DAM - Programación de servicios y procesos

Tema 1 - Introducción a Java

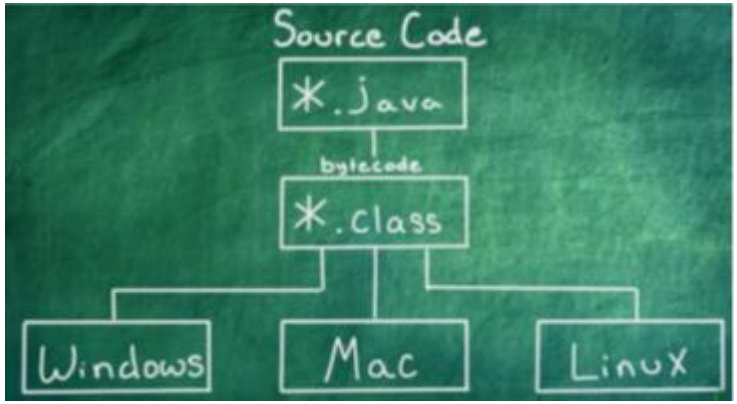
Roberto Sanz Requena
rsanz@florida-uni.es

Florida
Grup Educatiu

Índice

1. Introducción

- Lenguaje **compilado** e **interpretado** por una máquina **virtual** (JVM, independientes de las máquinas físicas).



- Muy utilizado a nivel mundial.
- Orientación a **objetos** (operador `new()`).
- **Seguridad** en el diseño:
  - Comprobación estricta de tipos de datos y métodos.
  - Garbage collector para objetos no referenciados.
  - Control obligatorio de excepciones.
  - Gestor de seguridad para controlar el acceso a los recursos del sistema.

1. Introducción

- **JDK** (Java Development Kit): paquete de herramientas de desarrollo.
  - Compilador (`javac`)
  - Máquina virtual (JVM)
  - Documentación (`javadoc`)
  - Empaquetador de proyectos (`jar`)

- **JRE** (Java Runtime Environment): paquete de herramientas de ejecución.
  - Máquina virtual (JVM)
  - Paquete básico de librerías

- **`CLASSPATH`**: variable de entorno donde la máquina virtual puede encontrar las clases Java que no están en el paquete básico (clases desarrolladas).

1. Introducción

- **IDE** (Integrated Development Environtment): aplicación que facilita el desarrollo de software incorporando ayudas técnicas.
  - IDEs populares para Java: Eclipse, NetBeans, IntelliJ IDEA, etc.

- Recomendaciones sobre la nomenclatura:
  - Los nombres de las clases empiezan por mayúscula.
  - Atributos, métodos e instancias de una clase empiezan por minúscula y, si están formados de varias palabras, sin espacios y empezando cada palabra (excepto al primera) en mayúscula.
  - Las constantes en mayúsculas.

2. Sintaxis de Java

- **Comentarios**
  - Comentario simple: //
  - Comentario bloque: /* … */
  - Comentario documentación: /** … */

- **Palabras reservadas**: no se pueden usar como identificadores (variables).

```
abstract    continue    for         new         switch
boolean     default     goto        null        synchronized
break       do          if          package     this
byte        double      implements  private     threadsafe
byvalue     else        import      protected   throw
case        extends     instanceof  public      transient
catch       false       int         return      true
char        final       interface   short       try
class       finally     long        static      void
const       float       native      super       while
```

2. Sintaxis de Java

- **Separadores**
  - Paréntesis `()` : introducir listas de parámetros, condiciones,…
  - Llaves `{}` : delimitar bloques de código y definir arrays.
  - Corchetes `[]` : declarar vectores (arrays) y acceder a sus elementos.
  - Punto y coma `;` : indica el final de una instrucción.
  - Coma `,` : separar elementos de una lista, encadenar expresiones,…
  - Punto `.` : acceder a los atributos y métodos de una clase.

3. Tipos de datos

- Tipos de datos **primitivos** y tipos de datos **referencia**.

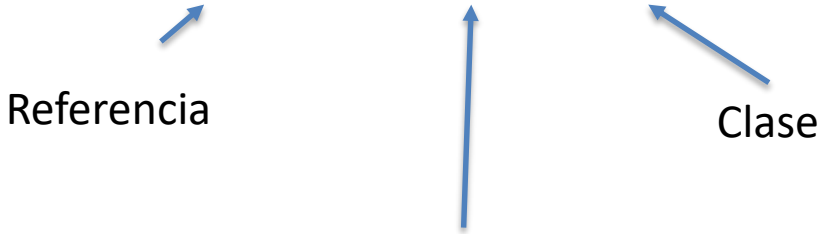
- Tipos primitivos (conversión entre tipos).

