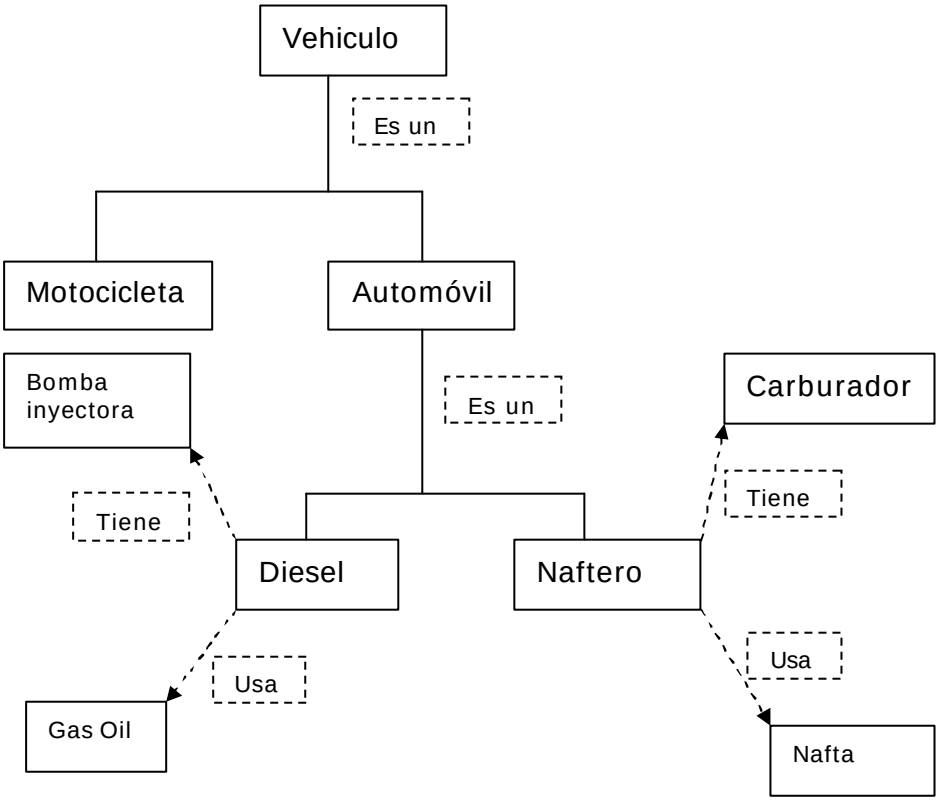
Modularidad

Se trata de como

distribuir responsabilidades:

estableciendo

RELACIONES ENTRE OBJETOS



Tipo de relación	Descripción
"tiene un"	El objeto de nuestra clase **tiene** atributos que son objetos de otras clases
"usa un"	Métodos pertenecientes al objeto de nuestra clase requieren (**usan**) comportamiento de otras clases
"es un"	El objeto de nuestra clase **es** una extensión o especialización de la clase de la cual hereda.

Que relaciones de objetos veremos?

Trataremos problemas sencillos que utilizan

- **SUCESIONES**
- **PROGRESIONES o SERIES**
- **SECUENCIAS.**

Definiciones:

Sucesión: Un término después de otro.
No hay una ley de vinculación entre ellos.
Típicamente necesitamos del término actual.

Progresión, serie: Un término después de otro.
Existe una ley de vinculación entre ellos.
Típicamente necesitamos del término actual y el anterior.

Secuencia: Un término después de otro.
No hay una ley de vinculación predefinida entre ellos.
En cualquier momento se debe disponer de todos o cualquiera de sus términos.

COMPOSICION USANDO UNA SUCESION DE NUMEROS

Enunciado: Producir una sucesión de números aleatorios y listarlos.
El listado debe concluir con el primer número divisible inclusive.

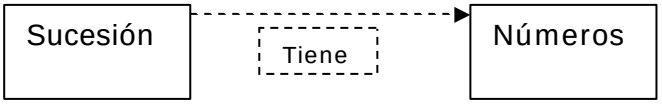
Conceptos	**Responsabilidades**
numero:	generación aleatoria, divisibilidad.
sucesion:	producir la sucesión de números, detener esta producción.

Tenemos dos conceptos,
implementaremos dos clases,
cuales serán las

Relaciones entre conceptos?

Un número **es** una sucesión?	**no**
Una sucesión **es** un número?	**no**
Un número **tiene** una sucesión?	**no**
Una sucesión **tiene** números?	**Si**

El objeto de la clase Sucesión **tiene** objetos Número.



```
class Numero {
        private int valor;      // valor del numero
        private boolean multip;       // es o no multiplo?
        public void setValor(int x)  {  // Genera valor aleatoriamente
                valor=(int)(100*Math.random());
                multip = multiplo(x);
        }
        public int getValor(){  // retorna el valor del numero
                return valor;
        }
        public boolean multiplo(int x){
                boolean retorno = false;
                if (valor % x == 0) // es multiplo
                        retorno = true;
                return retorno;
        }
        public String toString(){  //  hilera con informaci¢n del objeto
                String aux = "Numero ";
                if(multip) aux += "divisible ";
                else       aux += "no divis. ";
                aux += valor;
                return aux;
        }
}  // fin clase numero.

import Numero;
class Sucesion{
        Numero numero;              // Referencia a Numero
        int    multip;              // valor para chequear multiplicidad

        public Sucesion(int x){  // Constructor de Sucesion
                numero  = new Numero();  // el constructor  de Numero
                multip= x;
        }
        public void proSucesion(int x){
                do{
                   numero.setValor(multip);
                   System.out.println(numero.toString());
                }while(numero.multiplo(multip));
        }
}
```

```
Deseo procesar una sucesion
de numeros divisibles por 3
Numero divisible 9
Numero divisible 84
Numero divisible 6
Numero no divis. 61
Process Exit...
```

```
import In;
import Sucesion;
public class PrueSuc{
        public static void main(String args[]){
                System.out.println("\nDeseo procesar una sucesion");
                System.out.print("de numeros divisibles por ");
                int aux = In.readInt();
                Sucesion suc = new Sucesion(aux);  // Instanciamos la sucesión
                suc.proSucesion(aux);              //  y la procesamos ...
        }
} // PrueSuc
```

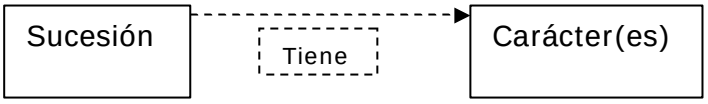
COMPOSICION USANDO UNA SUCESION DE CARACTERES

Enunciado: Procesar una sucesión de caracteres.
Cuantos de ellos son vocales, consonantes y dígitos decimales?.
El proceso concluye con la lectura del carácter '#' (numeral)

analizamos **"**sucesión de caracteres" , **dos conceptos:**

caracter: deteccion de que caracteres son letras, dígitos, mayúsculas, etc, etc.
sucesión: ciclo de lectura, contabilización, detección de su fin.

El objeto de la clase sucesión **tiene** objetos Carácter




```
public class Caracter{
    private int car;        // Parte interna de la clase, nuestro caracter

    Caracter(){car=' ';}        // Constructor, inicializa car en ' '

    Caracter(int cara){car = cara;};        // Constructor, inicializa car
      mediante parámetro

    boolean esLetMay(){             // Es letra mayúscula ?
      boolean mayus = false;
      String mayu = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      for(int i=0;i < mayu.length();i++)
            if(mayu.charAt(i)==car) mayus = true;    //    Es una letra
      mayúscula
      return mayus;
    }

    boolean esLetMin(){             // Es letra minuscula ?
      boolean minus = false;
      String minu = "abcdefghijklmnopqrstuvwxyz";
      for(int i=0;i < minu.length();i++)
            if(minu.charAt(i)==car) minus = true;  // Es una letra minúscula
      return minus;
    }
    boolean esLetra(){              // Retorna true si verdadero, false caso
      contrario
      boolean letra = false;
      if(esLetMay()) letra=true;
      if(esLetMin()) letra=true;
      return letra;      // Retornemos lo que haya sido
    } // esLetra()

    boolean esVocal(){...           // Es vocal ?
    boolean esConso(){...           // Es consonante
    boolean esDigDec(){...          // Es dígito decimal ?
    boolean esDigHex(){...          // Es dígito hexadecimal ?
    boolean esSigPun(){...          // Es signo de puntuación ?
    boolean lecCar() throws java.io.IOException{...    //    Lectura de
      caracteres
    int getCar(){...
    void setCar(int cara){...
};
```

```
import java.io.*;
import Caracter;
public class SuceCar{
        private         int vocal, conso, digDec; //  Contadores
        private Caracter car;
        public SuceCar(){       // Constructor
                vocal = 0;conso = 0; digDec = 0;
                car = new Caracter();
                System.out.println ("\nIntroduzca una sucesi¢n de ");
                System.out.println ("caracteres, finalizando con #\n");
        }
        public void proSuc() throws java.io.IOException{ // Cont. secuencia
                while (car.lecCar()){ // Mientras no sea '#'
                        if(car.esVocal()) vocal++;
                        if(car.esConso()) conso++;
                        if(car.esDigDec())digDec++;
                }
                System.out.println (this); // Exhibimos atributos del objeto

        }
        public String toString(){       //  Metodo para exhibir el objeto
                String aux = " \n";         //  Una l¡nea de separacion
                aux += " Totales proceso \n";                   //  Un titulo
                aux += "Vocales      " + vocal + "\n";
                aux += "Consonantes  " + conso + "\n";
                aux += "Digitos dec. " + digDec+ "\n";
                aux += "Terminado !!!\n";
                return aux;
        }
};
```

```
Introduzca una sucesi¢n de
caracteres, finalizando con #

Aqui me pongo a cantar ...#
Totales proceso
Vocales      9
Consonantes  9
Digitos dec. 0
Terminado !!!
Process Exit...
```

```
import SuceCar;
import java.io.IOException;
public class PruebaSuc{
        public static void main(String args[]) throws IOException{
                SuceCar suc = new SuceCar();
                // Construimos el objeto suc de la clase Sucesion,
                suc.proSuc();               // lo procesamos
        }
}
```

COMPOSICION USANDO UNA PROGRESION DE CARACTERES

Enunciado: Procesar una **progresión** de caracteres.
Cuantas alternancias consonante/vocal o viceversa ocurren.
Fin: lectura del carácter '#' (numeral).

Porque hemos cambiado de **Sucesión para Progresion**?

