Introduce un número: ");
   int a = sc.nextInt();
   System.out.println("El número que has introducido es " + a);
} catch (InputMismatchException e) {
   System.out.println("La cadena introducida no es un número.");
   System.out.println(e.getMessage());
} catch (NoSuchElementException e) {
   System.out.println("No hay más caracteres en la entrada.");
   System.out.println(e.getMessage());
} catch (Exception e) {
   System.out.println(e.getMessage());
}
```



Contenidos

- Generación de excepciones.
- Captura de excepciones.
- Generación de excepciones.
- Excepciones definidas por el programador.
- Bloques try/catch/finally.
- Entrada/Salida en Java.
- Serialización.

Generar una excepción

```
public class Estudiante extends Persona {
   private int puntuacionTotal;
   private int numeroCalificaciones;

   …
   public double getNotaMedia() {
      return ((double) puntuacionTotal)/numeroCalificaciones;
   }
}
```

¿Y si numeroCalificaciones = 0?

- Para generar una excepción, hay que:
  - Crear el objeto que contenga la excepción (de alguna subclase de `Exception`).
  - Lanzar dicho objeto al llamante mediante `throw`.

Generar una excepción

```
public class Estudiante extends Persona {

  public double getNotaMedia() throws Exception {
    if (numeroCalificaciones == 0)
      throw new Exception("No hay calificaciones");
    return ((double) puntuacionTotal)/numeroCalificaciones;
  }
}
```

```
Estudiante est = new Estudiante("David Sánchez", new Fecha(10, 3, 2002), 23155512);
int notaMedia = 0;
try {
  notaMedia = est.getNotaMedia();
} catch (Exception e) {
  System.out.println(e.getMessage());
}
```

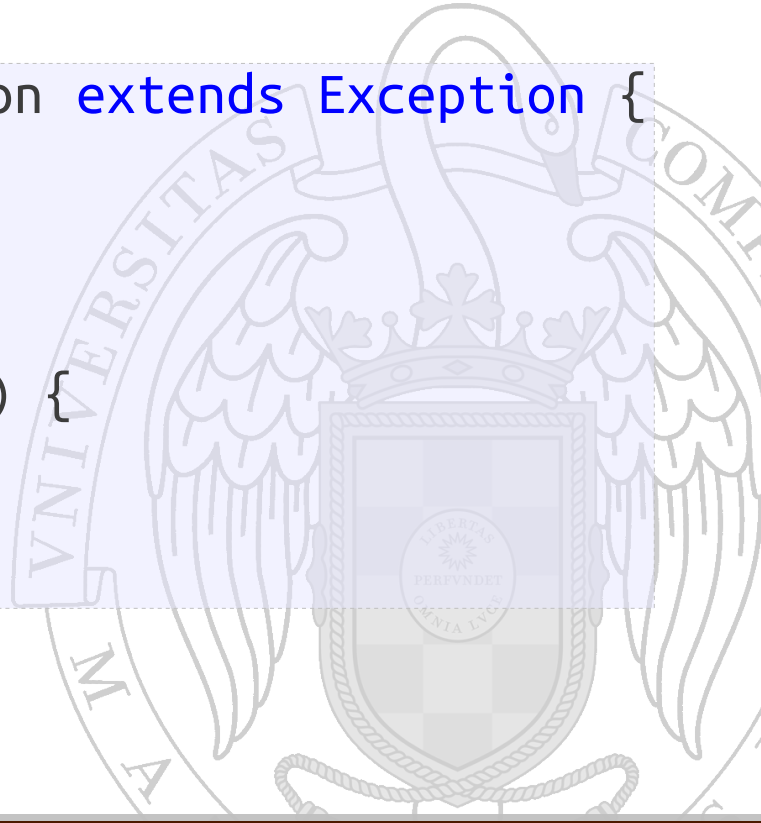
Contenidos

- Generación de excepciones.
- Captura de excepciones.
- Generación de excepciones.
- Excepciones definidas por el programador.
- Bloques try/catch/finally.
- Entrada/Salida en Java.
- Serialización.

Definir un tipo de excepción

- Si no hay ninguna subclase de `Exception` que se adapte a la excepción que queremos lanzar, podemos definir nuestra propia subclase.