Tipo	Descripción
`boolean`	`true` / `false`
`char`	Carácter Unicode de 16 bits
`byte`	Entero en complemento a dos con signo de 8 bits
`short`	Entero en complemento a dos con signo de 16 bits
`int`	Entero en complemento a dos con signo de 32 bits
`long`	Entero en complemento a dos con signo de 64 bits
`float`	Real en punto flotante según la norma IEEE 754 de 32 bits
`double`	Real en punto flotante según la norma IEEE 754 de 64 bits

3. Tipos de datos

- Tipos referencia: objetos (instancia de una clase).

```
Punto unPunto = new Punto();
```



Generador de instancia:

- Invoca un método constructor del objeto definido en la clase
- Reserva recursos en memoria

```java
public class Punto {

    protected float x;
    protected float y;

    public Punto() {
        x = 0.0f;
        y = 0.0f;
    }

    public Punto(float x, float y) {
        this.x = x;
        this.y = y;
    }

    public float get (String coordenada) {
        if(coordenada.equals("x"))
            return x;
        else if(coordenada.equals("y"))
            return y;
        }

    public float getX() {
        return x;
    }

    public void setX(float x) {
        this.x = x;
    }

    public void setY(float y) {
        this.y = y;
    }

    public float getY() {
        return y;
    }

    public Punto inverso() {
        return new Punto(-x, -y);
    }

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

3. Tipos de datos

- Tipos referencia: objetos (instancia de una clase).

```
Punto unPunto = new Punto();
unPunto.print();
Punto otroPunto = unPunto;
otroPunto.setX(1.0f);
otroPunto.setY(2.0f);
unPunto.print();
```

```
Coordenadas del punto (0.0,0.0)
Coordenadas del punto (1.0,2.0)
```

¿`unPunto` y `otroPunto` son lo mismo?

3. Tipos de datos

- **Ámbito** de una variable: parte del código donde se declara más los bloques anidados que queden por debajo.

- **Operadores**:
  - Aritméticos: `*` `/` `%` `+` `-`
  - Unarios: `++` `--` `!`
  - Asignación: `=`
  - Relacionales: `==` `!=` `<` `<=` `>` `>=`
  - Lógicos: `&&` `||`
  - Ternario: `?:`
  - Bit a bit: `&` `|` `^` `~`

- **Precedencia**: orden de las operaciones (uso de paréntesis).

3. Tipos de datos

Arrays: clase Array

```
int arrayDeEnteros[] = null;  //Declarar array
Punto arrayDePuntos[] = null;

arrayDeEnteros = new int[100];    // Reservar memoria
arrayDePuntos = new Puntos[100];

int arrayDeEnteros[] = {1, 2, 3, 4, 5};   //Inicializa y asigna
Punto arrayDePuntos[] = {new Punto(), new Punto(1.0f, 1.0f)};

Punto matriz[][] = new Punto[3][3]; // Declarar matriz
Punto fila[] = {new Punto(),new Punto(1.0f,1.0f),new Punto(2.0f, 2.0f)};
matriz[0] = fila[];
```

¿Por qué el método principal en Java siempre incluye: `main(String args[])`?

3. Tipos de datos

Strings: clase String (almacenar cadenas de caracteres no modificables).

```
String frase = "Esta cadena es una frase"
String fraseLarga = frase + " que se puede convertir en una frase larga."
char letras[]={"a", "b", "c"};
String otraCadena = new String(letras);
String pi = String.valueOf(3.141592);
```

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

4. Estructuras de control

Estructuras condicionales

```
if(condicion1) {

    bloqueDeInstrucciones();

} else if(condicion2) {

    bloqueDeInstrucciones();

} else {

    bloqueDeInstrucciones();

}
```

```
switch(expresionEntera) {

    case valor1:
        instrucciones();
        break;

    case valor2:
        instrucciones();
        break;

    default:
        instrucciones();

}
```

4. Estructuras de control

Estructuras de repetición (bucles)

```
for(iniciación; condición; incremento) {
    // Bloque de instrucciones
}
```

Palabras clave **break** y **continue**:

break: terminar la ejecución del bucle
continue: termina el bloque de instrucciones y pasa a la siguiente iteración