Para detectar alternancia, necesitamos de **dos objetos Carácter**.
El anterior y el actual.
Progresión es el **concepto** que debemos modelar.

```
import Caracter;
class  Progresion{
        private int siAlt, noAlt;               //  Contadores de alternancias y no alt.
        private Caracter carAnt, carAct;
        public Progresion(){                    //  Constructor,
                siAlt=0; noAlt=0;                       //  Contadores a cero
                carAnt =  new Caracter();
                carAct =  new Caracter();
                System.out.println("\nIntroduzca caracteres, fin: #");
        }

        public void cicloCar(){
                carAnt.setCar(carAct.getCar());     //  carAnt ← carAct
        }

        public boolean exiAlter(){                      // Existe alternancia ?
                boolean alter = false;
                if((carAnt.esConso() && carAct.esVocal()) // Si la hubo asi
                || (carAnt.esVocal() && carAct.esConso())) // o de esta otra manera
                        alter = true;
                return alter;   // contabilizaremos que no
        }

        public void proAlter() throws java.io.IOException{
                do{
                        carAct.lecCar();
                        if(exiAlter()) siAlt++;                 // Detectamos alternancia
                        else                noAlt++;    // No la hubo
                        cicloCar();
                }while (carAct.getCar() != '#');     // Mientras no sea '#'
                System.out.println(this);
        }

        public String toString(){
                String aux = "\n Totales \n";
                aux += "Alternan " + siAlt + " \n";
                aux += "No alter " + noAlt + " \n";
                aux += "Terminado !!!\n";
                return aux;
        }
};
```

Documentando

En el método **public void cicloCar()** tenemos la expresión

```
carAnt.setCar(carAct.getCar());
```

Reconocemos que a primera vista es bastante criptica
El objeto **carAnt** invoca el método **setCar(…)**,
quien recibe como argumento el retorno de la la
expresión **carAct.getCar()**.

*No sería mas sencillo, simplemente **carAnt = carAct** y listo?*

Lamentablemente no lo es.
Estamos igualando dos referencias
(direcciones de memoria)
a los objetos Carácter.
Lo que la expresión hace es que ambas apunten al objeto **carAnt**,
entonces el método exiAlter() jamas detectaría ninguna alternancia,
estaría siempre comparando el anterior con si mismo.

*- de acuerdo, pero podria funcionar se hacemos **carAnt.car = carAct.car?***

Buena pregunta. Veamos que ocurre. Reemplazamos el método por la expresión que Ud propone, compilamos y

```
public void cicloCar(){
    carAnt.car = carAct.car;
}
```

```
utput  Build  Find in Files
E:\jdk1.3\bin\javac.exe  Progresion.java
Working Directory - E:\Tymos\Catedras\AED2005\Unidad II
Class Path - .;E:\Kawa4.01\kawaclasses.zip;e:\jdk1.3\li
File Compiled...

---------------------------- Compiler Output ------------
Progresion.java:17: car has private access in Caracter
            carAnt.car = carAct.car;
                         ^
Progresion.java:17: car has private access in Caracter
            carAnt.car = carAct.car;
                               ^
2 errors
```

El compilador nos informa que **car es inaccesible**.
Correcto, en la clase carácter lo hemos declarado privado.
Estamos en otra clase, no lo podemos acceder.
Podriamos “suavizar” el encapsulado y retirarle a car el nivel private.
Pasaría a ser friendly, …Recompilamos Carácter, Progresion …
Compilación OK. La ejecución tambien.

```
import Progresion;
import java.io.IOException;
class PrueProg{
    public static void main(String args[]) throw
        Progresion pro = new Progresion();
        pro.proAlter();
    }
}
```

```
Introduzca caracteres, fin: #
Es la ultima copa ...#
Totales
Alternan 9
No alter 12
Terminado !!!
Process Exit...
```

MÁS UNA COMPOSICION USANDO UNA PROGRESION DE CARACTERES

Tratamiento de frases

(Cuantas palabras, cuantos caracteres, longitud promedio de la palabra)

Enunciado: Procesar una progresión de caracteres.
Necesitamos conocer cuantas palabras contiene,
cuantos caracteres en total y la
longitud promedio de la palabra.
Fin: carácter '#' (numeral).

Tenemos frases constituidas por caracteres que forman palabras.

Que comportamiento debemos modelar?
detectar si el carácter leido pertenece a una palabra
detectar si la palabra ha concluido.

El profe , ... dice el alumno, ... es un burro#
Pertenece a la palabra

El profe , ... dice el alumno, ... es un burro#
No pertenece, la palabra ya terminó

El profe, ... dice el alumno, ... es un burro#
Un separador adicional

La estrategia: **ocurre un fin de palabra cuando el carácter actual no pertenece a la palabra, pero el anterior si** (Es letra o dígito)..

CLASE FRASE

La clase **Frase,** usando comportamiento de la clase Caracter:
- Contabilizará palabras.y caracteres.
- Calculará el promedio.
- Mostrará resultados

```
import Caracter;
public class Frase{
    private Caracter carAct, carAnt;
    private int contCar, contPal;

    public Frase(){                    // Constructor,
      carAct = new Caracter(); //    instanciamos sus
      carAnt = new Caracter(); //    objetos atributos
      contCar = contPal = 0;
      System.out.println("\nIntroduzca caracteres, fin: #\n");
    }

    public void cicloCar(){
      carAnt.setCar(carAct.getCar()); //    Anterior <== Actual
    }


    public boolean actLetDig(){  // El actual es letra o digito ?
      boolean letDig = false;
      if  (carAct.esLetra() || carAct.esDigDec())  // Pertenece a la palabra
```

```
            letDig = true;
        return letDig;
    }

    public boolean antLetDig(){  // El anterior es letra o digito ?
        boolean letDig = false;
        if  (carAnt.esLetra() || carAnt.esDigDec()) // Pertenece a la palabra
            letDig = true;
        return letDig;
    }

    public boolean finPal(){              // Es fin palabra?
        boolean fPal = false;
        if  (antLetDig() && !actLetDig())
            fPal = true;
        return fPal;
    }

    public void proFrase() throws java.io.IOException{
            // Nuestro ciclo de lectura de caracteres
        do {
            carAct.lecCar();
            if (actLetDig()) contCar++;
            if (finPal())      contPal++;
            cicloCar();
        }while(carAct.getCar() != '#');       // Mientras no sea un caracter '#'

        System.out.println(this);
    }


  public String toString(){   // la exhibición de los resultados
        float         auxPro = (float)contCar/contPal;
        String aux = "\n Totales \n";
        aux += "Procesamos frase con " + contPal + " palabras,\n";
        aux += "constituidas por     " + contCar  + " caracteres, \n";
        aux += "su longitud promedio " + auxPro  + " caracteres \n";
        aux += "Terminado !!!\n";
        return aux;
    }
};
```

```
Introduzca caracteres, fin: #
to be or not to be, thas is the ...#

Totales
Procesamos frase con 9 palabras,
constituidas por     22 caracteres,
su longitud promedio 2.4444444 caracteres
Terminado !!!

Process Exit...
```

```
import Frase;
import java.io.IOException;
class PrueFras{
    public static void main(String args[]){
            Frase frase = new Frase();
            frase.proFrase();
    }
}
```

*REFERENCIAS Y ARRAYS (*Vectores de objetos)

Java dispone de arrays de tipos primitivos o de clases.
Vamos a hacer que la clase **Hotel2** tenga un objeto que sea un arreglo (array) de referencias a la clase **Habitación** y **que** tenga un método capaz de informarnos sobre sus comodidades.

```
public class Habitacion  {
        private int numHabitacion;  private int numCamas;

        public Habitacion() {       // Constructor sin argumentos
                numHabitacion       = 0;
                numCamas            = 0;
        }
        public Habitacion( int numHab,int numCam) {//Sobrecarga del constructor
                numHabitacion       = numHab;
                numCamas            = numCam;
        }
         public Habitacion(Habitacion hab) {// Mas sobrecarga del constructor
                numHabitacion       = hab.numHabitacion;
                numCamas            = hab.numCamas;
        }
        public int getNumHab() {//      Metodo público
                return numHabitacion;
        }
        public int getNumCam() {//      Método público
              return numCamas;
        }
        public void setNumHab (int numHab){//      Método público
                numHabitacion =  numHab;
        }
        public void setNumCam (int numCam){ //     Método público
                numCamas    = numCam;
        }
}       // public class Habitacion
```

```
import Habitacion;
public class Hotel2 {
        private int cantHab;  //       Cantidad de habitaciones de nuestro hotel
        private Habitacion habitacion[];//  Una referencia al array de habitaciones

        public Hotel2(int cant){    //  Un constructor del hotel
                cantHab = cant;
                habitacion = new Habitacion[cantHab]; // Obtenemos el array ...
                int canCam;                           // Una variable de trabajo
                for(int i = 0; i < cantHab; i++){
                        if(i < 3) canCam = 2;
                        // Las tres primeras habitaciones tienen 2 camas,
                        else      canCam = 3;               // el resto 3 camas
                        habitacion[i] = new Habitacion(i+1,canCam);
                        // Instanciamos cada objeto habitación
                }

        }       //      public Hotel2
```

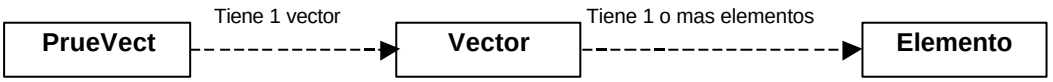
```
        public void Comodidades(){
                System.out.println("Nuestro hotel dispone de las siguientes
                                        habitaciones");
                for(int i = 0; i < cantHab; i++)
                        System.out.println("Hab. N " + habitacion[i].getNumHab() +
                                                " con " + habitacion[i].getNumCam() + "
                                                camas");
        }
}       //      public class Hotel2
```

```
Nuestro hotel dispone de las …
Hab. N 1 con 2 camas
Hab. N 2 con 2 camas
Hab. N 3 con 2 camas
Hab. N 4 con 3 camas
Hab. N 5 con 3 camas
Hab. N 6 con 3 camas
Nada mas tenemos para ofrecerle ...
Process Exit...
```

```
import Hotel2;
public class DemoHotel2 {
        public static void main( String args[] ) {
                Hotel2 miHotel = new Hotel2(6); // Instanciamos un objeto Hotel2 con 6
                                                habitaciones
                miHotel.Comodidades();      //  Mostramos sus comodidades
                System.out.println("Nada mas tenemos para ofrecerle ...");
        }
}
```

COMPOSICION USANDO UN VECTOR DE ELEMENTOS

Encontrar el menor de los elementos numéricos del vector tiene mucho que ver con el vector en si.
No es una propiedad de un Elemento aislado. Solo tiene sentido si existen otros contra los que puede comparase.




```
import In;
class Elemento{
        private int valor;      // atributos instanciables de la clase

        public Elemento(int ind){  // carga desde el teclado el valor de un numero
                System.out.print("Valor del elemento ["+ind+"]");
                valor = In.readInt();
        }

        public int getValor(){  // retorna al mundo exterior el valor del numero
                return valor;
        }
}  // fin clase Elemento.

import Elemento;
class Vector{
        Elemento element[];  // vector de objetos Elemento
        private int orden = 0, menor;       // atributos no instanciables
```

```
        public Vector(int tamanio){
                element = new Elemento[tamanio]; // crea un vector de Elemento del
                                                 tamaño informado
                for(int i=0;i<element.length;i++)
                        element[i] = new Elemento(i); // Construye cada uno de los
                                                      objetos Elemento
                menor = element[0].getValor();
        }

        public String toString(){
                int auxVal;
                String aux = "El arreglo contiene \n";
                aux       += "Ord. Val\n";
                for(int i =0;i<element.length;i++) {
                        auxVal =  element[i].getValor();   //  obtenemos el valor
                        aux += "["+i+"]  "+auxVal+"\n";  //  lo incorporamos a la hilera
                        detMenor(auxVal,i);          //  vemos si es el menor
                }
                aux += "\ny su menor elemento es el orden " + getOrden() + "\n";
                return aux;
        }
        private void detMenor(int val, int ind){
                        if (val < menor){
                        menor = val;
                        orden = ind;
                }
        }
        private int getOrden(){
                return orden;
        }
}
```

```
Cuantos elementos tendra el vector? 5
Valor del elemento [0]10
Valor del elemento [1]20
Valor del elemento [2]30
Valor del elemento [3]9
Valor del elemento [4]33
El arreglo contiene
Ord. Val
[0]  10
[1]  20
[2]  30
[3]  9
[4]   33
y su menor elemento es el orden 3
Process Exit...
```

```
import  Vector;
public class PrueVect{
        public static void main(String args[]){
                System.out.print("\nCuantos elementos tendra el vector? ");
                int aux = In.readInt();
                Vector vect = new Vector(aux);     // Creamos el objeto VectorNum
                System.out.println(vect.toString());
        }
}
```

COMPOSICION USANDO UNA SECUENCIA DE NUMEROS (Silvio Serra)

Enunciado: Cuantos números de una secuencia son múltiplos de n ?

