Programación orientada a objetos en java:

LAS CLASES

PROGRAMACIÓN O. OBJETOS CON JAVA

JAVA es un lenguaje de programación orientado a objetos. A continuación se profundiza en cada una de los conceptos y características de la programación orientada objetos.

7.1 LAS CLASES

En JAVA todos los programas son clases.

La mayor ventaja de definir y usar clases es la posibilidad de reutilizar código.

La clase es quien define el comportamiento de los objetos. Cuando se trata de crear una clase hay que definir y crear **los métodos** y **datos miembro** que compondrán la clase para que a posteriori podamos utilizarlos en nuestros programas. Los datos miembro deben definirse como **private.**

Una clase contendrá un numero determinado de datos miembro (variables) y mediante los métodos accedemos a los datos miembro para leerlos o para modificarlos. Las clases deben proporcionar métodos que permitan el acceso a los datos miembro de la clase, tanto para asignar como para leer sus valores. También debe poseer métodos que realizan operaciones con esos datos mienbros.

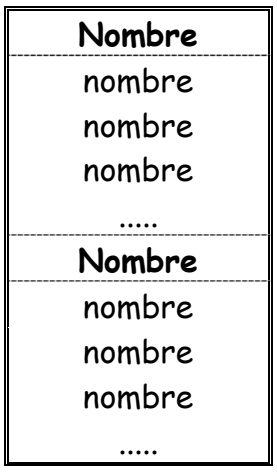
REPRESENTACIÓN GRÁFICA DE CLASES Y OBJETOS

CLASES

Nombre de la clase

Atributos (Datos Miembro)
Características de la clase
lo que se es o cómo está

Métodos
Lo que puede hacerse
con objetos de esa clase:
Posible comportamiento.



OBJETOS

nombre del objeto

valores de los atributos del objeto
memoria o estado
particular de cada objeto

Todos los objetos de una clase
pueden hacer exactamente
lo mismo

Las clases están organizadas en paquetes. Un paquete contiene un conjunto de clases con una utilidad similar. Ahora bien, dentro de un archivo Java puede haber definidos varias clases, pero dentro de un archivo Java **sólo puede haber una clase pública**, y además su nombre tiene que **coincidir con el nombre del archivo Java**.

Ejemplo de creación de una clase:

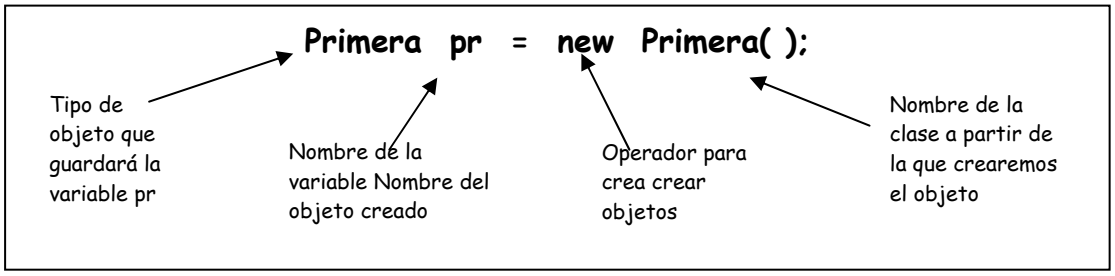
```
[public] class  MiClase   {
            // Datos Miembro de la clase
            private  String   nombre;
            private   int   valor;

            // Métodos de la clase
            public String  getNombre( )   {
                    return nombre;
             }
            public void    setNombre(String  n) {
                    nombre=n;
            }
  }
```

La instrucción **return** permite a la clase devolver un valor.

En Java no hay distinción entre funciones y procedimientos y les llamamos métodos.

Cuando un método devuelve un valor en su definición hay que indicar el tipo del valor que devuelve, si no devuelve nada, se indica con la palabra clave **void**.

Para crear un objeto a partir de una clase, se hace con el operador **new**. Los objetos se almacenan en variables. Los objetos se tratan como si fueran datos normales, es decir, se guardan en variables.