```
public class NoHayCalificacionesException extends Exception {
  public NoHayCalificaciones() {
    super("No hay calificaciones");
  }

  public NoHayCalificaciones(String msg) {
    super(msg);
  }
}
```



Definir un tipo de excepción

```
public class Estudiante extends Persona {

  public double getNotaMedia() throws NoHayCalificacionesException {
    if (numeroCalificaciones == 0)
      throw new NoHayCalificacionesException();
    return ((double) puntuacionTotal)/numeroCalificaciones;
  }
}
```

```
Estudiante est = new Estudiante("David Sánchez", new Fecha(10, 3, 2002), 23155512);
int notaMedia = 0;
try {
  notaMedia = est.getNotaMedia();
} catch (NoHayCalificacionesException e) {
  System.out.println(e.getMessage());
}
```

Contenidos

- Generación de excepciones.
- Captura de excepciones.
- Generación de excepciones.
- Excepciones definidas por el programador.
- Bloques try/catch/finally.
- Entrada/Salida en Java.
- Serialización.

Bloques try/catch/finally

- Las operaciones en el bloque `finally` se realizarán siempre al abandonar el bloque try/catch, independientemente de si se lanza una excepción o no.
- Se utiliza para liberar recursos (cerrar archivos abiertos, por ejemplo)

