

Sesión 7. Manejo de Errores. Excepciones

Metodología y Desarrollo de Programas

- Introducción
- Estructura de excepciones
- Clases Exception en Java
- Bibliografía usada

Ventajas de usar excepciones: Separar código de casos de error

➤ Supongamos el siguiente ejemplo

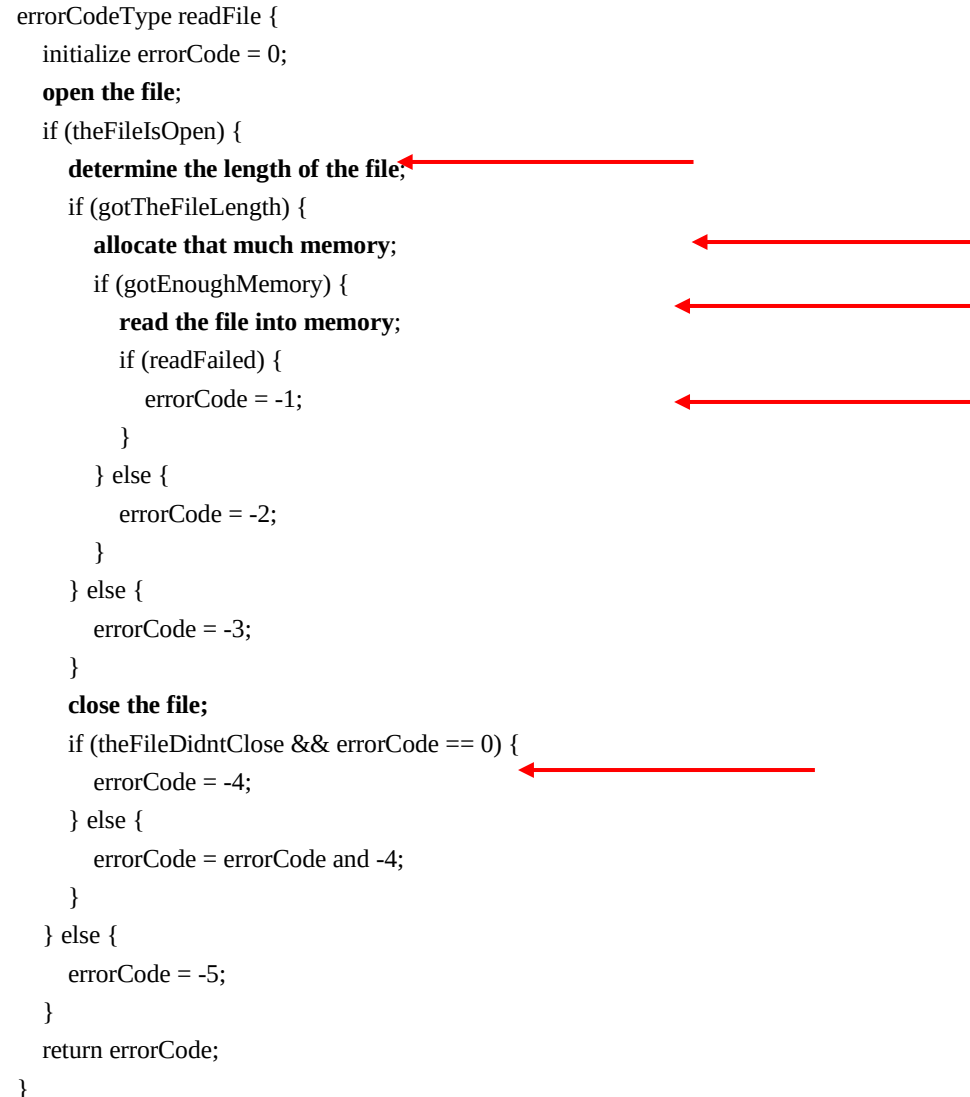
```
readFile {  
    open the file;  
    determine its size;  
    allocate that much memory;  
    read the file into memory;  
    close the file;  
}
```

¿Qué errores pueden producirse?