Cuando creamos una clase debemos de abstraernos del uso particular que tendrá cada objeto en los programas. Tan solo hay que pensar en cómo conseguir la funcionalidad que pretendemos dar a la clase con sus datos miembro y métodos, sin pensar nunca en el programa que va a dar uso a esos objetos.

Ejemplo: Crear la clase Triangulo.java:

Datos miembro: base, altura

Métodos: área, perímetro

Una posible solución:

```
class Triangulo    {
        //DATOS MIEMBRO
        private  int   base;
        private  int   altura;
        // METODOS
        public void   inicializa(int a, int h)   {
                base=a;
                altura=h;
        }
        public int   getBase( ) {
                return base;
        }
        public void   setBase(int pbase)  {
                base=pBase;
        }
        public int   getAltura( )   {
                return altura;
        }
        public void   setAltura(int paltura)  {
                altura=paltura;
        }
        public double  area( ) {
                return (base * altura)/2;
        }
        public double   perímetro( ) {        //supongamos que es válido
            int hipot= Math.sqrt(Math.pow(base, 2) + Math.pow(altura, 2));
            return ( base + altura + hipot );
        }
  }
```

Ahora para probar si mi clase funciona y esta correctamente escrita, lo primero que hacemos es compilar. Una vez depurada y sin errores realizamos un pequeño programa de prueba en el método main( ).

```
.....Y en alguna clase.....
public static void main (string[ ] args)
  {
    Triangulo tri= new Triangulo( );   //se crea el ojeto de la clase Triangulo
    tri.setBase(10);
    tri.setAltura(2);
    tri.inicializa( 100,50);     // otra forma de inicializar la base y la altura
    System.out.println("base: " + tri.getBase( ));
    System.out.println("altura: " + tri.getAltura( ));
    System.out.println("area: " + tri.area( ));
    System.out.println("perimetro: " + tri.perimetro( ));
  }
```

PROGRAMA A REALIZAR POR LOS ALUMNOS (ClsCalculadora.java):

Hacer una clase : **ClsCalculadora.java** que será capaz de sumar,..., DOS numeros.

Tendrá cuatro métodos:

```
void   inicializar ( )   //inic. cada operando con un cinco
int    sumar (int,  int)
int    restar(int, int)
int    multiplicar(int, int)
int    dividir(int, int)
```

Hacer una clase **ClsOperacioenes.java** que desde su método MAIN ejecute los métodos de la clase ClsCalculadora.

PROGRAMA A REALIZAR POR LOS ALUMNOS (clase aleatoria):

Realizar un programa que genere números aleatorios dentro de un rango. El usuario nos dará la cantidad de números a generar así como el número más pequeño y el número más grande a generar.

Métodos:

public void generar (int num, int rangoini, int rangofinal)

- num ⇒ Nº de números aleatorios a generar
- rangoini ⇒ Nº desde el que comienza a generar los aleatorios
- rangofinal ⇒ Nº hasta el que genera números aleatorios

public void ordenar (boolean tipo)

- tipo ⇒ indica si se ordena en ascendente(true) o descendentemente(false)

public int total( ) ⇒ indica el nº de números aleatorios generados

public int elemento(int pos) ⇒ Devuelve el elemento generado que ocupa la posición "pos", comprobar antes que dicha posición esta dentro de la tabla.