```
try {
  …
} catch (IOException e) {
  …
} finally {
  // Esto siempre se ejecutará
}
```

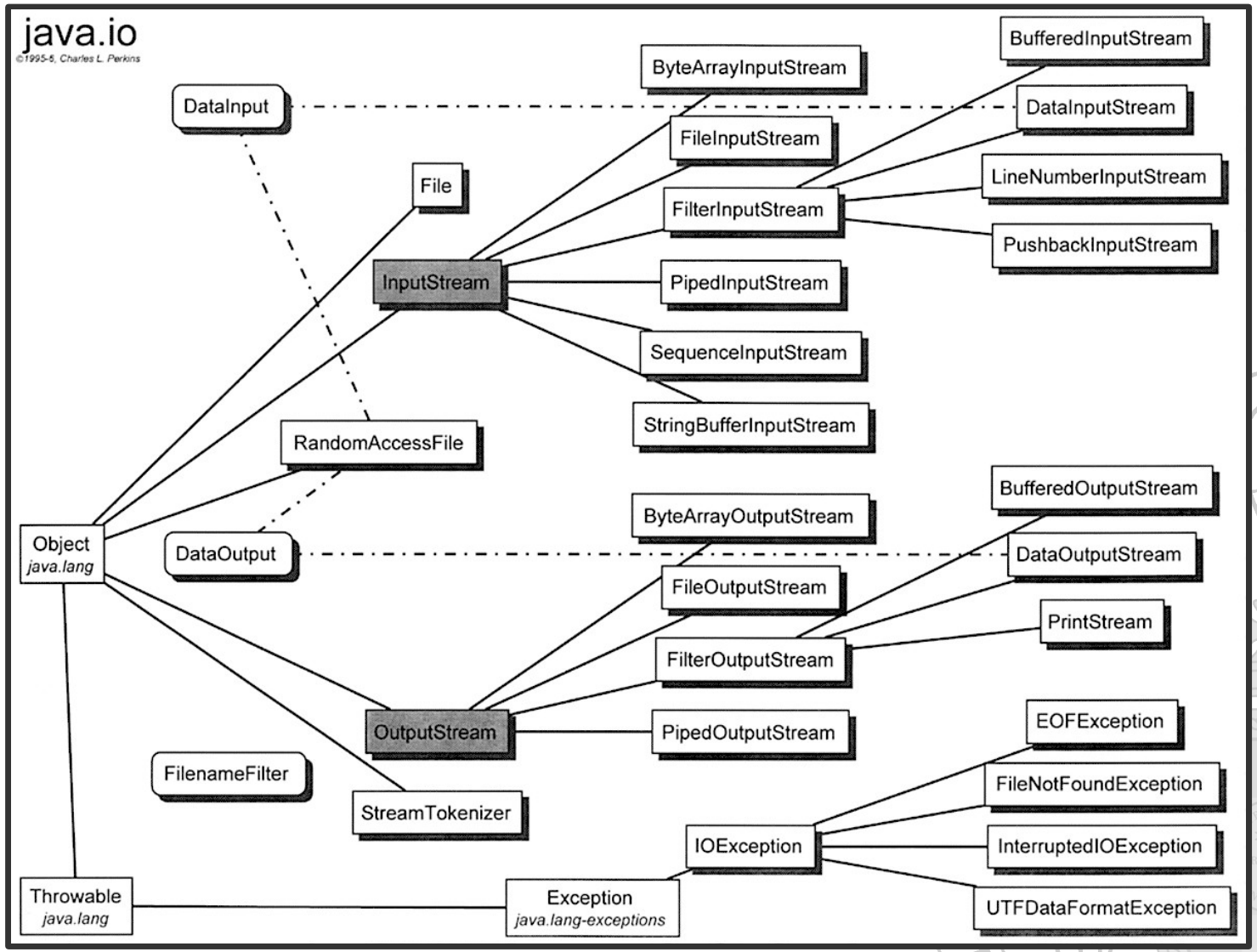
Contenidos

- Generación de excepciones.
- Captura de excepciones.
- Generación de excepciones.
- Excepciones definidas por el programador.
- Bloques try/catch/finally.
- Entrada/Salida en Java.
- Serialización.

Entrada/Salida en Java

- Las clases que realizan funciones de E/S se encuentran en el paquete `java.io`.
- Proporcionan clases que permiten abstraer el dispositivo a traves del cual se realiza la lectura o estructura.
  - Fichero.
  - Conexión de red.
  - etc.
- … y el modo en el que realiza.
  - Secuencial.
  - Acceso aleatorio.
  - Por líneas, palabras, carácter a carácter.

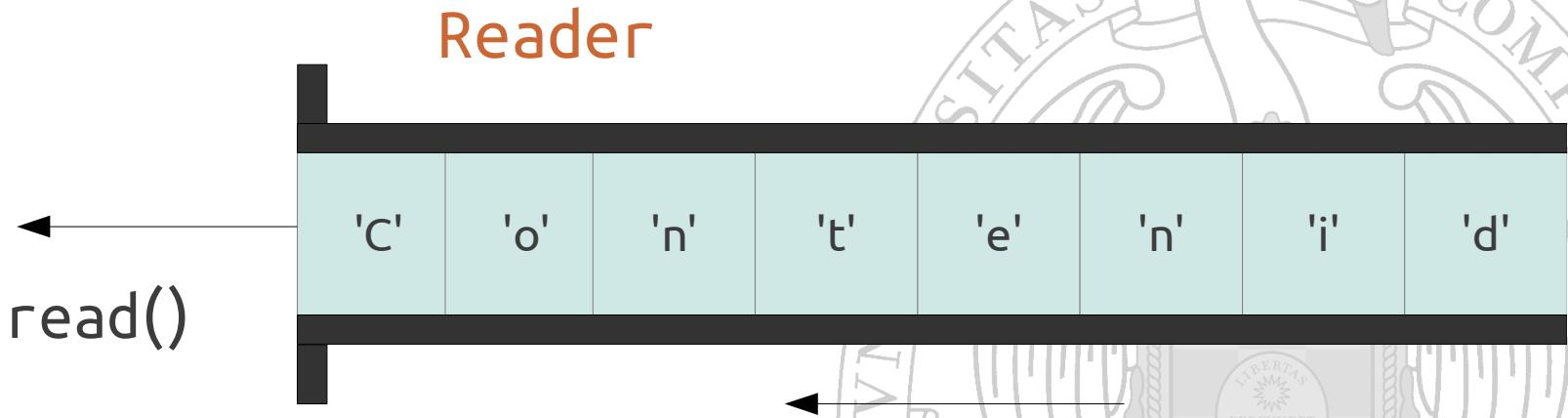
Entrada/Salida en Java



Tomado de: http://101.lv/learn/Java/ch30.htm

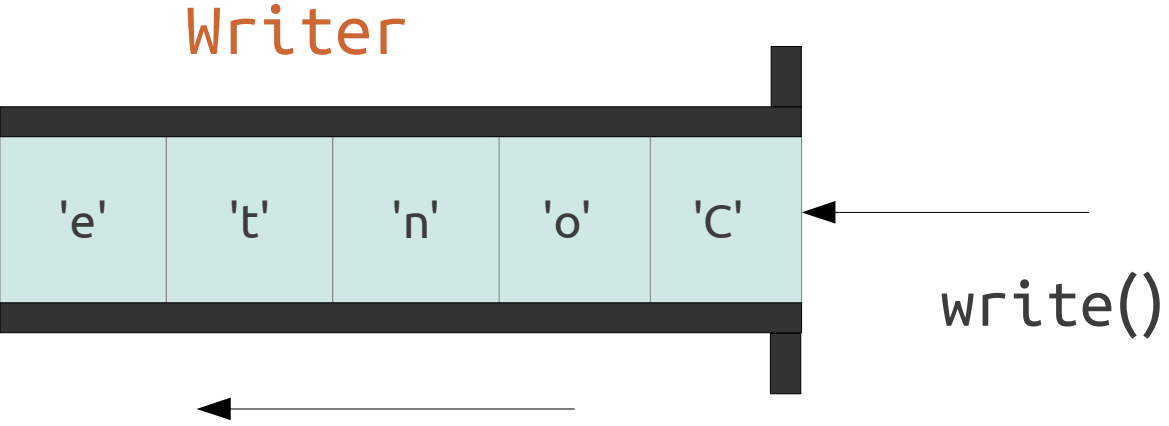
Flujos de Entrada/Salida

- La abstracción que proporcionan las operaciones de E/S se realiza mediante flujos.
- Un flujo es una secuencia de elementos que se procesa secuencialmente, de principio a fin.



Flujos de Entrada/Salida

- La abstracción que proporcionan las operaciones de E/S se realiza mediante flujos.