Ventajas de usar excepciones: Separar código de casos de error

Sin excepciones

```
errorCodeType readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        } else {
            errorCode = errorCode and -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```



The diagram illustrates the error handling logic in the provided code. Red arrows point to the following lines:

- determine the length of the file;**
- allocate that much memory;**
- read the file into memory;**
- errorCode = -1;
- errorCode = -4;

- ¿Qué hacer si de repente se produce un error irreversible en el programa? Dos posibles soluciones:
 - ❖ Tratarlo mediante una instrucción selectiva:
 - ✓ Mostrar un mensaje de error
 - ✓ Devolver al procedimiento que lo llama un valor especial
 - ✓ Interrumpir la ejecución
 - ❖ No tratarlo: No hacer nada y continuar como si nada hubiera pasado
- C++ y Java posee un mecanismo de gestión de errores incorporado que se denomina *manejo de excepciones*
- La utilización del manejo de excepciones permite gestionar y responder a los errores en tiempo de ejecución
 - ❖ Divisiones por cero
 - ❖ Reserva de memoria no concedida
 - ❖ Apertura de ficheros inexistentes
 - ❖ Lectura de ficheros incorrecta
 - ❖ etc.

➤ Construido a partir de cuatro palabras reservadas:

❖ **try**

- ✓ Bloque de código donde se quiere controlar una circunstancia anómala en la ejecución del programa
- ✓ Si se produce se lanza una excepción, se interrumpe el flujo de ejecución y el control se pasa al manejador de excepciones

❖ **catch**

- ✓ Manejador: Bloque de código donde se captura la excepción producida anteriormente y se procesa decidiendo que hacer
- ✓ Siempre que exista una instrucción try debe existir al menos un catch

❖ **throw**

- ✓ Cuando se produce una excepción, ésta puede ser lanzada por el sistema o por el usuario. En este segundo caso se utiliza la instrucción throw.

- **finally (java)**

- ✓ Instrucciones que se ejecutan siempre tanto si ha existido excepción como si no ha existido

Ventajas de usar excepciones: Separar código de casos de error

Con excepciones

```
readFile {  
    try {  
        open the file;  
        determine its size;  
        allocate that much memory;  
        read the file into memory;  
        close the file;  
    } catch (fileOpenFailed) {  
        doSomething;  
    } catch (sizeDeterminationFailed) {  
        doSomething;  
    } catch (memoryAllocationFailed) {  
        doSomething;  
    } catch (readFailed) {  
        doSomething;  
    } catch (fileCloseFailed) {  
        doSomething;  
    }  
}
```

Ejemplo 1. Excepciones

➤ Captura de excepción

```
public class Ejemplo01_Exception {  
    public static void main(String args[]) {  
        int val1, val2;  
        try {  
            System.out.println("Inicio");  
            val1 = 0;  
            val2 = 25 / val1;  
            System.out.println(val2);  
            System.out.println("No llega NUNCA");  
        } catch (ArithmeticException e) {  
            // handler for ArithmeticException  
            System.out.println("ArithmeticException :: División por cero!!");  
        }  
        System.out.println("Después del try-catch");  
    }  
}
```


Ejemplo 2. Excepciones

➤ Captura de múltiples excepciones

```
public class Ejemplo02_Exception {
    public static void main(String args[]) {
        try {
            System.out.println("Inicio");
            int myArray[] = new int[] {1,2,3,4,5};
            myArray[5] = 10 / 0;
            System.out.println("No llega NUNCA");
            // multiple catch blocks
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception :: División por cero!!");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("ArrayIndexOutOfBoundsException :: Fuera de rango");
        } catch (Exception e) {
            System.out.println("Otra Excepcion :: " + e.getMessage());
        }
        System.out.println("Después del try-catch");
    }
}
```

Ejemplo 3. Excepciones

➤ Try-catch anidados