```
for(int i = 0; i < 10; i++) {
     for(int j = 0; j < 10; j++) {
          if (j > i)
               break;
          System.out.print(j+",");
     }
     System.out.println("");
}
```

```
0,
0,1,
0,1,2,
0,1,2,3,
0,1,2,3,4,
0,1,2,3,4,5,
0,1,2,3,4,5,6,
0,1,2,3,4,5,6,7,
0,1,2,3,4,5,6,7,8,
0,1,2,3,4,5,6,7,8,9,
```

4. Estructuras de control

Estructuras de repetición (bucles)

```
while(condición) {
    // Bloque de instrucciones
}


do {
    // Bloque de instrucciones
} while(condición);
```

→ Evalúa condición antes de ejecutar

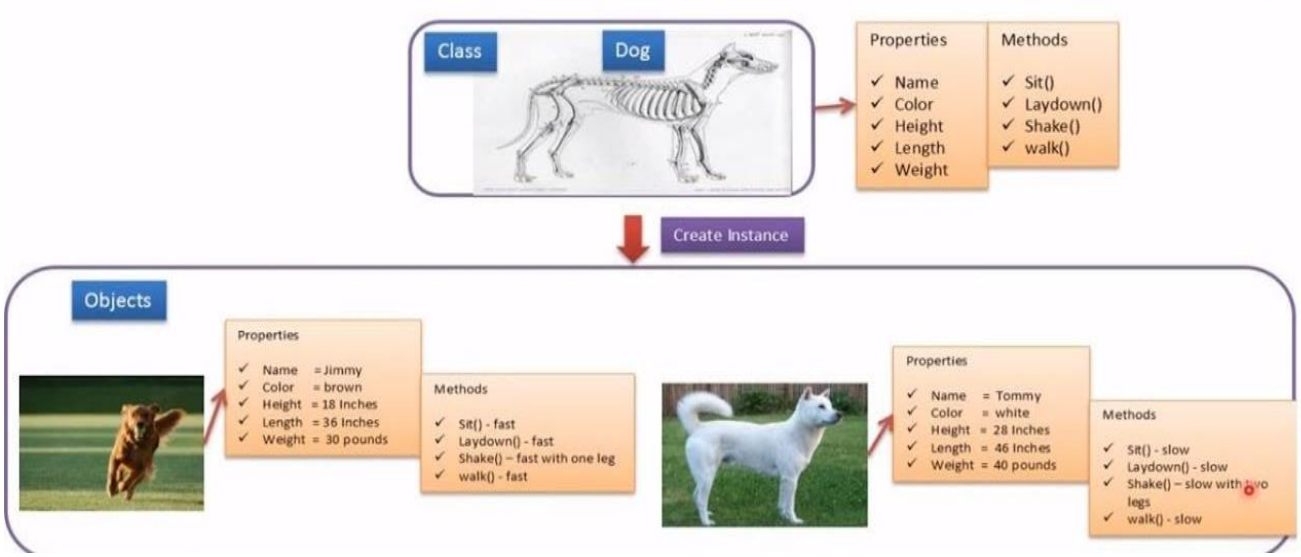
→ Ejecuta una vez al menos

Se modifica la/s variable/s de control asociadas a la condición
Admiten también las instrucciones `break` y `continue`

5. Clases

- En POO los objetos son instancias de una clase (plantilla) con atributos (valores) y métodos (acciones).

- Clase `Object` (clase raíz en Java): implementa métodos básicos como `toString()`, que muestra información y otros como `wait()`, `notify()` y `notifyAll()`, importantes para la concurrencia.



5. Clases

Atributos de una clase: definición

- Modificador de acceso:
  - `private`: accesible solo desde la clase.
  - `protected`: accesible desde la clase y las clases que la heredan.
  - `(Sin modificador)`: accesible desde clases del mismo paquete (`package`).
  - `public`: accesible desde cualquier clase.
- Modificadores adicionales (opcionales):
  - `static`: atributo de clase, común a todas las instancias de la clase.
  - `final`: atributo no modificable común a todas las instancias de la clase.
- Tipo de dato: primitivo o referencia (objeto).

5. Clases

Métodos de una clase: definición

- Modificador de acceso:
  - `private`: accesible solo desde la clase.
  - `protected`: accesible desde la clase y las clases que la heredan.
  - `(Sin modificador)`: accesible desde clases del mismo paquete.
  - `public`: accesible desde cualquier clase.
- Modificadores adicionales (opcionales):
  - `static`: método de clase, se puede usar sin instanciar la clase.
  - `final`: método no modificable por ninguna clase que herede de esta.
- Tipos de datos: primitivo o referencia (objeto) que recibe/devuelve el método, con la palabra `void` si no devuelve nada.

5. Clases

Métodos de una clase: método Constructor

- Mismo nombre que la clase.
- Admite sobrecarga (polimorfismo): varios métodos con el mismo nombre pero con número/tipo diferente de argumentos.
- Constructor por defecto: sin argumentos.