Cuales son los objetos ? Cuales sus atributos y comportamiento ?

NUMERO

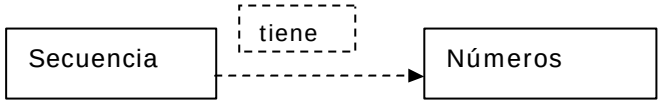
Atributos:
- Valor

Responsabilidades:
- Numero()
- InformarValor()
- CargarValor()
- Multiplo(x)

SECUENCIA

Atributos:
- valores
- tamaño

Responsabilidades:
- Secuencia(x)
- CargarValores()
- CuantosMultiplos(x)




```
import java.io.*;
import In;
class numero{
        private int valor;  // atributo de la clase
        public numero(){valor = 0;}  // constructor.

        public void CargarValor(){valor=In.readInt();}

        public int InformarValor(){return valor;}

        public boolean Multiplo(int x){  // retorna si el numero x es o no multiplo
                boolean retorno = false;
                if (valor % x == 0) // es multiplo
                        retorno = true;
                return retorno;
        }
}  // fin clase numero.
```

```
class secuencia{
        numero valores[];  // vector de numeros
        int tamaño;

        public secuencia(int x){  // crea un vecrtor de numeros de tamaño x
                valores = new numero[x];  // el vector de numeros
                for(int i=0;i<x;i++)
                  valores[i] = new numero();
                tamaño = x;  // el tamaño del vector
        }

        public void CargarValores(){  // carga los valores de la secuencia.
                System.out.println("\nTipee 5 numeros, separados por espacios\n");
                for(int i=0;i<tamaño;i++){
                        valores[i].CargarValor();
                }
        }
```

Una ejecucion
Tipee 5 numeros, separados por espacios

10 20 30 40 50
La cantidad de números multiplos de 4 es 2
Process Exit...

```
        public int CuantosMultiplos(int x){
                int retorno = 0;
                for(int i=0;i<tamaño;i++){
                        if(valores[i].Multiplo(x)==true)  // el numero es multiplo
                                retorno++;
                }
                return retorno;
        }
}       // clase secuencia
```

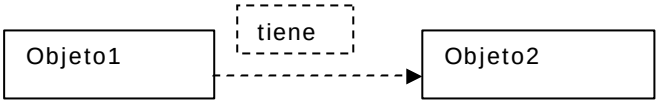
Otra
Tipee 5 numeros, separados por espacios

123 345 567 789 444
La cantidad de números multiplos de 4 es 1
Process Exit...

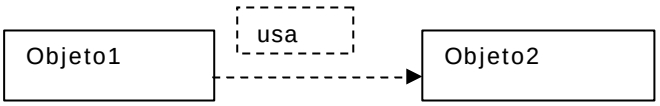
```
public class programasecuencia{
        public static void main(String args[]){
                secuencia UnaSecuencia = new secuencia(5);  // Array 5 objetos numero
                UnaSecuencia.CargarValores();  // Carga desde teclado la secuencia
                int cantidad;
                cantidad = UnaSecuencia.CuantosMultiplos(4); // retornara la cantidad
                                        // de números multiplos de 4 en UnaSecuencia.
                System.out.println("La cantidad de números multiplos de 4 es "
                                                                        +cantidad);
        }
} // programasecuencia.( Silvio Serra)
```

HERENCIA (Ilustración filmina 1)

Para modularizar, hemos visto las relaciones:



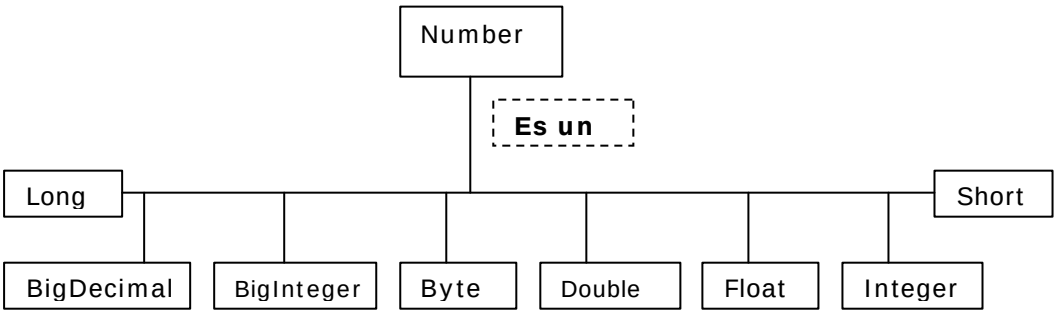
y



Esta capacidad **se amplía** mediante una técnica llamada ***herencia.***

. Se aplica en los conceptos organizables en una **estructura jerárquica**.
. **Clases genéricas se especializan en clases más particulares**.
. Estas **clases particulares** (subclases, clases derivadas)
  . **Heredan** los métodos generales (No se re escriben, se **rehúsan**)
  . **Redefinen** los métodos generales con < > implementación (**Polimorfismo**)
  . **Incorporan** los métodos que sean especializados para esta *subclase* particular

El mecanismo de la herencia establece una **nueva relación**.



public **abstract class Number extends Object** implements Serializable

public Number()
byte byteValue() Returns the value of the specified number as a byte.
abstract double doubleValue() Returns the value of the specified number as a double.
abstract float floatValue() Returns the value of the specified number as a float.
...

Las subclasses de Number implementan métodos para convertir su representación a:

byte, double, float, int, long, and short.

Por ejemplo:
public **final class Double extends Number** implements Comparable

byte **byteValue()** Retorna el valor del Double como Byte. // **Método Redefinido**
// **métodos incorporados**
int compareTo(Double anotherDouble) Compara dos Doubles numericamente.
int **compareTo(Object o)** Compares this Double a otro Object.
...
// (hay mas 18 métodos incorporados)

```
public class Madre{
        protected int x;
        public Madre(int v){x=v;}
        public int getValx(){return x;}
        public String toString(){
                String aux = "Soy Madre, valgo: "+x;
                return aux;
        }
}

class Hija extends Madre{
        protected int y,z,w;
        public Hija(int v,int v1,int v2,int v3){
                super(v);y=v1;z=v2;w=v3;
        }
        public int getValy(){return y;}
        public int getValz(){return z;}
        public int getValw(){return w;}
        public String toString(){
            String aux = "Soy Hija, valgo: "
                  +x+","+y+","+z+","+w;
            return aux;
        }
}

class PruHerencia{
        public static void main(String args[]){
                Madre madre = new Madre(100);
                System.out.println(madre);
                System.out.println("Valor de x "+madre.getValx());
                Hija hija = new Hija(10,20,30,40);
                System.out.println(hija);
                System.out.println("Valor de x "+hija.getValx());
                System.out.println("Valor de y "+hija.getValy());
                System.out.println("Valor de z "+hija.getValz());
                System.out.println("Valor de w "+hija.getValw());
        }
}
```

```
Soy Madre, valgo: 100
Valor de x 100

Soy Hija, valgo: 10,20,30,40
Valor de x 10
Valor de y 20
Valor de z 30
Valor de w 40

Process Exit...
```

Despacho dinámico

hija.getValx())

Trata de ejecutarse en el entorne de Hija, allí no existe.
Sube, trata de hacerlo en el entorne de Madre. Allí si lo encuentra.
Este algoritmo de **despacho dinámico**, es un mecanismo efectivo para localizar los programas que se reúsan.
También proporciona una poderosa técnica: el ***polimorfismo.***

POLIMORFISMO

En forma literal, "**polimorfismo**" significa "**muchas formas**".
La idea base es **invocar una acción genérica con un único mensaje**.
Se requiere de clases organizadas en una **estructura hereditaria**.
En la **clase ancestro siempre se puede referenciar descendientes**.
Un ejemplo:

```
public class Madre{
        protected int x,y;
        public Madre(int v1,int v2){x=v1;y=v2;}
        protected int getCalculo(){return x+y;}
        public String toString(){
                String aux = "Soy Madre, valgo: "
                                +getCalculo();
                return aux;
        }
}

class Hija extends Madre{
        public Hija(int v1,int v2){super(v1,v2);}
        protected int getCalculo(){return x-y;}
        public String toString(){
                String aux = "Soy Hija, valgo: "
                                +getCalculo();
                return aux;
        }
}

class Nieta extends Hija{
        public Nieta(int v1,int v2){super(v1,v2);}
        protected int getCalculo(){return x*y;}
        public String toString(){
                String aux = "Soy Nieta, valgo: "
                                +getCalculo();
                return aux;
        }
}

class PruPoli{
        public static void main(String args[]){
                Madre madre[] = new Madre[3];
                madre[0] = new Madre(10,20);
                madre[1] = new Hija (10,20);
                madre[2] = new Nieta(10,20);
                System.out.println("Probando Polimorfismo");
                for(int i=0;i<madre.length;i++)
                        System.out.println(madre[i]);
        }
}
```