- Un flujo es una secuencia de elementos que se procesa secuencialmente, de principio a fin.



Flujos de Entrada

- `Reader` es una clase abstracta.
- Sus subclases concretan el dispositivo sobre el que se realiza la lectura.

Subclase	Dispositivo	Constructor
`FileReader`	Archivo	`new FileReader("archivo.txt")`
`CharArrayReader`	Array de caracteres	`char[] b = {'H', 'o', 'l', 'a'};` `new CharArrayReader(b)`
`StringReader`	Cadena de texto	`new StringReader("Cadena")`
`FilterReader`	Otro flujo	…

Flujos de Entrada

- Métodos de `Reader`:
  - `int read() throws IOException`
  - `void close() throws IOException`
- Las operaciones de entrada ofrecidas por `Reader` son muy primitivas.
- Mediante las subclases de `FilterReader` podemos ampliar el catálogo de operaciones.

Flujos de Entrada

- `Scanner` permite leer tipos de datos primitivos a partir de un flujo de texto
  - `boolean nextBoolean() throws …`
  - `byte nextByte() throws …`
  - `int nextInt() throws …`
  - …
- `BufferedReader` permite leer líneas completas.
  - `String readLine() throws IOException`

Flujos de Salida

- `Writer` es una clase abstracta.
- Sus subclases concretan el dispositivo sobre el que se realiza la escritura.

Subclase	Dispositivo	Constructor