```
public Punto() {
    x = 0.0f;
    y = 0.0f;
}

public Punto(float x, float y) {
    this.x = x;
    this.y = y;
}
```

5. Clases

Clase ArrayList (colecciones y listas)

```java
// A.java
public class A {
     protected int a = 0;
     public A() {};
     public A(int a) { this.a = a; }
     public void getA() {
          System.out.println("Clase A,
               valor: "+a);
     }
}

// B.java
public class B extends A {
     public B() { super(0); }
     public B(int a) { super(a); }
     public void getA() {
          System.out.println("Clase B,
               valor:"+a);
     }
}
```

```java
// Coleccion.java
import java.util.ArrayList;
import java.util.Iterator;

public class Coleccion {
     public static void main(String args[]) {
          ArrayList al = new ArrayList();
          for(int i = 0; i < 10; i += 2)
               al.add(new A(i));
          for(int i = 1; i < 10; i += 2)
               al.add(new B(i));
          A a = null;
          Iterator it = al.iterator();
          while(it.hasNext() == true) {
               a = (A)it.next();
               a.getA();
          }
     }
}
```

5. Clases

Clases envoltura (wrapper)

- Se emplean cuando hay que manejar los tipos primitivos como objetos.
- Aplicaciones: trabajo con colecciones (`ArrayList`), que no admiten tipos primitivos, fácil conversión número-string y viceversa, etc.

```
ArrayList<int> myNumbers = new ArrayList<int>(); // Invalid
ArrayList<Integer> myNumbers = new ArrayList<Integer>(); // Valid
```

```
Integer myInt = new Integer(15);
String myString = myInt.toString();
int entero = myInt.intValue();
```

```
String s="200";
int i=Integer.parseInt(s);
System.out.println(s+100);
System.out.println(i+100);
```

https://www.w3schools.com/java/java_wrapper_classes.asp

5. Clases

Gestión de clases en un proyecto Java

- Definición de paquetes (`package`).
- Organización jerárquica de clases.
- Nomenclatura y organización adecuada a la aplicación.
- Ejemplo de nomenclatura: `com.ejemplo.package.`
    - La clase compilada se almacenaría en la ruta `com/ejemplo/package.`
    - El objetivo es evitar interferencias en los nombres de las clases utilizadas cuando la JVM ejecute la aplicación
- `import` para acceder a clases de otro paquete.

https://en.wikipedia.org/wiki/Java_package#Package_naming_conventions

6. Herencia e interfaces

- **Herencia**: extender el comportamiento de otra clase (reaprovechar código).
- Palabra reservada `extends`: hereda atributos y métodos que no sean de tipo `private`.
- Acceso a los métodos de la clase antecesora mediante la palabra reservada `super`.
- Permite añadir nuevos métodos y atributos.
- Permite sobreescribir métodos y atributos (precaución).
- Cuando se crea una instancia de una clase que extiende a otra, hay implícita una llamada al constructor de la clase antecesora, por lo que es importante implementar siempre constructores por defecto.

```java
public class Punto3D extends Punto {
    protected float z;
    public Punto3D() {
        super();
        this.z = 0.0f;
    }
    public Punto3D(float x, float y) {
        super(x,y);
        this.z = 0.0f;
    }
    public Punto3D(Punto unPunto) {
        super(unPunto.getX(), unPunto.getY());
        this.z = 0.0f;
    }
    public Punto3D(float x, float y, float z) {
        super(x, y);
        this.z = z;
    }
    public float getZ() {
        return z;
    }
    public Punto inverso() {
        return new Punto3D(-getX(),-getY(),-z);
    }
    public String toString() {
        return "("+getX()+","+getY()+","+z+")";
    }
    public void print() {
        System.out.println("Coordenadas "+this);
    }
}
```

6. Herencia e interfaces

- **Clase abstracta:** plantilla para la creación de otras clases.
    - Las clases que la hereden deben sobreescribir los métodos declarados como abstract.
- **`Interface`**: generalización de la clase abstracta.
    - Especifica los atributos y métodos que debe tener una clase, aunque no los implementa (no se pueden crear instancias de una `interface`).
    - La clase que implemente una `interface` debe implementar todos sus métodos también (“contrato” de implementación).
    - Las `interfaces` pueden heredar de otras `interfaces`.

7. Excepciones

- **Excepción**: situación anómala en la ejecución del código que debe ser gestionada para evitar la inestabilidad de la aplicación.
- Obligatoriedad de gestionarlas de alguna forma.
- Caso particular: excepciones del tipo `RunTimeException` (como `NullPointerException`) generadas durante la ejecución del código.
    - El compilador Java no exige tratarlas, pero puede ser conveniente para preservar la estabilidad de la aplicación.
- Gestión
    - Mediante `try … catch … finally`
    - Mediante `throws` y `throw`

7. Excepciones

- Mediante `try … catch … finally`