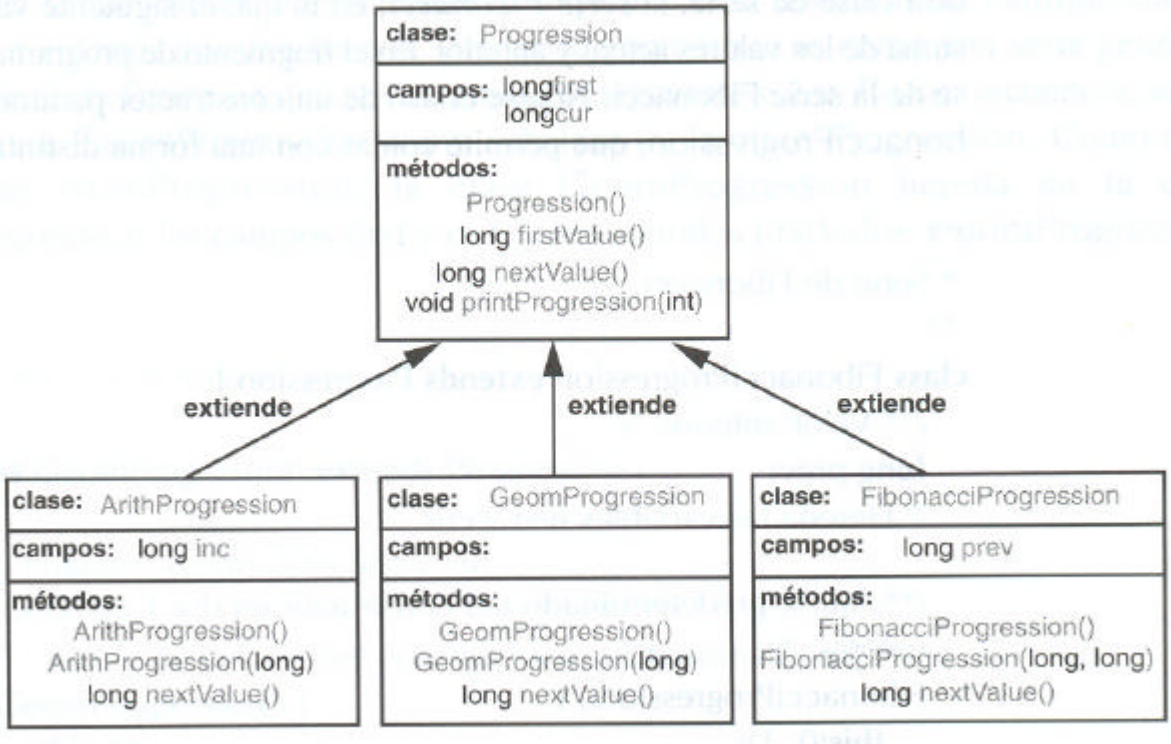
```
Probando Polimorfismo

Soy Madre, valgo: 30

Soy Hija, valgo: -10

Soy Nieta, valgo: 200

Process Exit...
```

Ejemplos de herencia en Java

```
public class Progression {
        protected long first;  // Primer valor de la serie
        protected long cur;   // Valor actual de la serie.
        Progression(){      // Constructor predeterminado
                cur = first = 0;
        }
        protected long getFirstValue(){ // reinicializar y regresar el primer valor
                cur = first; return cur;
        }
        protected long getNextValue(){// Retona corriente previamente avanzado
                return ++cur;
        }
        public void printProgression(int n){ // Imprime n primeros valores
                System.out.print(getFirstValue());
                for (int i = 2; i <= n; i++){
                        System.out.print(" " + getNextValue());
                }
        }
}

import Progression;
class ArithProgression extends Progression {
        //       Hereda las variables first y cur
        protected long inc;   // incremento
        ArithProgression() {//     Constructor, llama al parametrico
                this(1 );
        }
        ArithProgression(long increment) {// Constructor paramétrico
                super(); inc = increment;
        }
        protected long getNextValue(){        // metodo sobrescrito
                cur += inc;    return cur;  //        Avanza la serie
        }
        //      Hereda getFirsValue() y printProgression(..)
}

import Progresión;
class GeomProgression extends Progression {
        //       Hereda las variables first y cur
        GeomProgression(){      // Constructor, llama al parametrico
                this(2);
        }
        GeomProgression(long base){  // Constructor parametrico,
                super();
                first = cur = base;    // Define Primero y corriente
        }
        protected long getNextValue(){
                cur *= first;  // Avanza la serie multiplicando la base por el valor actual
                return cur;             //      regresar el siguiente valor de la serie
        }
        //      Hereda getFirstValue()
        //      Hereda printProgression(int).
}
```

Clase de la serie de Fibonacci

```
import Progression;
class FibonacciProgression extends Progression{
        long prev;      // Valor anterior
        // Hereda las variables first y cur
        FibonacciProgression(){ //  Constructor predeterminado
                this(0, 1);             // llama al parametrico
        }
        FibonacciProgression(long value1, long value2){//Const. paramétrico
                super();
                first = value1;         //      Definimos primer valor
                prev = value2 - value1; // valor ficticio que antecede al primero
        }
        protected long getNextValue() {         //      Avanza la serie
                long temp = prev;
                prev = cur;
                cur += temp;            //      sumando el valor anterior al valor actual
                return cur;             //      retorna el valor actual
        }
        // Hereda getFirstValue()
        // Hereda printProgression(int).
}
```



```
/** Programa de prueba para las clases de series */
import ArithProgression;
import GeomProgression;
import FibonacciProgression;
class Tester{
  public static void main(String[] args){ Progression prog;
        System.out.println("\n\nSerie aritmética con incremento predeterminado: ");
        prog = new ArithProgression();
        prog.printProgression(10);
        System.out.println("\n\nSerie aritmética con incremento 5: ");
        prog = new ArithProgression(5);
        prog.printProgression(10);

        System.out.println("\n\nSerie geométrica con la base predeterminada: ");
        prog = new GeomProgression();
        prog.printProgression(10);
        System.out.println("\n\nSerie geométrica con base 3: \n");
        prog = new GeomProgression(3);
        prog.printProgression(10);

        System.out.println("\n\nSerie de Fibonacci con valores predeterminados: ");
        prog = new FibonacciProgression();
        prog.printProgression(10);
        System.out.println("\n\nSerie Fibonacci con valores iniciales 4 y 6: ");
        prog = new FibonacciProgression(4,6);
        prog.printProgression(10);
  }
}
```

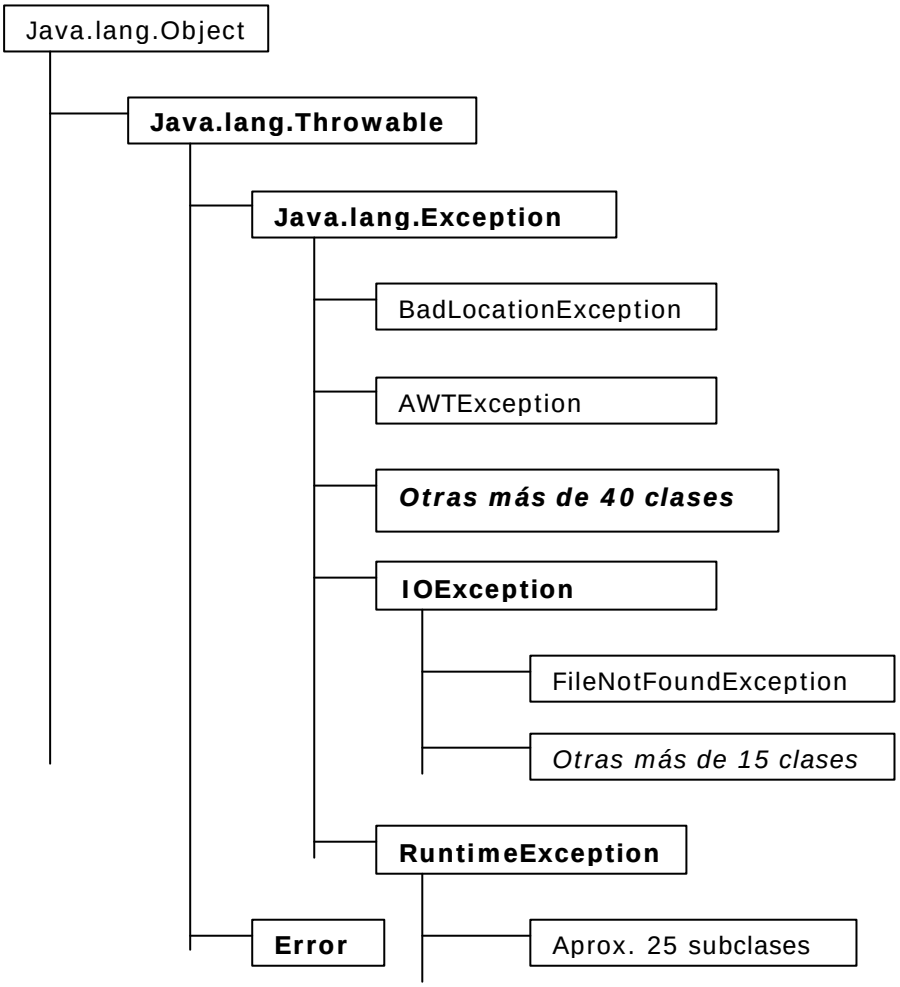
```
Serie aritmética con incremento predeterminado:
0 1 2 3 4 5 6 7 8 9

Serie aritmética con incremento 5:
0 5 10 15 20 25 30 35 40 45

Serie geométrica con la base predeterminada:
2 4 8 16 32 64 128 256 512 1024

Serie geométrica con base 3:
3 9 27 81 243 729 2187 6561 19683 59049

Serie de Fibonacci con valores iniciales predeterminados:
0 1 1 2 3 5 8 13 21 34

Serie Fibonacci con valores iniciales 4 y 6:
4 6 10 16 26 42 68 110 178 288
Process Exit...
```

EXCEPCIONES

Eventos inesperados que suceden durante la ejecución de un programa.

- una condición de error
- un dato no previsto

Las excepciones son objetos de clases que extienden **Java.lang.exception.**



Clase **Throwable,** que denota cualquier objeto que se puede lanzar y atrapar
Clase **Error** se usa para condiciones anormales en ejecución
Clase **Exception** es la raíz de la jerarquía de excepciones.
Clase **RuntimeException** se deben declarar en la cláusula **throws** de cualquier método que las pueda lanzar.

Excepciones comprobadas (o *síncronas),* lo que significa que el compilador comprueba que nuestros métodos lanzan sólo las excepciones que ellos mismos han declarado que pueden lanzar. Son la mayoría. Y se tratan.

Excepciones no comprobadas *(o asíncronas),* errores estándar en tiempo de ejecución, del tipo **RuntimeException** o **Error.** Un mensaje y a otra cosa.

Lanzamiento de excepciones

Las excepciones son objetos *"lanzados"* por una condición inesperada.
Por **nuestro código**, desborde de una array, división por cero…
Por el **ambiente de ejecución en Java**, se acabó la memoria, disco lleno…

Una excepción puede:

- Ser ***atrapada*** por el programa que la "maneja" de alguna manera
- El programa se termina en forma abrupta.