`FileWriter`	Archivo	`new FileWriter("archivo.txt")`
`CharArrayWriter`	Array de caracteres	`char[] b = new char[40];` `new CharArrayReader(b)`
`StringWriter`	Cadena de texto	`new StringWriter()`
`FilterWriter`	Otro flujo	…

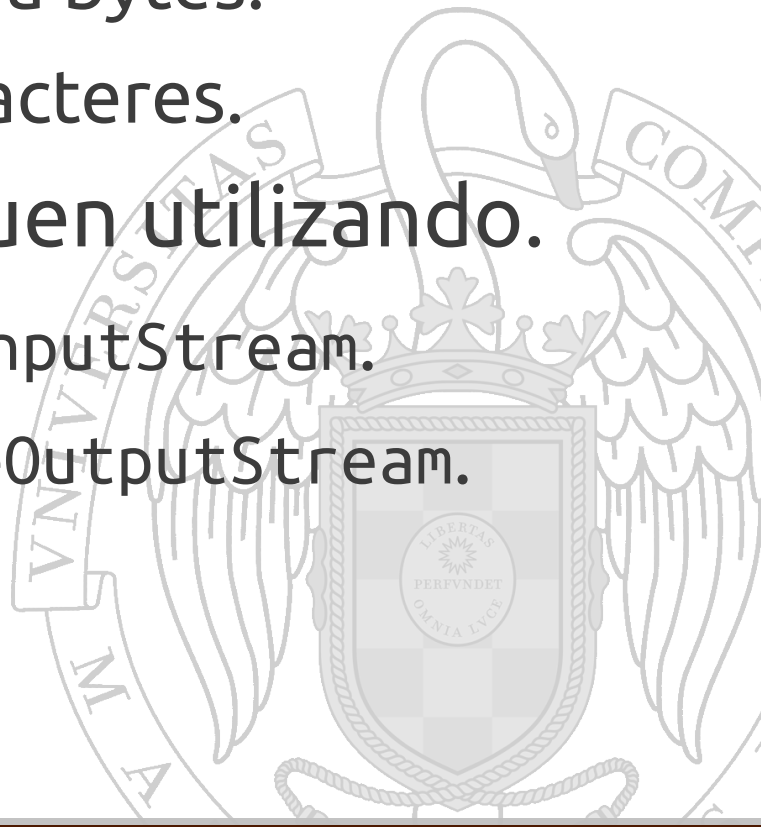
Flujos de Salida

- Métodos de `Writer`:
  - `Writer append(char c) throws IOException`
  - `void write(int c) throws IOException`
  - `void write(String s) throws IOException`
  - `void close() throws IOException`

- Mediante las subclases de `FilterWriter` podemos ampliar el catálogo de operaciones.
  - `PrintWriter`
  - `BufferedWriter`

InputStream y OutputStream

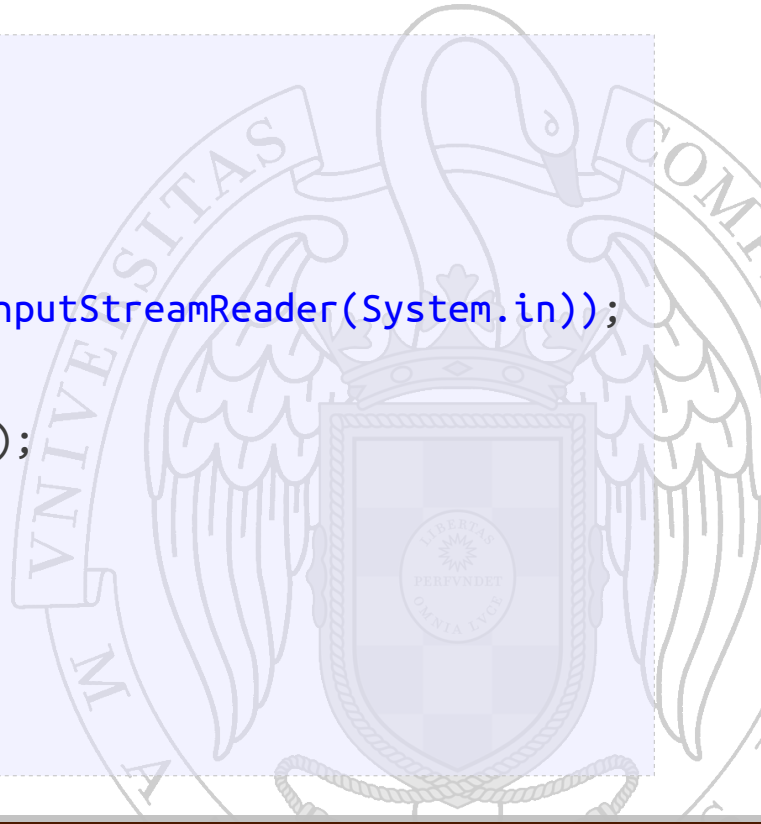
- En versiones de Java anteriores a la 1.1, la E/S se realizaba a través de las clases `InputStream` y `OutputStream`.
  - `InputStream`: Entrada orientada a bytes.
  - `Reader`: Entrada orientada a caracteres.
- Algunas de estas clases se siguen utilizando.
  - `System.in` es instancia de `FileInputStream`.
  - `System.out` es instancia de `FileOutputStream`.



InputStream y OutputStream

- Las clases `InputStreamReader` y `OutputStreamReader` permiten obtener un `Reader/Writer` a partir de un `InputStream/OutputStream`.