```
try {
     readFromFile("esteFicheroNoExiste");
}
catch(FileNotFoundException e) {
     //Aquí tratamos esta excepción
     e.printStackTrace();
}
catch(IOException e) {
     //Aquí tratamos esta otra
     e.printStackTrace();
}
finally {
     //Aquí realizamos las tareas comunes
}
```

```
try {
     readFromFile("esteFicheroNoExiste");
}
catch(Exception e) {
     //Tratamiento general
     e.printStackTrace();
}
finally {
     //Aquí realizamos las tareas comunes
}
```

7. Excepciones

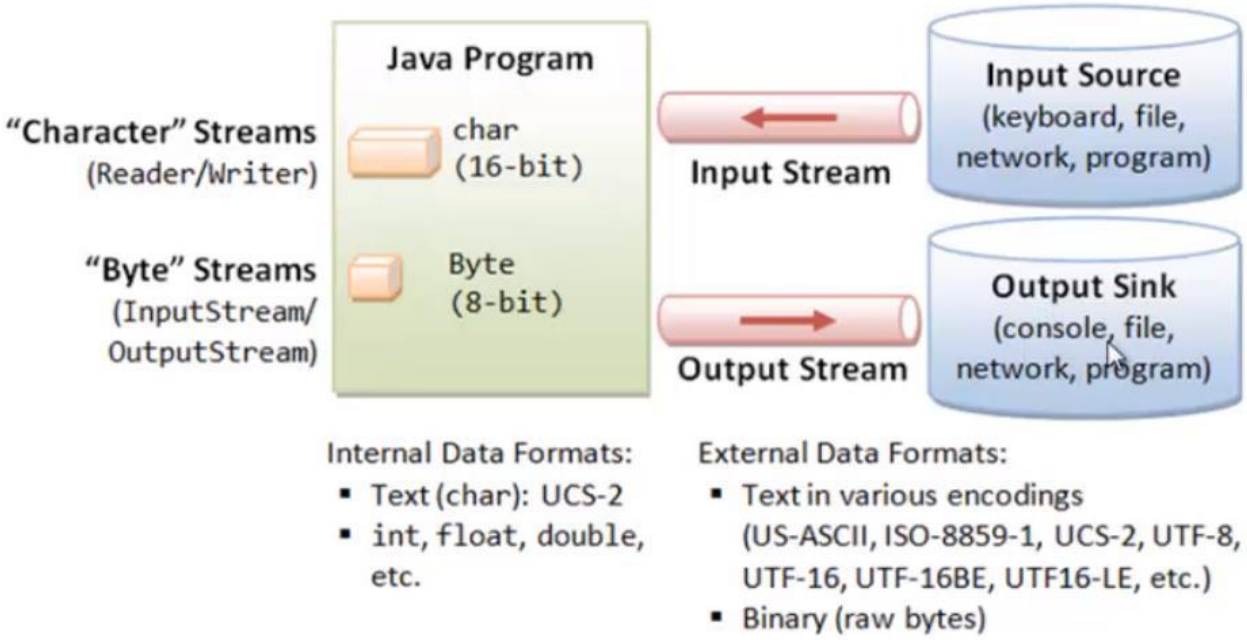
- Mediante `throws` y `throw`