Hagamos un primer ejemplo.
Tomamos dos arreglos de elementos int de **distinta longitud,**
y hacemos un ciclo donde nume[i] es dividido por deno[i].
El denominador tiene **un 0 en el cuarto casillero**,
así que esperamos la **excepción por división por cero** …

```
class Excep01{                  / Su ejecución ➔
  static int nume[] = {10,20,30,40,50};
  static int deno[] =  {  2, 4,   6,  0,10,12};

  public static void excep() {
        int i;
        float coc;
        System.out.println("\nExcepciones, ej 01
        for(i = 0; i < deno.length; i++){
                coc = (float)nume[i]/deno[i];
                System.out.println("Cociente "+i+" vale "+coc);
        }
  }
  public static void  main(String args[]){excep();}
}
```

```
Excepciones, ej 01
Cociente 0 vale 5.0
Cociente 1 vale 5.0
Cociente 2 vale 5.0
Cociente 3 vale Infinity
Cociente 4 vale 5.0
java.lang.ArrayIndexOutOfBoundsException
        at Excep01.excep(Excep01.java:12)
        at Excep01.main(Excep01.java:18)
Exception in thread "main"
Process Exit...
```

Sorpresa!!!. La división por 0 no causa una excepción.
Si ocurre *java.lang.**ArrayIndexOutOfBoundsException***

Nos preocupa lo de la división por cero.

```
import java.io.IOException;
class Infinity{
        public static void infinito() {
                double a = 10.0, b = 0.0, c, d,e = 5.0, f,g;
                System.out.println("\nPreocupandonos por el Infinito\n");
                c = a+e;
                System.out.println("valor de c = "+c);
                d = a/b;        // Supuestamente Infinity
                System.out.println("valor de d = "+d);
                e+=d;           // Supuestamente Infinity
                System.out.println("valor de e = "+e);
                f = a - d;      // Supuestamente -Infinity
                System.out.println("valor de f = "+f);
                g = d/e;        // Que pasara ?
                System.out.println("valor de g = "+g);
        }
        public static void main(String args[]){infinito();}
}
```

```
Preocupandonos por el Infinito
 valor de c = 15.0
 valor de d = Infinity
 valor de e = Infinity
 valor de f = -Infinity
 valor de g = NaN
 Process Exit...
```

Atrapado de excepciones

Las excepciones se pueden atrapar
- En el método que la disparó.
- En un método de la pila de llamadas donde se llama al disparador.

Una excepción atrapada se puede analizar y manejar:
Codificamos dentro de un bloque **try** el código que puede disparar excepciones.
Si se dispara, el control salta al bloque **catch.**
Si alguna de las cláusulas **catch** es del tipo de la excepción disparada, la atrapamos.
Y en el bloque del catch correspondiente la tratamos.

Atrapando excepciones en el propio método

```
import java.io.IOException;
class Excep02{                          //          Ejecución ➔
        static int nume[] = {10,20,30,40,50};
        static int deno[] = { 2, 4, 6, 0,10,12};
        public static void excep() {
                int i;
                float coc;
                try{
                        System.out.println("\nExcepciones, ej 02");
                        for(i = 0; i < deno.length; i++){
                                coc = (float)nume[i]/deno[i];
                                System.out.println("Cociente "+i+" vale "+coc);
                }
                        System.out.println("Saliendo del bloque de prueba ...");
                }catch(ArrayIndexOutOfBoundsException e){
                        System.out.println("Excepcion capturada !!!");
                }
                System.out.println("Exit excep(), class Excep02");
        }
        public static void main(String args[]){
                excep();
        }
}
```

```
Excepciones, ej 02
Cociente 0 vale 5.0
Cociente 1 vale 5.0
Cociente 2 vale 5.0
Cociente 3 vale Infinity
Cociente 4 vale 5.0
Excepcion capturada !!!
Exit    excep(),    class
Excep02
Process Exit...
```

Que pasó?

Se produjo ArrayIndexOutOfBounds**Exception** dentro del bloque **try.**
Esa **exception** es capturada por **catch(ArrayIndexOutOfBoundsException e)**.

Como es la secuencia?

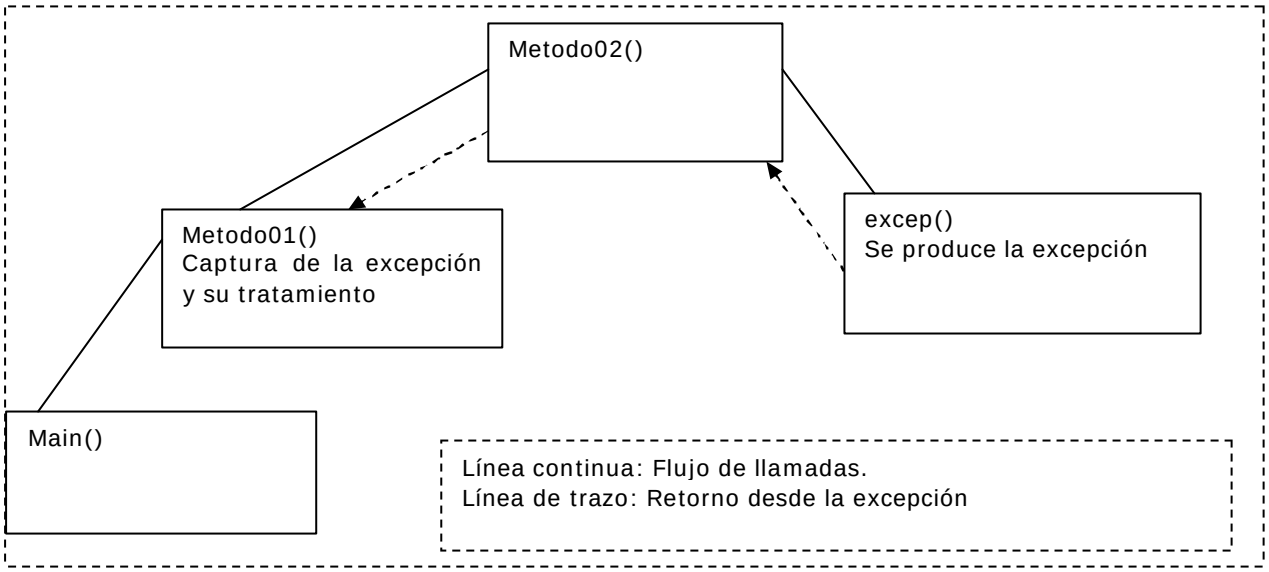
Se comienza ejecutando el **try{bloque_ }**.

Si esa ejecución **no genera excepciones**,
(Si hay **finally** su bloque se ejecuta)
➔ abajo del último **catch**

Si esa ejecución **genera excepciones**,
➔ **catch** con tipo compatible (Clase o subclase)
(Si hay **finally** su bloque se ejecuta)
se completa la ejecución de ese bloque catch,
➔ abajo del último **catch**

Si no hay catch compatible
(Si hay **finally** su bloque se ejecuta)
excepción ➔ al **método llamador**.

Atrapando excepciones “pasadas” desde métodos posteriores.




```
import java.io.IOException;
class Excep03{
        static int nume[] = {10,20,30,40,50};
        static int deno[] = { 2, 4, 6, 0,10,12};
        public static void Metodo01(){
                System.out.println("\nEntrada a Metodo01()");
                try{    //  en el bloque de prueba esta la llamada a
                        Metodo02();
                }catch(ArrayIndexOutOfBoundsException e){
                        System.out.println("Capturada en Metodo01()!!!");
                }
                System.out.println("Salida de Metodo01()");
        }
        private static void Metodo02(){
                System.out.println("Entrada a Metodo02()");
                excep();
                System.out.println("Salida de Metodo02()");
        }
        private static void excep() throws ArrayIndexOutOfBoundsException{
                int i;
                float coc;
                System.out.println("Entrada a excep()");
                for(i = 0; i < deno.length; i++){
                        coc = (float)nume[i]/deno[i];
                        System.out.println("Cociente "+i+" vale "+coc);
                }
                System.out.println("Salida de excep(), Excep03");
        }
        public static void main(String args[]){
                Metodo01();
        }
}
```

```
Entrada a Metodo01()
Entrada a Metodo02()
Entrada a excep()
Cociente 0 vale 5.0
Cociente 1 vale 5.0
Cociente 2 vale 5.0
Cociente 3 vale Infinity
Cociente 4 vale 5.0
Capturada en Metodo01()!!!
Salida de Metodo01()
Process Exit...
```

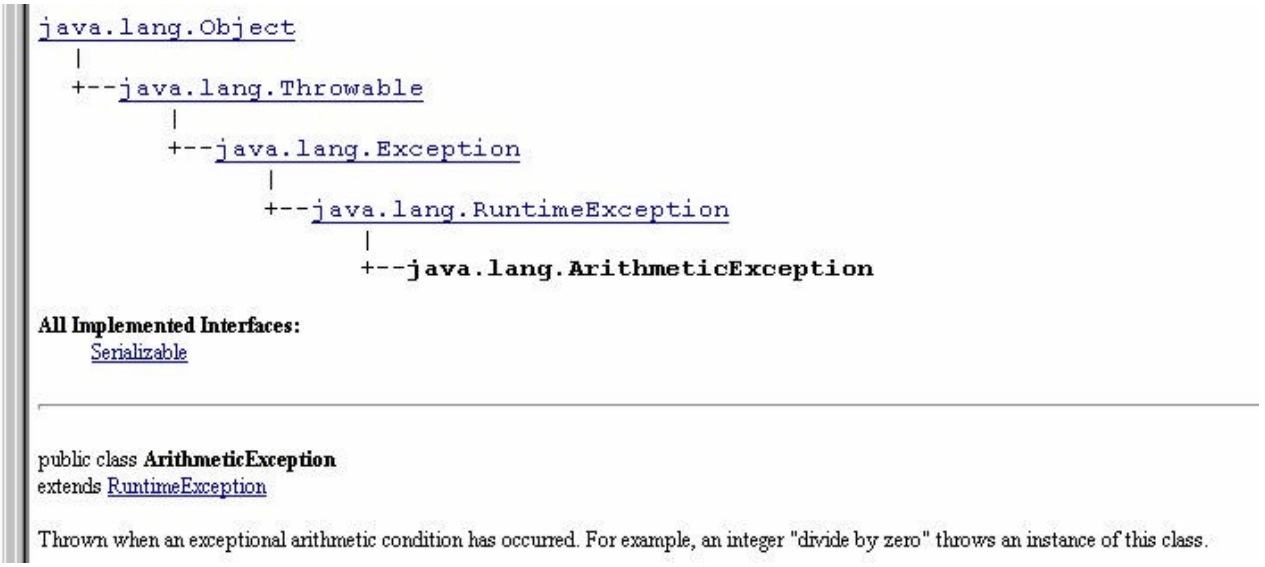
Donde **capturamos/tratamos** la excepción?

Si la excepción es especifica del método, allí mismo.

Si la producen muchos métodos lo más atrás posible.

Generando nuestras propias excepciones.

Del Help de java obtenemos:



Una excepción de división por cero es del tipo aritmético.
*Una división "**entero por cero**" dispara una instancia de esta clase.*
Solo la división por cero en números enteros produce excepción
Si queremos que esto ocurra con reales podemos disparar **nuestra** excepción.