```
public static void main(String args[]) {  
    try {  
        try { // try block1  
            System.out.println("Inicio try bloque 1");  
            int num = 15 / 0;  
            System.out.println("No llega nunca: bloque 1");  
        } catch (ArithmeticException e1) {  
            System.out.println("Bloque 1 Exception: e1");  
        }  
        try { // try block2  
            System.out.println("Inicio try bloque 2");  
            int num = 100 / 0;  
            System.out.println("No llega nunca: bloque 2");  
        } catch (ArrayIndexOutOfBoundsException e2) {  
            System.out.println("Bloque 2 Exception: e2");  
        }  
        System.out.println("Después bloque 1 y 2");  
    } catch (ArithmeticException e3) {  
        System.out.println("Main Arithmetic Exception");  
    } catch (ArrayIndexOutOfBoundsException e4) {  
        System.out.println("Main ArrayIndexOutOfBoundsException");  
    } catch (Exception e5) {  
        System.out.println("Main General Exception");  
    }  
    System.out.println("Después del try-catch");  
}
```

Ejemplo 4. Excepciones

➤ Try-catch-finally

```
public class Ejemplo04_Exception {  
    public static void main(String args[]) {  
        int val1, val2;  
        try {  
            System.out.println("Inicio");  
            val1 = 0;  
            val2 = 25 / val1;  
            system.out.println(val2);  
            System.out.println("No llega NUNCA");  
        } catch (ArithmeticException e) {  
            // handler for ArithmeticException  
            System.out.println("ArithmeticException :: División por cero!!");  
        } finally {  
            System.out.println("Siempre ejecuto esto!!");  
        }  
        System.out.println("Después del try-catch");  
    }  
}
```

Ejercicio 1. Controlar las excepciones

```
public static void main(String args[]) {
    Scanner scan = new Scanner(System.in);
    // NumberFormatException
    String s=scan.nextLine(); //Introducimos una a
    int x= Integer.parseInt(s);
    //InputMismatchException
    x=scan.nextInt(); //Introducimos una a
    //NullPointerException
    String s1 = null; //null value
    System.out.println(s1.charAt(0));
    //ArithmeticException
    int b= scan.nextInt(); //Introducimos un 0
    int a = 30;
    int c = a / b; // cannot divide by zero
    System.out.println("Result = " + c);
    //ArrayIndexOutOfBoundsException
    int []v = new int[] {1,2,3,4,5};
    for (int i=0;i<=v.length;i++) {
        System.out.println(v[i]);
    }
}
```

Ejemplo 5. Excepciones

➤ Métodos con excepciones

```
public class Ejemplo02_Exception {
    public static void main(String args[]) {
        try {
            metodo1();
        } catch (NullPointerException e) {
            System.out.println("Excepcion en main");
        }
    }
    private static void metodo1() {
        try {
            metodo2();
        } catch (NullPointerException e) {
            System.out.println("Excepcion en método 1");
        }
    }
    private static void metodo2() {
        metodo3();
    }
    private static void metodo3() {
        Object a=null;
        System.out.println(a.toString());
    }
}
```

Ejemplo 6a. Excepciones

➤ Métodos con excepciones. Con throws

```
public class Ejemplo06_Exception {
    public static void main(String args[]) {
        try {
            metodo1();
        } catch (NullPointerException e) {
            System.out.println("Excepcion en main");
        }
    }
    private static void metodo1() {
        try {
            metodo2();
        } catch (NullPointerException e) {
            System.out.println("Excepcion en método 1");
        }
    }
    private static void metodo2() throws NullPointerException {
        metodo3();
    }
    private static void metodo3() throws NullPointerException {
        try {
            Object a=null;
            System.out.println(a.toString());
        } catch (NullPointerException e) { throw e;}
    }
}
```

Ejemplo 6b. Excepciones

➤ Métodos con excepciones. Con throws