```
void metodoLanzador(int a) throws IOException, ClassNotFoundException {
    ...
}
```

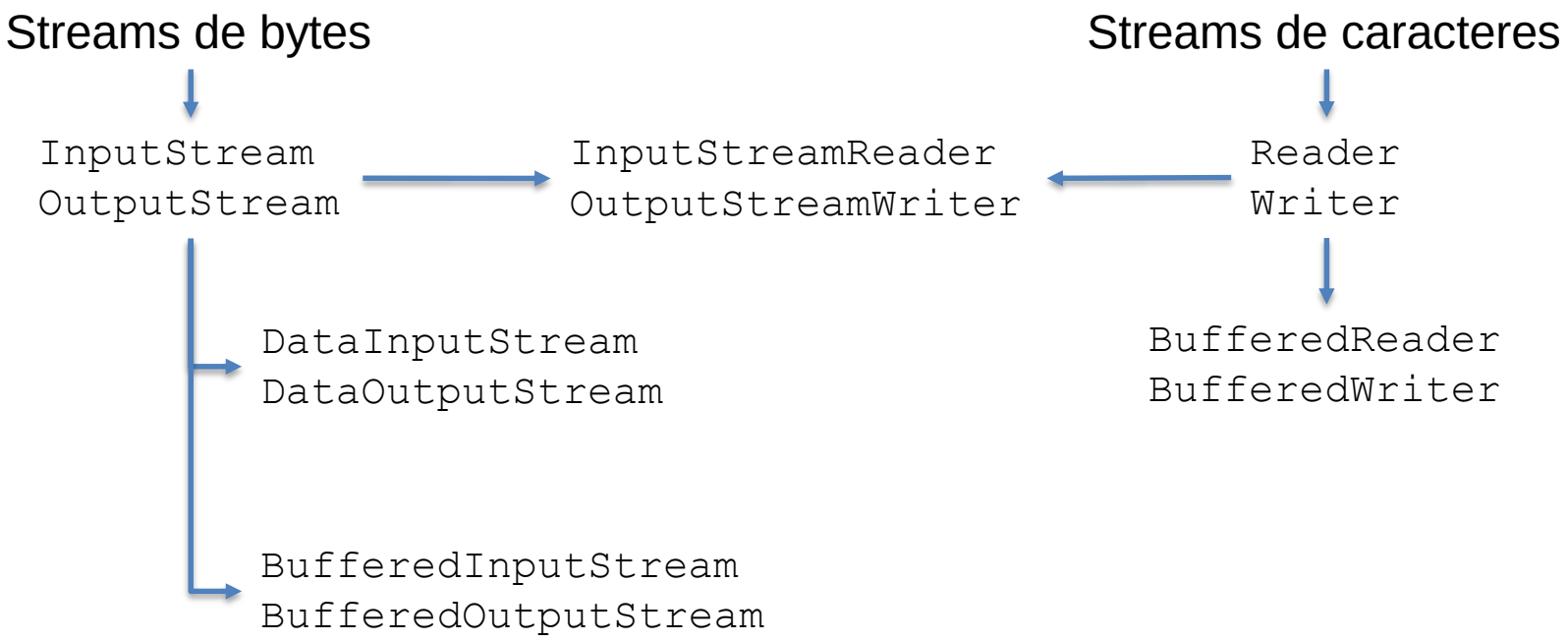
```
public static void addInteger(Integer value) throws IllegalArgumentException
{
    if (integers.contains(value)) {
        throw new IllegalArgumentException("Integer already added.");
    }
    integers.add(value);
}
```

Otro método deberá encargarse de gestionar estas excepciones con `try…catch`.

8. Entrada y salida



8. Entrada y salida



8. Entrada y salida

```
// Reader a partir de System.in (entrada teclado)
InputStreamReader isr = new InputStreamReader(System.in);

// Wrapper para utilizar el buffer
BufferedReader br = new BufferedReader(isr);

try{

    String cadena = br.readLine(); // Leer cadena de caracteres

} catch(IOException e) {

    e.printStrackTrace();

}
```

8. Entrada y salida

Clase `File`

- Listar, crear, renombrar, obtener información o borrar ficheros y directorios.
- No gestiona el contenido de un fichero.
- Streams desde/hacia ficheros: bytes (`FileInputStream`, `FileOutputStream`) y caracteres (`FileReader`, `FileWriter`), además de sus versiones “buffer”.

```
File f = new File("fichero.txt");
FileReader fr = new FileReader(f);
BufferedReader br = new BufferedReader(fr);
String linea =br.readLine();
```

Actividad Entregable 1 - Java

Presentación de la Actividad Entregable 1 (AE1_T1_Java)

Otros recursos

https://introcs.cs.princeton.edu/java/11cheatsheet/

https://www.youtube.com/playlist?list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk

https://guru99.es/java-tutorial/