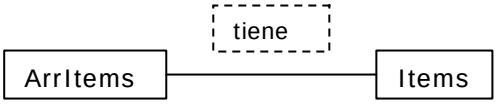
```
import java.io.IOException;
class RealDivideExcep extends ArithmeticException{
        static int nume[] = {10,20,30,40,50};
        static int deno[] = { 2, 4, 6, 0,10,12};

        public RealDivideExcep(){
                super();
        }
        public void excep() {  // Ejecución ➔
                int i; float coc;
                try{    System.out.println("\nExcepciones, ej 04");
                          for(i = 0; i < deno.length; i++){
                                if (deno[i] == 0)      //  La situación que nos preocupa
                                        throw new RealDivideExcep(); // Disparamos
                                coc = (float)nume[i]/deno[i];
                                System.out.println("Cociente "+i+" vale "+coc);
                          }
                        System.out.println("Saliendo del bloque de prueba ...");
                }catch(ArrayIndexOutOfBoundsException e){
                        System.out.println("Excepcion, ArrayIndexOut... capturada !!!");
                }catch(ArithmeticException e){
                        System.out.println("Excepcion,RealDivideExcep... capturada !!!");
                }
                System.out.println("Exit RealDivideExcep()");
        }
        public static void main(String args[]){
                RealDivideExcep rDivEx = new RealDivideExcep();
                rDivEx.excep();
        }
}
```

```
Excepciones, ej 04
Cociente 0 vale 5.0
Cociente 1 vale 5.0
Cociente 2 vale 5.0
Excepcion, RealDivideExcep... capturada !!!
Exit RealDivideExcep()
Process Exit...
```

BÚSQUEDA EN ARRAYS

Estamos comprando en un supermercado.
El operador de caja muestra al lector el código de barras.
El sistema le devuelve un valor que se imprime en el ticket.
Normalmente el valor no es lo que está codificado en las barras.
Lo que viene en el código de barras es el código del producto.
*El código de producto **está asociado** a su valor.*
Hay muchas maneras de almacenar esta asociación
*Por ejemplo, un **array de items** conteniendo asociaciones código /valor.*

***Buscar el código** en el array obteniendo su valor asociado.*



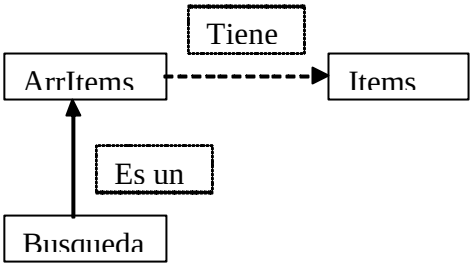

```
class Item {    // Una clase de claves asociadas a un valor ...
        protected int codigo;
        protected float valor;
        public Item(){};        // Constructor sin argumentos
        public Item(int cod, float val){ //  Constructor de la clase
                codigo=cod; valor=val;
        }
        public String toString(){
                String aux = "";
                aux+=codigo+" - "+valor;
                return aux;
        }       //      Exhibimos
        public int  getCodigo(){return codigo;}
        public float getValor(){return valor;}
        public Item  getItem(){return this;}
        public void  setCodigo(int cod){codigo=cod;}
        public void  setValor(float val){valor=val;}

        public boolean esMayor(Item item){
                //  Es mayor el obj. invocante que el parámetro ?
                return(codigo > item.codigo?true:false);
        }

        public boolean esMenor(Item item){
                return(codigo < item.codigo?true:false);
                //  Es menor el obj. invocante que el parámetro ?
        }

        public void  intercambio(Item item){
                Item burb= new Item(item.codigo,item.valor);
                // intanciamos burb con datos parametro
                item.codigo = this.codigo; // asignamos atributos del objeto invocante al
                item.valor  = this.valor;    // objeto parametro
                this.codigo = burb.codigo;  // asignamos atributos del objeto burb al
                this.valor  = burb.valor;     // objeto invocante
        }
}
```

```
import Item;
class ArrItems{           //      Una clase de implementación de un
      protected Item[] item;  // array de items, comportamiento mínimo ...
      protected int talle;   //  Tamaño del array de objetos Item
      public ArrItems(int tam, char tipo) {// Constructor de un array de Item's
            int     auxCod = 0;       // Una variable auxiliar
            talle=tam;                  // inicializamos talle (tamaño)
            item = new Item[talle];    // Generamos el array
            for(int i=0;i<talle;i++){     // la llenaremos de Item's, dependiendo
                  switch(tipo){          // del tipo de llenado requerido
                        case 'A':{     // los haremos en secuencia ascendente
                              auxCod = i;
                              break;
                        }
                        case 'D':{     // o descendente ...
                              auxCod = talle - i;
                              break;
                        }
                        case 'R':{     // o bien randomicamentente (Al azar)
                              auxCod = (int)(talle*Math.random());
                        }
                  }
                  item[i] = new Item(auxCod, (float)(talle*Math.random()));
            }
            System.out.print(this);
      }

      public String toString(){
            int ctos = (talle < 10 ? talle : 10);
            String aux = " Primeros "+ctos+" de "+talle+"\n elementos Item\n";
            for(int i=0;i<ctos;i++)
                  aux+=item[i].toString()+"\n";
            return aux;
      }
}
```

BUSQUEDA SECUENCIAL

Se compara cada elemento del array con la *clave* de búsqueda.
La clave puede encontrar igual en la primera, última, cualquiera o ninguna.
Este método es adecuado con arrays pequeños o no ordenados.

```
import ArrItems;
class Busqueda extends ArrItems{// Busqueda secuencial en un array de items
    int posic;  int clav;                          //  Posicion del ultimo encuentro
    public  Busqueda(int cant, char tipo) {        //       Un constructor
        super(cant,tipo);                          // que invoca otro
        posic = talle+1;                           //  posici¢n fuera del array
        clav  = 0;
    }

    protected void setClav(int clave){
        clav = clave;
    }
    protected int getClav(){
        return clav;
    }

    protected int leerClav(){
        System.out.print("Que clave? (999: fin) ");
        clav = In.readInt();
        return clav;
    }

    protected boolean existe(int clave){ ;      // Existe la clave par
        boolean exist = false;
        for(int i=0;i<talle;i++)
                if (clave==item[i].getCodigo()){
                   posic = i; exist = true;
                }
        return exist;
    }

    protected int cuantas(int clave) { ; // Cuantas veces existe la c
        int igual=0;                           //  Cuantos iguales ...
        for(int i=0;i<talle;i++)
                if(clave==item[i].getCodigo())igual++;
        return igual;
    }

    protected void demoBusqueda(){
        int cant;
        while (leerClav() != 999){
          cant =cuantas(clav);
          if(existe(clav))
            System.out.println("Código " + clav + ", existe " + cant+"
          else
            System.out.println("Codigo " + clav + " inexistente");
        }
        System.out.println("Terminamos !!!\n");
    }
};
class  pruBusSeq{
        public static void main(String args[]){
                Busqueda busq = new Busqueda(100,'R');
                busq.demoBusqueda();
        }
}
```




```
Primeros 10 de 100
 elementos Item
13 - 91.50563
58 - 69.4227
84 - 8.443384
42 - 77.41279
71 - 99.434616
87 - 66.86657
96 - 43.86491
88 - 6.7017875
52 - 26.867092
75 - 91.86902
Que clave? (999: fin)
71
Código 71, existe 1
veces
Que clave? (999: fin)
52
Códig
Que clave? (999: fin)
58
Código
Que clave? (999: fin)
999
Terminamos !!!

Process Exit...
```

BUSQUEDA BINARIA // Jueves 12 continuamos, si Deuz …

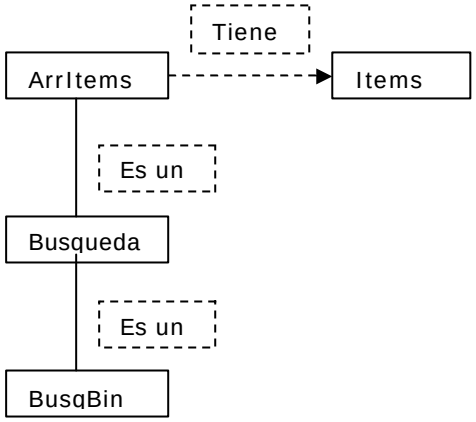
Ejemplo típico: la búsqueda de un número en un directorio telefónico.
Requiere del array ordenado.
Nos posicionamos en el centro del array. Puede ocurrir:

- Nuestra **clave coincide** con la del elemento.
- **La clave es mayor**: debemos buscar en el tramo superior.
- **La clave es menor**: la búsqueda será en el tramo inferior.

y este razonamiento se aplicará las veces necesarias.

```
import Busqueda;
class BusqBin extends Busqueda{      //Busqueda Binaria en un array de items
        public  BusqBin(int cant, char tipo){      // Un constructor
                super(cant,tipo);                  // llama al de la clase ancestro
        }
        protected boolean existe(int clave){;   // Existe la clave parámetro ?
                boolean exist = false;
                int alt=talle-1,baj=0;
                int indCent, valCent;
                while (baj <= alt){
                        indCent = (baj + alt)/2;      // índice de elemento central
                        valCent = item[indCent].getCodigo();
                                // valor del elemento central
                        if (clave == valCent){       // encontrado valor;
                           posic = indCent; exist = true; break;
                        }
                        else if (clave < valCent)
                                        alt = indCent - 1;   //  ir a sublista inferior
                                else  baj = indCent + 1;     // ir a sublista superior
                }
                return exist;
        };
        protected int cuantas(int clave){;        // Cuantas veces existe la clave ?
                int igual=0; boolean exist;
                if (exist= existe(clave)){     //Si la clave existe, puede estar repetida
                        for(int i=posic;i<talle;i++)//  para arriba
                                if (clave==item[i].getCodigo())
                                        igual++;
                                else break;
                        for(int i=posic-1;i>0;i--)    //       tambien para abajo
                                if (clave==item[i].getCodigo())
                                        igual++;
                                else break;
                }       // if (exist= existe(clave))
                return igual;
        }
};

import BusqBin;
class  pruBusBin{
        public static void main(String args[]){
        BusqBin busq = new BusqBin(100,'A');
        // Generamos un array de 100 objetos item ascenden
                busq.demoBusqueda();
        }
};
```

```
Primeros 10 de 100
 elementos Item
0 - 94.08064
1 - 70.68094
2 - 40.423695
3 - 76.080284
4 - 84.71842
5 - 33.304916
6 - 9.811888
7 - 62.708015
8 - 66.173584
9 - 46.92478
Que clave? (999: fin) 11
Código 11, existe 1 veces
Que clave? (999: fin) 25
Código 25, existe 1 veces
Que clave? (999: fin)
Código 0, existe 1 veces
Que clave? (999: fin) 101
Codigo 101 inexistente
Que clave? (999: fin) 999
```



Comparación de la búsqueda binaria y secuencial

Números de comparaciones considerando el peor caso

Tamaño array	Búsqueda binaria	Búsqueda secuencial
1	1	1
10	4	10
1.000	11	1.000
5.000	14	5.000
100.000	18	100.000
1.000.000	21	1.000.000

ORDENAMIENTO - Introducción

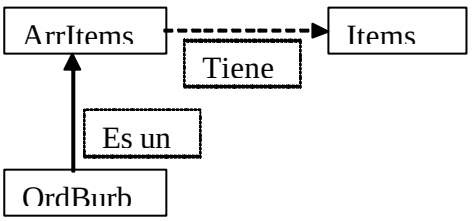
Algoritmos de ordenamiento básicos y mejorados.