```
import java.io.*;
public class LecturaArchivos {
    public static void main(String[] args) {
        try {
            BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("¿Cual es tu nombre? ");
            String nombre = b.readLine();
            System.out.println("Tu nombre es " + nombre);
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```



Ejemplo

- Queremos leer de un archivo `datos.txt` con la siguiente información, que queremos representar en una ventana.

```
200 150 100
250 150 50
150 150 50
200 100 50
200 200 50
200 150 70
200 150 60
200 150 50
200 150 40
```

Centro

Radio

Ejemplo

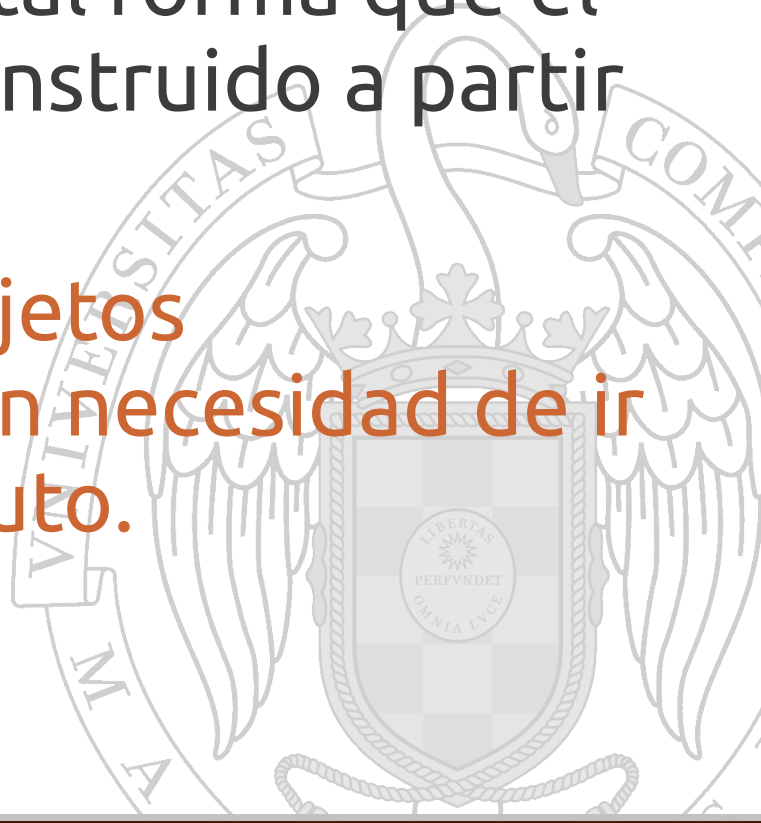
```
import java.io.*;
import java.util.*;

public class Circulos {

    public static void main(String[] args) {

        Ventana v = new Ventana();
        FileReader reader = null;
        try {
            reader = new FileReader("datos.txt");
            Scanner scan = new Scanner(reader);
            while (scan.hasNextInt()) {
                Punto centro = new Punto(scan.nextInt(), scan.nextInt());
                int radio = scan.nextInt();
                Circulo c = new Circulo(centro, radio);
                c.dibujar(v);
            }
            v.abrir();

        } catch (IOException e) {

            System.out.println("Error de entrada/salida: " + e.getMessage());

        } finally {

            try { if (reader != null) reader.close(); }
            catch (IOException e) { System.out.println("Error al cerrar el archivo"); }

        }

    }

}
```