```
public class Ejemplo06_Exception {  
    public static void main(String args[]) {  
        try {  
            metodo1();  
        } catch (NullPointerException e) {  
            System.out.println("Excepcion en main");  
        }  
    }  
    private static void metodo1() {  
        try {  
            metodo2();  
        } catch (NullPointerException e) {  
            System.out.println("Excepcion en método 1");  
        }  
    }  
    private static void metodo2() throws NullPointerException {  
        metodo3();  
    }  
    private static void metodo3() {  
        Object a=null;  
        System.out.println(a.toString());  
    }  
}
```

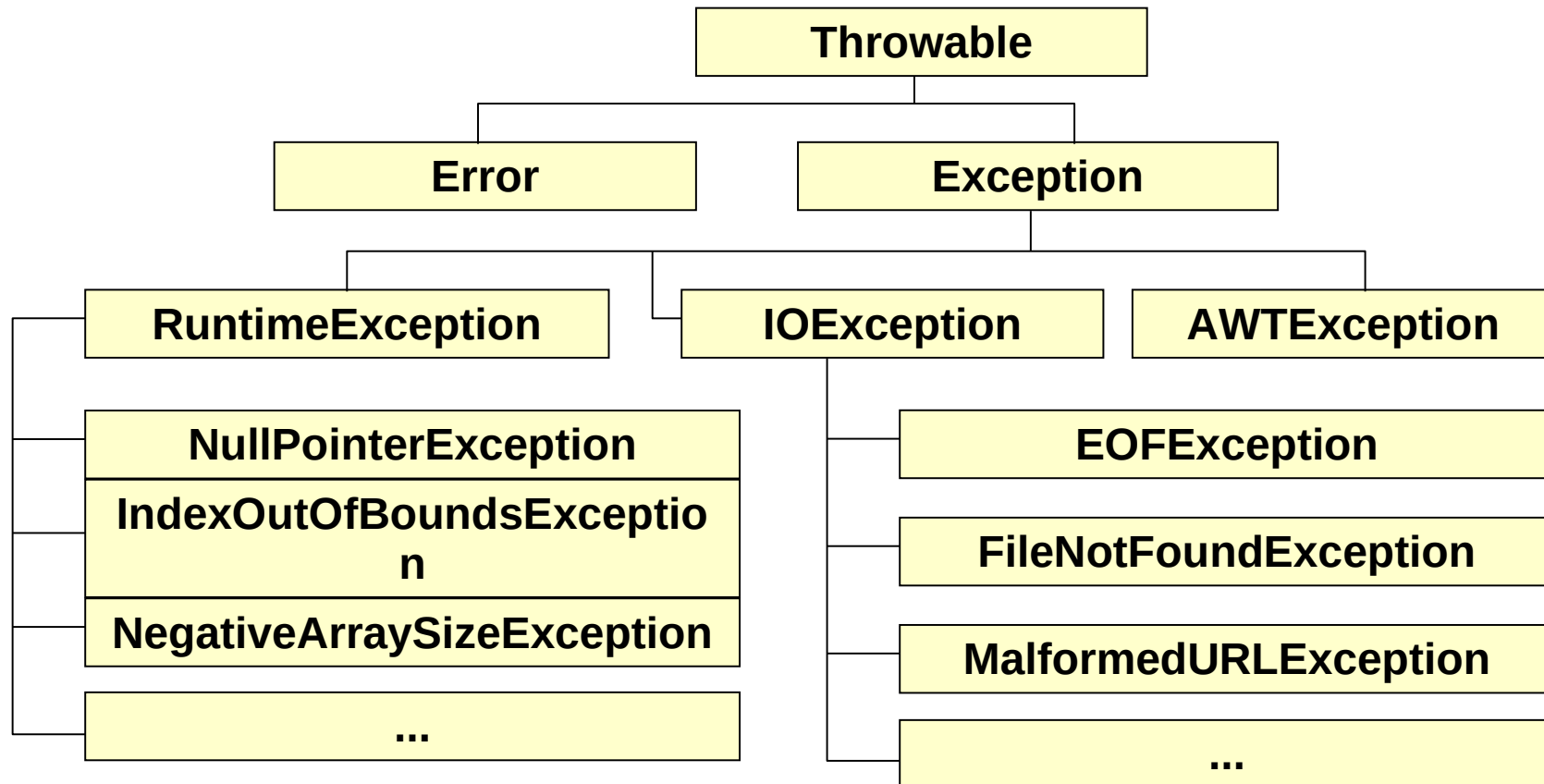
Ejemplo 7. Excepciones

- Relanzar o no una excepción. This is the question!!!!