- Los básicos, de codificación relativamente simple.
- Los mejorados, de muy alta eficiencia en su tarea de ordenar.

Ordenamiento por Método Burbuja.

Muy popular entre los estudiantes de programación.
El menos eficiente de todos.
Técnica: **ordenación por hundimiento**
Se comparan elementos de a pares.
El método realiza n-1 pasadas.
No detecta si el array ya esta ordenado.

```
import ArrItems;
public class OrdBurb extends ArrItems{      //      Ordenamiento p/Burbuja
      public OrdBurb(int cant, char tipo){      // Un constructor
            super(cant,tipo);
            System.out.println("\nPasadas de ordenamiento");
      }

      public String toString(){          // Mostrando los valores del array
            String aux  = "";
            int cuant=(talle < 10 ? talle : 10);  // Lo que sea menor
            for (int i=0;i<cuant;i++)
                  aux += item[i].getCodigo()+", ";
            return aux;
      }
      void ordenar(boolean trace){; // El método de ordenamiento
            int i,j;
            for (i = 1; i<talle; i++){    // Indicando talle - 1 pasadas
                  for (j = talle-1; j>=i; j--) // realizando la pasada
                        if (item[j].esMayor(item[j-1]))
                              // Si invocante es mayor que parámetro
                              item[j].intercambio(item[j-1]); // intercambio
                  if (trace)
                        System.out.println(this);  // array despues de la pasada
            }

      }
}
```




```
import OrdBurb;
class PrueBurb{
      public static void main(String args[]){
            System.out.println("\nOrdenamiento metodo Burbuja");
            OrdBurb burb  = new OrdBurb(10,'R'); // array de 10 objetos item random
            burb.ordenar(true);              // y lo ordenamos, trazando
      }
};
```

```
Ordenamiento metodo Burbuja
1, 0, 2, 8, 9, 1, 2, 7, 5, 8,
Pasadas de ordenamiento
9, 1, 0, 2, 8, 8, 1, 2, 7, 5,
9, 8, 1, 0, 2, 8, 7, 1, 2, 5,
9, 8, 8, 1, 0, 2, 7, 5, 1, 2,
9, 8, 8, 7, 1, 0, 2, 5, 2, 1,
9, 8, 8, 7, 5, 1, 0, 2, 2, 1,
9, 8, 8, 7, 5, 2, 1, 0, 2, 1,
9, 8, 8, 7, 5, 2, 2, 1, 0, 1,
9, 8, 8, 7, 5, 2, 2, 1, 1, 0,
9, 8, 8, 7, 5, 2, 2, 1, 1, 0,
Process Exit...
```

Una pasada innecesaria.

```
Ordenamiento metodo Burbuja
2, 1, 7, 3, 1, 9, 5, 2, 0, 8,
Pasadas de ordenamiento
9, 2, 1, 7, 3, 1, 8, 5, 2, 0,
9, 8, 2, 1, 7, 3, 1, 5, 2, 0,
9, 8, 7, 2, 1, 5, 3, 1, 2, 0,
9, 8, 7, 5, 2, 1, 3, 2, 1, 0,
9, 8, 7, 5, 3, 2, 1, 2, 1, 0,
9, 8, 7, 5, 3, 2, 2, 1, 1, 0,
9, 8, 7, 5, 3, 2, 2, 1, 1, 0,
9, 8, 7, 5, 3, 2, 2, 1, 1, 0,
9, 8, 7, 5, 3, 2, 2, 1, 1, 0,
Process Exit...
```

tres.

Ordenamiento por Método Sacudida.

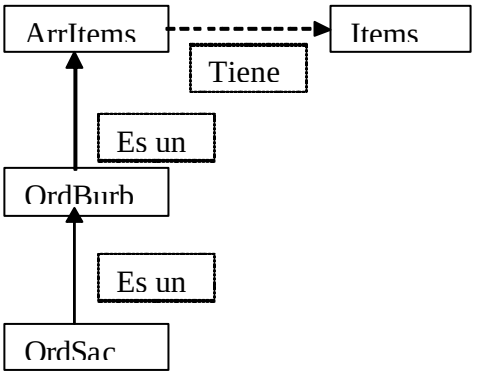
Incorpora un par de ventajas:

- Alterna pasadas de izquierda a derecha y viceversa (menores ➔derecha, mayores ➔ izquierda, igual velocidad)
- "Recuerda" la última inversión realizada en la pasada anterior (no recorrer tramos de la pasada innecesariamente)

```
public class OrdSac extends OrdBurb{        //Ordenamiento p/Sacudida
        public OrdSac(int cant, char tipo){     //      Un constructor
                super(cant,tipo);
        }
        void ordenar(boolean trace) {
                int j,k = talle-1,iz = 1,de = talle-1; Item aux;
                do {                            // Ciclo de control de pasadas
                        for (j = de; j>=iz; j--) // Pasada descendente
                                if (item[j].esMayor(item[j-1])){
                                        item[j].intercambio(item[j-1]);
                                        k = j;      // último intercambio
                                }
                        iz = k+1;
                        if (trace)
                                System.out.println(this);  // el array, como quedó
                        or (j = iz; j<=de; j++)   // Pasada ascendiente
                                if (item[j].esMayor(item[j-1])){
                                        item[j].intercambio(item[j-1]);
                                        k = j;  // último intercambio
                                }
                        de = k-1;
                        if (trace)
                                System.out.println(this);  //   el array, como quedó
                } while (iz<=de);
        }       //      void ordenar
}       //      public class OrdSac
```

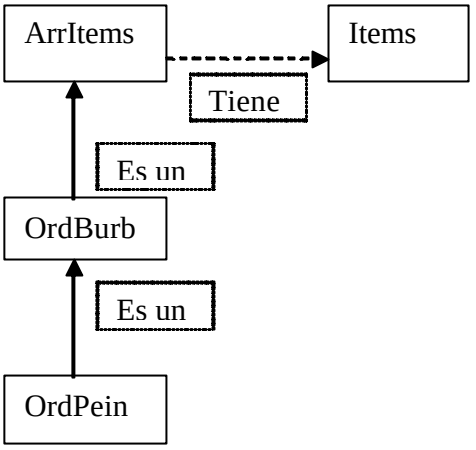
```
import OrdSac;
class PrueSac{
   public static void main(String args[]){
        System.out.println("\nOrdenamiento metodo
Sacudida");
        OrdBurb sac  = new OrdSac(10,'R');
         // Generamos un array de 10 objetos item
random
        sac.ordenar(true);   //   y   lo   ordenamos,
trazando
   }
};
```

```
Ordenamiento metodo Sacudida
5, 6, 9, 9, 3, 0, 2, 8, 1, 1,
Pasadas de ordenamiento
9, 5, 6, 9, 8, 3, 0, 2, 1, 1,
9, 6, 9, 8, 5, 3, 2, 1, 1, 0,
9, 9, 6, 8, 5, 3, 2, 1, 1, 0,
9, 9, 8, 6, 5, 3, 2, 1, 1, 0,
Process Exit...
```



Ordenamiento por Método "Peinado"

Es muy veloz, sin superar al Quick Sort.
Su "paso" es variable, y lo va ajustando
Compara elementos no contiguos.




```
import OrdBurb;
public class OrdPein extends OrdBurb{
        public OrdPein(int cant, char tipo) {
                super(cant,tipo);
        }
        void ordenar(boolean trace) {
          int i,paso = talle;
          boolean cambio;
          do {
                paso = (int)((float) paso/1.3);
                paso = paso>1 ? paso : 1;
                cambio = false;
                for (i = 0; i<talle-paso; i++)
                        if (item[i].esMayor(item[i+paso])){
                                item[i].intercambio(item[i+paso]);
                                cambio = true;
                        }
                if (trace)
                        System.out.println(this);
          } while (cambio || paso>1);
        }       //      void ordenar
}               //      public class OrdPein
```

```
import OrdPein;
class PruePein{
   public static void main(String args[]) {
        System.out.println("\nOrdenamiento          metodo
peinado");
        OrdPein pein  = new OrdPein(10,'R');
        // Generamos un array de 10 objetos item random
        pein.ordenar(true);     // y lo ordenamos, trazando
   }
};
```

```
Ordenamiento metodo Peinado
4, 5, 9, 1, 3, 7, 7, 3, 0, 6,
Pasadas de ordenamiento
3, 0, 6, 1, 3, 7, 7, 4, 5, 9,
3, 0, 4, 1, 3, 7, 7, 6, 5, 9,
1, 0, 4, 3, 3, 5, 7, 6, 7, 9,
1, 0, 3, 3, 4, 5, 7, 6, 7, 9,
0, 1, 3, 3, 4, 5, 6, 7, 7, 9,
0, 1, 3, 3, 4, 5, 6, 7, 7, 9,
Process Exit...
```

```
Ordenamiento metodo Peinado
4, 1, 4, 5, 5, 7, 8, 2, 3, 3,
Pasadas de ordenamiento
2, 1, 3, 5, 5, 7, 8, 4, 3, 4,
2, 1, 3, 3, 4, 7, 8, 4, 5, 5,
2, 1, 3, 3, 4, 5, 5, 4, 7, 8,
2, 1, 3, 3, 4, 4, 5, 5, 7, 8,
1, 2, 3, 3, 4, 4, 5, 5, 7, 8,
1, 2, 3, 3, 4, 4, 5, 5, 7, 8,
Process Exit...
```

ANALISIS DE TIEMPOS

```
import java.util.Calendar;
import java.util.GregorianCalendar;
import OrdBurb;
class PrueGrecal{
        static int minut, segun, milis;
        static long horIni, horFin, tiempo;
        public static void main(String args[]){
                Calendar cal1 = new GregorianCalendar();
                minut = cal1.get(Calendar.MINUTE);
                segun = cal1.get(Calendar.SECOND);
                milis = cal1.get(Calendar.MILLISECOND);
                horIni= minut*60*1000 + segun*1000 + milis;
                System.out.println("\nOrdenamiento metodo Burbuja");
                System.out.println("Inicio "+minut+":"+segun+"."+milis);
                OrdBurb ord = new OrdBurb(1000,'R');
                ord.ordenar(false);          // y lo ordenamos, sin trazado
                System.out.println(ord);
                Calendar cal2 = new GregorianCalendar();
                minut = cal2.get(Calendar.MINUTE);
                segun = cal2.get(Calendar.SECOND);
                milis = cal2.get(Calendar.MILLISECOND);
                horFin= minut*60*1000 + segun*1000 + milis;
                System.out.println("Fin    "+minut+":"+segun+"."+milis);
                tiempo = horFin - horIni;
                System.out.println("Tiempo: " + tiempo + " milisegundos");
        }
};
```

Si lo ejecutamos obtenemos la siguiente salida:

```
Ordenamiento metodo Burbuja
Inicio 49:55.450
999, 698, 846, 885, 335, 474, 838, 738, 244, 565,
Pasadas de ordenamiento
999, 999, 999, 997, 997, 997, 994, 993, 992, 991,
Fin    49:55.560
Tiempo: 110 milisegundos
Process Exit...
```

para ordenar un array de 1000 elementos,
una cpu (Pentium III, 750 Mhs, 64 MB RAM),

Tabla comparativa de tiempos

Array de	1000	5000	10000	20000	50000	100000
Burbuja	110	2190	10490	47570	345970	
Sacudida	110	1930	8840	38830	270010	
Peinado	50	60	110	280	600	1270

Conclusiones

Sacudida no es mucho mejor que Burbuja.
Peinado ordena un array de 50000 elementos **450 veces** más rápido.