Contenidos

- Generación de excepciones.
- Captura de excepciones.
- Generación de excepciones.
- Excepciones definidas por el programador.
- Bloques try/catch/finally.
- Entrada/Salida en Java.
- Serialización.

Serialización

- Permite convertir cualquier objeto en una secuencia de bytes, para que pueda ser escrita en un archivo.
- La transformación se hace de tal forma que el objeto original puede ser reconstruido a partir de esa secuencia de bytes.
- Esto permite leer y escribir objetos directamente en un archivo, sin necesidad de ir escribiendo atributo por atributo.



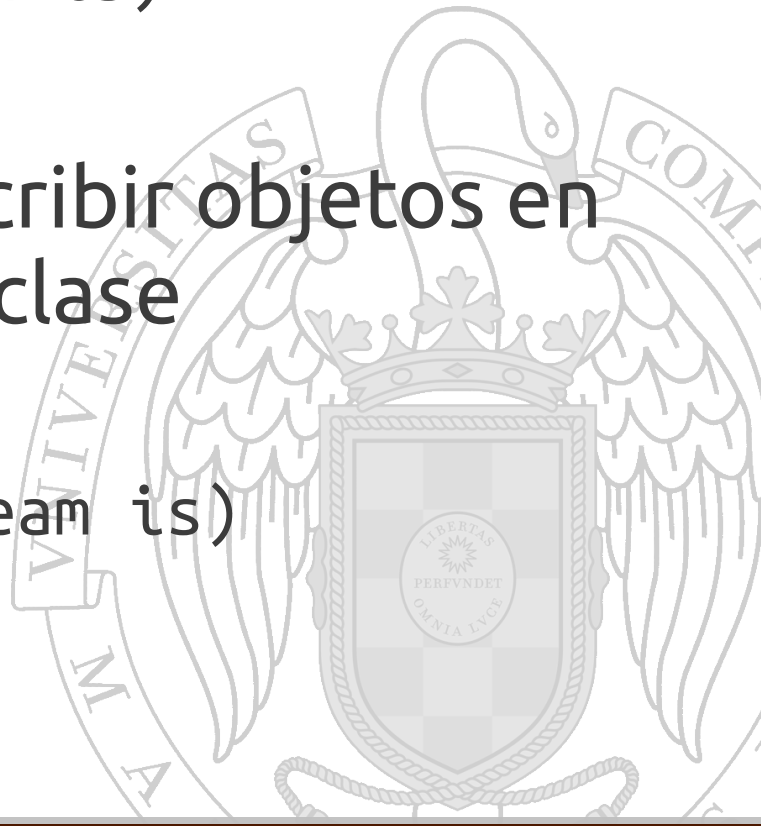
Serialización

- Los objetos susceptibles de serializar han de implementar la interfaz `Serializable`
  - No hace falta implementar ningún método. La interfaz es sólo un indicador de que el objeto se puede serializar.

```
public class Persona implements Serializable {

    …

}
```

Serialización

- Un objeto serializable puede ser leído de un flujo de entrada mediante la clase `ObjectInputStream`.
  - `ObjectInputStream(InputStream is)`
  - `Object readObject()`
- De modo similar, podemos escribir objetos en un flujo de salida mediante la clase `ObjectOutputStream`.
  - `ObjectOutputStream(OutputStream is)`
  - `void writeObject(Object o)`