```
public class Ejemplo07_Exception {
    Scanner scan = new Scanner(System.in);
    public static void main(String args[]) {
        Ejemplo02_Exception e= new Ejemplo02_Exception();
        int x=e.leer();
        int y=e.leer();
        System.out.println(e.dividir(x, y));
    }
    public int leer() { //No devolver excepción
        String s = scan.nextLine();
        int x = Integer.parseInt(s);
        return x;
    }
    public int dividir(int a, int b) {
        return a / b;
    }
}
```


Excepciones en Java

➤ Jerarquía de clases en Java



Excepciones en Java

- La clase **java.lang.Throwable** es la superclase de las clases utilizadas para manejar excepciones
- Existen dos tipos de excepciones de la clase Throwable:
 - ❖ **java.lang.Error** representa los errores de compilación y del sistema. Estos errores son ajenos al programador y son **irrecuperables**.
 - ❖ **Java.lang.Exception** representa las excepciones generadas por la aplicación y que deben ser capturadas por el programador mediante (try/catch/finally).
- Dentro de Exception, dos tipos:
 - ❖ **Runtime Exceptions:** Son las excepciones que se producen en el sistema de ejecución de Java tales como referencias null, hacer división por cero, acceder a un elemento inexistente en un array
 - ❖ **NonRuntime Exceptions:** Son excepciones que se producen fuera del sistema de ejecución de Java. Son ejemplo de estas las excepciones que se producen por acceso a archivos (IOExceptions)

Excepciones en Java

- Las clases derivadas de Exception pueden pertenecer a distintos paquetes
 - ❖ java.lang: Throwable, Exception, RuntimeException,...
 - ❖ java.io: FileNotFoundException,...
 - ❖
- Toda clase heredada de Throwable tiene los siguientes métodos
 - ❖ String **getMessage()**: Recupera el mensaje asociado a la excepción
 - ❖ String **toString()** String que describe la excepción
 - ❖ void **printStackTrace()** muestra el método que lanzó la excepción

Excepciones en Java

- Excepciones más comunes:
 - ❖ IOException Generalmente fallas de entrada o salida, tal como la imposibilidad de leer desde un archivo
 - ❖ NullPointerException: Referencia a un objeto NULL
 - ❖ NumberFormatException: Una conversión fallida entre Strings y números
 - ❖ OutOfMemoryException: Muy poca memoria para instanciar un objeto nuevo (new)
 - ❖ SecurityException: Un applet tratando de realizar una acción no permitida por la configuración de seguridad del browser
 - ❖ StackOverflowException: El sistema corriendo fuera de espacio en Stack (stack space)

Excepciones en Java

- Programador se puede crear sus propias excepciones, para lo cual sólo tiene que heredar de la clase Exception
- Exception tiene 4 constructores, de los cuales:
 - ❖ Por defecto (no recibe parámetros)
 - ❖ Parametrizado, recibe una cadena con el mensaje de error

```
class MiExcepcion extends Exception {  
    public MiExcepcion() {  
        super();  
        ...  
    }  
    public MiExcepcion(String s) {  
        super(s);  
        ...  
    }  
}
```

Ejemplo 8. Excepciones Propias

➤ Con excepción propia: MDPEException

```
public class Ejemplo08_Exception {  
    public static void main(String args[]) {  
        try {  
            metodo1();  
        } catch (MDPEException e) {  
            System.out.println("Excepcion en main");  
        }  
    }  
    private static void metodo1() {  
        try {  
            metodo2();  
        } catch (MDPEException e) {  
            System.out.println("Excepcion en método 1");  
        }  
    }  
    private static void metodo2() throws MDPEException {  
        metodo3();  
    }  
    private static void metodo3() {  
        try {  
            Object a = null;  
            System.out.println(a.toString());  
        } catch (NullPointerException e) {  
            throw new MDPEException();  
        }  
    }  
}
```

Práctica del Desguace

➤ Se pide implementar las siguientes excepciones propias para la práctica del Desguace.

❖ ExceptionVectorLleno

❖ ExceptionVectorVacio

❖ ExceptionVectorFueraRango

❖ ExceptionBastidorNoEncontrado

❖ ExceptionPiezaNoEncontrada

Práctica del Desguace

- Para el desarrollo de esta práctica se proporciona el código de algunos métodos de desguace pero sin control de errores

```
public Vehiculo getVehiculoBastidor(Integer bastidor) {  
    for (int i = 0; i < vehiculos.size(); i++) {  
        if (vehiculos.get(i).getBastidor().equals(bastidor)) {  
            return vehiculos.get(i);  
        }  
    }  
    return null;  
}  
  
public boolean addVehiculo(Vehiculo p) {  
    return vehiculos.add(p);  
}  
public boolean addPiezaDesguace(Pieza p) {  
    return piezas.add(p);  
}
```


Práctica del Desguace

- Para el desarrollo de esta práctica se proporciona el código de algunos métodos de desguace pero sin control de errores