ARRAYS MULTIDIMENSIONALES

Tantas sub indicaciones como dimensiones.
Son usuales arrays de dos dimensiones.
Se les llama también tablas o matrices.
No hay límite al número de dimensiones.

<Tipo de dato> <Nombre Array> [<Numero de filas>] [<Numero de columnas>]

Algunos ejemplos de declaración de matrices.

```
char pantalla [25 ] [80];     // Para trabajar una pantalla en modo texto
int matCurs [10 ] [12];       // Cantidad de alumnos por materia/curso
```

Inicialización

Los arrays multidimensionales se pueden inicializar, al igual que los una dimensión, cuando se declaran. Esto lo hacemos asignándole a la matriz una lista de constantes separadas por comas y encerradas entre llaves, ejemplos.

```
public class PruMat01{
        protected int[] [] mat01 = {{51, 52, 53}, {54, 55, 56}};
        // Declaramos e inicializamos el objeto array bidimensional

        public PruMat01(){        // El constructor
                System.out.print(this);
        }
        public String toString(){ // Visualización del objeto
                String aux = "Matriz mat01 \n";
                int f,c;
                for(f=0;f<mat01.length;f++){                    // Recorremos filas
                        for(c=0;c<mat01[0].length;c++)  // Recorremos columnas
                                aux+=mat01[f][c]+", ";
                        aux+="\n";          // A la fila siguiente
                }
                return aux;
        }
        public static void main(String args[]){
                PruMat01 mat = new PruMat01();
        }
}
```

```
Matriz mat01
51, 52, 53,
54, 55, 56,
Process Exit...
```

Acceso

```
public class PruMat02 extends PruMat01{
      public PruMat02(){ // El constructor
       super();
      }

      public void seteos(){ // Modificando valores del objeto
       int f,c,cont=0;
       for(f=0;f<mat01.length;f++){                    // Recorremos filas
              for(c=0;c<mat01[0].length;c++)  // Recorremos columnas
                     mat01[f][c]=++cont;
       }
      }

      public static void main(String args[]){
       PruMat02 mat = new PruMat02();
       mat.seteos();
       System.out.print(mat);
      }
}
```

```
Matriz mat01
51, 52, 53,
54, 55, 56,
Matriz mat01
1, 2, 3,
4, 5, 6,
Process Exit...
```

```
import java.io.IOException;
public class PruMat03 extends PruMat02{
      public PruMat03(){ // El constructor
       super();
      }

      public void setValor(int f, int c, int val){
       mat01[f][c] = val;
      }

      public void lectValor() throws java.io.IOException{
       int f,c,val;
       System.out.print("A que fila pertenece el valor? ");
       f = System.in.read();
       System.out.print("A que col. pertenece el valor? ");
       c = System.in.read();
       System.out.print("Cual es el valor en cuestion ? ");
       mat01[f][c] = System.in.read();
      }

      public int getValor(int f, int c){
       return mat01[f][c];
      }
}
```

```
Matriz mat01
51, 52, 53,
54, 55, 56,
A que fila pertenece el valor? 0
A que col. pertenece el valor? 1
Cual es el valor en cuestion ? 33
aux = mat.getValor(0,1): 33

Matriz mat01
51, 33, 53,
54, 55, 25,
Process Exit...
```

```
import PruMat03;
import java.io.IOException;
public class PruMat04{
      public static void main(String args[]) throws java.io.IOException{
       int aux;
       PruMat03 mat = new PruMat03(); // Instanciamos objeto
       mat.lectValor();               //  Leemos un valor
       mat.setValor(1,2,25);                  // asignamos otro
       aux = mat.getValor(0,1);       //  Obtenemos otro
       System.out.println("\naux = mat.getValor(0,1): "+aux);      //  lo imprimimos
       System.out.println(mat);       //  Imprimimos el objeto
      }
}
```

Una segunda ejecución:

```
Matriz mat01
51, 52, 53,
54, 55, 56,
A que fila pertenece el valor? 2
A que col. pertenece el valor? 2
Cual es el valor en cuestion ?
java.lang.ArrayIndexOutOfBoundsExcepti
on
        at
PruMat03.lectValor(PruMat03.java:20)
        at PruMat04.main(PruMat04.java:8)
Exception in thread "main"
Process Exit...
```

Hemos intentado leer un elemento fuera de la matriz.
No existe fila 2, solo tenemos 0 y 1.
Se ha producido una excepción **ArrayIndexOutOfBoundsException**;

Podriamos capturarla y tratarla?. Lo que pretendemos es que:

- Informemos al operador el problema en cuestion: desborde de indices
- Pueda repetir el procedimiento para la lectura, **public void lectValor()**

```
import java.io.IOException;
import In;
public class PruMat05 extends PruMat03{
   public PruMat05(){      // El constructor
     super();
  }

public void lectValor() throws java.lang.ArrayIndexOutOfBoundsException{
    int f,c,val;
    while (true){
        try{
          System.out.print("\nA que fila pertenece el valor? ");
          f = In.readInt();
          System.out.print("A que col. pertenece el valor? ");
          c = In.readInt();
          System.out.print("Cual es el valor en cuestion ? ");
          mat01[f][c] = In.readInt();
          break;       // Lectura exitosa, salimos del ciclo
         }catch(java.lang.ArrayIndexOutOfBoundsException e){
          System.out.println("\nCuidado, desborde de indice...");
          continue;   // Seguimos intentando la lectura
         }
    } //      while
  }   //      void lectValor()
}     //      class PruMat05
```

```
Matriz mat01
51, 52, 53,
54, 55, 56,

A que fila pertenece el valor? 2
A que col. pertenece el valor? 2
Cual es el valor en cuestion ?
Cuidado, desborde de indice...

A que fila pertenece el valor? 1
A que col. pertenece el valor? 2
Cual es el valor en cuestion ?
25
aux = mat.getValor(0,1): 52
Matriz mat01
51, 52, 25,
54, 55, 25,
Process Exit...
```

```
import PruMat05;
import java.io.IOException;
public class PruMat06{
        public static void main(String args[]) throws java.io.IOException{
                int aux;
                PruMat05 mat = new PruMat05();  // Instanciamos objeto
                mat.lectValor();                //  Leemos un valor
                mat.setValor(0,2,25);           //  asignamos otro
                aux = mat.getValor(0,1);    //  Obtenemos otro
                System.out.println("\naux = mat.getValor(0,1): "+aux);
                System.out.println(mat);    //  Imprimimos el objeto
        }
}
```

Matriz de objetos Item

Aprovechando la clase Item, que ya usamos.

```
import Item;
class MatItems{          //       Matriz de objetos Items
        protected Item[][] item; //
        protected int filas, cols;//  Dimensiones de la matriz
        public MatItems(int fil, int col, char tipo) { // Constructor
                int f,c,auxCod = 0;       // Una variable auxiliar
                filas=fil;cols=col;                     // inicializamos tamaño
                item = new Item[filas][cols]; // Generamos la matriz
                for(f=0;f<item.length;f++)                // Barremos filas
                        for(c=0;c<item[0].length;c++){ // y columnas
                                switch(tipo){   // segun tipo de llenado requerido
                                        case 'A':{ // los haremos en secuencia ascendente
                                                auxCod = c;
                                                break;
                                        }
                                        case 'D':{ // o descendente ...
                                                auxCod = cols - c;
                                                break;
                                        }
                                        case 'R':{ // o bien randomicamentente (Al azar)
                                                auxCod = (int)(cols*Math.random());
                                        }
                                }                                       // switch
                                item[f][c] = new Item();
                                item[f][c].setCodigo(auxCod);
                                item[f][c].setValor((float)(cols*Math.random()));
                        }       // for(c=0
        }       //      public MatItems

        public String toString(){
                int ctos = (cols < 10 ? cols : 10);
                String aux = " Primeros "+ctos+" elementos de la fila 0\n";
                for(int i=0;i<ctos;i++)
                        aux+=item[i][0].toString()+"\n";
                return aux;
        }
        protected int cantCodEnFila(int cod, int fila){
                // Cuantas veces ocurre el cod parámetro en la fila parámetro
                int cuantas = 0,i;
                for(i=0;i<item[fila].length;i++)
                        if(item[fila][i].getCodigo() == cod)cuantas++;
                return cuantas;
        }
        protected double promValCodEnCol(int cod, int col){
                //  retorna el promedio de los valores asociados
                // al código parámetro en la columna parámetro.
                int contCod = 0,i;
                double acuVal = 0.0, prom;
                for(i=0;i<item.length;i++)
                  if (item[i][col].getCodigo() == cod){
                     contCod++; acuVal+=item[i][col].getValor();
                  }
                prom = (double)(acuVal/contCod);
                return prom;
        }
}
```

```
import MatItems;
class  pruMatItems{
       public static void main(String args[]){
              MatItems mtIt = new MatItems(20,20,'R');
              //  Generamos una matriz de 20 x 20 objetos Item,
              // codigos y valores al azar
              System.out.println(mtIt);
              System.out.println("En la fila 10 el codigo 10 ocurre "+
                                  mtIt.cantCodEnFila(10,10)+" veces");
              System.out.println("En la columna 10 el valor promedio\n"+
                                  "de los valores asociados al codigo 10\n"+
                                  "es de "+mtIt.promValCodEnCol(10,10));
       }
};
```

```
Primeros 10 elementos de la fila 0
4 - 2.7692828
17 - 3.8633575
12 - 0.31003436
10 - 19.872202
17 - 8.419615
0 - 8.653201
19 - 19.992702
9 - 0.018264936
15 - 12.856614
17 - 14.902431
En la fila 10 el codigo 10 ocurre 2 veces
En la columna 10 el valor promedio
de los valores asociados al codigo 10 es de 12.118072032928467

Process Exit...
```