Escritura de objetos

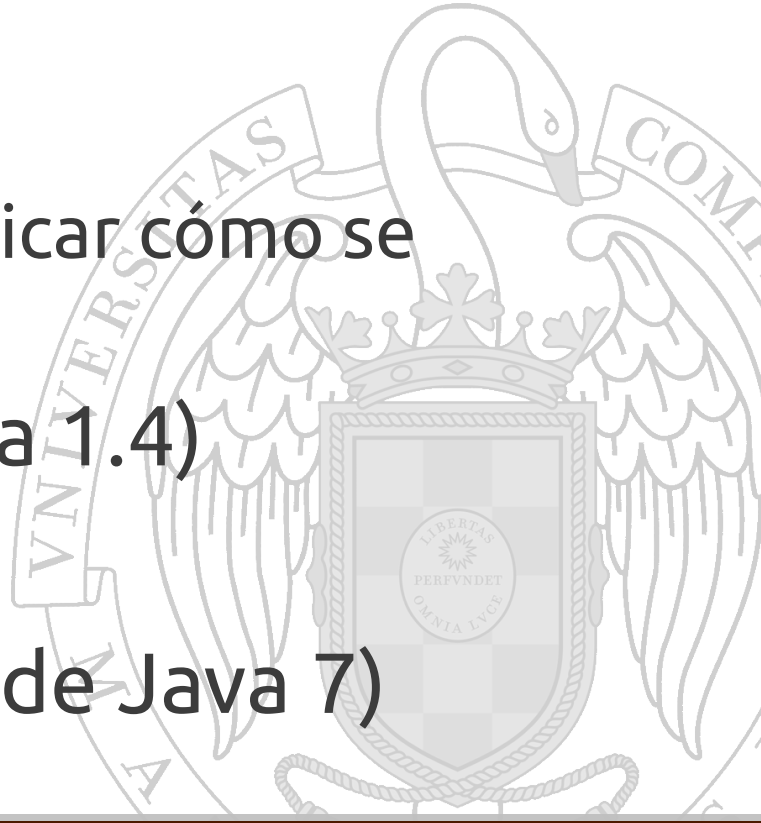
```
import java.io.*;

public class TestEmpleadoEscritura {
    public static void main(String[] args) {
        ObjectOutputStream os = null;
        try {
          os = new ObjectOutputStream(new FileOutputStream("salida.dat"));
          Persona p = new Persona("Fulanito Pérez", new Fecha(10, 12, 1985), 46722311);
          os.writeObject(p);
          System.out.println("Fichero escrito correctamente.");
        } catch (IOException e) {
          System.out.println("Error de E/S: " + e.getMessage());
        } finally {
          try {
              if (os != null) os.close();
          } catch (IOException e) { System.out.println(e.getMessage()); }
        }
    }
}
```

Lectura de objetos

```
import java.io.*;


public class TestEmpleadoLectura {
    public static void main(String[] args) {
        ObjectInputStream is = null;
        try {
            is = new ObjectInputStream(new FileInputStream("salida.dat"));
            Persona p = (Persona) is.readObject();
            p.imprimirDatos();
        } catch (IOException e) {
            System.out.println("Error de E/S: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                if (is != null) is.close();
            } catch (IOException e) { System.out.println(e.getMessage()); }
        }
    }
}
```

Aún hay más...

- La clase `File`
  - Sus objetos representan nombres de archivos.
  - Sirve para realizar operaciones sobre el sistema de archivos.
- Interfaz `Externalizable`
  - Permite al programador especificar cómo se serializan sus objetos.
- La “nueva” E/S (a partir de Java 1.4)
  - Paquete `java.nio`
- La nueva “nueva E/S” (a partir de Java 7)



Referencias

- B. Eckel
  Thinking in Java (3rd Edition)
  Cap. 12
- P. Deitel, H. Deitel
  Java. How to Program (9th Edition)
  Cap. 17
- E. R. Harold
  Java I/O, (2nd Edition)
- Documentación de librerías de Java
  http://docs.oracle.com/javase/6/docs/api/