```
public Pieza getPiezaDesguace(String id) {  
    Iterator<Pieza> it = catalogo.iterator();  
    while (it.hasNext()) {  
        Pieza pieza = (Pieza) it.next();  
        if (pieza.getId().equals(id)) {  
            return pieza;  
        }  
    }  
    return null;  
}
```

```
public boolean addPiezaVehiculo2(String id, Integer bastidor) {  
    Pieza p1 = getPiezaDesguace(id);  
    Vehiculo aux = getVehiculoBastidor(bastidor);  
    Pieza p = new Pieza(p1.getId(), p1.getNombre(), 1);  
    aux.addPiezaV(p);  
    p1.setStock(p1.getStock()-1);  
}
```

Práctica del Desguace

- ExceptionVectorLleno se debe generar manualmente en el método addPiezaV de Vehiculo (por tanto se deberá controlar en los métodos que hagan su llamada)

```
public boolean addPiezaV(Pieza p) throws ExceptionVectorLleno {  
    for (int i = 0; i < piezas.length; i++) {  
        if (piezas[i] != null && piezas[i].equals(p))  
            return false;  
    }  
    if (cont == piezas.length)  
        throw new ExceptionVectorLleno();  
    else {  
        piezas[cont] = p;  
        cont++;  
        return true;  
    }  
}
```

- ExceptionVectorVacio: lo genera getPiezaV de Vehiculo. Ver en qué caso se debe generar
- ExceptionVectorFueraRango: lo genera getPiezaV de Vehiculo. Ver en qué casos se debe generar.

Práctica del Desguace

- Una vez realizado estos cambios, se debe controlar estas excepciones en Desguace. Probablemente tengamos algún error de compilación en Desguace por la incorporación de excepciones en Vehículo
- A continuación se pide hacer dos nuevas excepciones:
 - ❖ ExceptionBastidorNoEncontrado: Lo debe generar getVehiculoBastidor() en Desguace
 - ❖ ExceptionPiezaNoEncontrada: Lo debe genera el método getPiezaDesguace() en Desguace
- A continuación se pide que la excepción de **PiezaNoEncontrada** no sea relanzada fuera de Desguace mientras que la excepción de **BastidorNoEncontrado** sea relanzada fuera de Desguace.
- Se pide modificar el main de la sesión 6 (factoría) para controlar las excepciones que se puedan producir.

Práctica del Desguace

- ExceptionVectorLleno se debe generar manualmente en el método addPiezaV de Vehiculo (por tanto se deberá controlar en los métodos que hagan su llamada).
- ExceptionVectorVacio: lo genera getPiezaV de Vehiculo
- ExceptionVectorFueraRango: lo genera getPiezaV de Vehiculo
- • ExceptionBastidorNoEncontrado: Lo debe generar getVehiculoBastidor() en Desguace
- • ExceptionPiezaNoEncontrada: Lo debe genera el método getPiezaDesguace() en Desguace
-

Bibliografía Recomendada

➤ Libros C++:

- ❖ C++ Guia de autoenseñanza. Herbert Schildt.
- ❖ Programación en C++ : algoritmos, estructuras de datos y objetos. Luis Joyanes Aguilar. [S004.43C++joy]
- ❖ Como programar en C++ . H. M. Deitel. [S004.43C++dei]
- ❖ Resolución de problemas con C++. Savitch. [S004.43C++sav]

➤ Manuales en Internet C++

- ❖ Aprenda C++ como si estuvieras en primero. Universidad de Navarra.

➤ Libros Java

- ❖ **Piensa en Java. 4ª Edición. Bruce Eckel. Pearson Prentice Hall.**
- ❖ Core Java 2. Autores Cay S. Horstmann Y Gary Cornell. Editorial Pearson Educación
- ❖ Java 2. Manual De Programación. Luis Joyanes Aguilar; Matilde Fernández Azuela. Editorial McGraw-Hill