EN LA SOLUCIÓN PROPUESTA DEBAJO SE HACE TODO EN UNA SOLA CLASE

```
// PROGRAMA QUE CARGA "X" NUMEROS ALEATRORIOS A PARTIR DE UN RANGO INTRODUCIDO POR
//            EL USUARIO.      Copyright (c) 2000 fycsa
   import java.io.*;

public class Ejer040_AleaTodo

   {

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

      {
      BufferedReader bf= new BufferedReader(new InputStreamReader(System.in));
      System.out.print("CUANTOS NUMEROS ALEATORIOS QUIERES GENERAR: ");

      int numeros= new Integer(bf.readLine()).intValue();
                                 // Desglosado quedaria:
                                 //             String  cadena=bf.readline();
                                 //             Integer I=new Integer(cadena);
                                 //             int  numeros= I.intValue();
      System.out.print("\n INTRODUCE EL RANGO MENOR: ");
      int primero= new Integer (bf.readLine()).intValue();
      System.out.print("INTRODUCE EL RANGO MAYOR: ");
      int  ultimo= new Integer (bf.readLine()).intValue();
      int matriz[]=new int[numeros];

      System.out.println("\n LOS NUMEROS CREADOS SON ");
      System.out.println("\n ======================= ");
      for (int n=0; n< numeros;n++)
      {
       matriz[n]=(int)Math.floor(Math.random()*(ultimo-primero)+(primero));
      /*  hemos hecho un casting porque Math devuelve un double, y nuestra
       matriz guarda int. La conversión es explícita*/
       System.out.println("Numero "+(n+1)+":  " + matriz[n] );
      }
      boolean swcambio=true;
      while (swcambio)
      {
       swcambio=false;
       for (int n=0;n<matriz.length-1;n++)
       {
        if (matriz[n]>matriz[n+1])
        {
         swcambio=true;
         int aux=matriz[n];
         matriz[n]=matriz[n+1];
         matriz[n+1]=aux;
        }
       }
      }
     /*     // ORDENACION DEL ARRAY DE UNA FORMA
      //===================================
```

```
    /*      // ORDENACION DEL ARRAY DE UNA FORMA
     //===================================
       for (I1=0; I1<9 ; I1++)          // (I1=0;I1 < ArrayNumerosAleatorios.length-1;I1++)
           for (I2=I1+1; I2<10 ; I2++) // (I2=0;I2 < ArrayNumerosAleatorios.length;I2++)
                   if ( ArrayNumerosAleatorios[I1] > ArrayNumerosAleatorios[I2])
                   {
                         Aux = ArrayNumerosAleatorios[I2] ;
                         ArrayNumerosAleatorios[I2] = ArrayNumerosAleatorios[I1] ;
                         ArrayNumerosAleatorios[I1] =  Aux ;
                   }
    */

    System.out.println("\n LOS NUMEROS ORDENADOS SON ");
    System.out.println("\n ========================= ");
    for (int n=0;n<matriz.length;n++)
        System.out.println("Numero "+(n+1)+"........:  " + matriz[n]);
    System.in.read();    // no necesario con PC-GRASP
  }
}
```

SOLUCION: CREACIÓN DE LA CLASE ALEATORIA.

```
//  CLASE  QUE CARGA "X" NUMEROS ALEATRORIOS A PARTIR DE UN RANGO
//                       INTRODUCIDO POR  EL USUARIO.

class  Ejer042_NumAleatorios
{
 private int Numeros[];

   //   METODO  QUE  CREA EL ARRAY Y LO CARGA CON NUMEROS ALEATORIOS
   //=====================================================================

  public void generar ( int num, int rangoini, int rangofinal)

   {
    Numeros= new int[num];
    int VarTempo;
    int ValorMedio;
    int Intentos=0;
               // int inc=rangofinal-rangoini;
               // for(int n=0; n<num;n++)
               //   numeros[n]=(int)(Math.floor((Math.random()*inc)+rangoini));

    ValorMedio = (rangofinal + rangoini)/2 ;
    for(int n=0; n<num;n++)
    {        //mientras el número esta fuera de rango
     VarTempo=(int)(Math.floor(Math.random()* ValorMedio ));
     Intentos=1 ;
     while ( VarTempo < rangoini  ||  VarTempo > rangofinal )
     {         //mientras el número esta fuera de rango
      VarTempo=(int)(Math.floor(Math.random()* ValorMedio ));
      Intentos ++;
     }
     System.out.println("\n Numero de intentos = " + Intentos);
     Numeros[n]= VarTempo;
     System.out.println("\n El elemento["+n+"] = " + Numeros[n] );
    }
   }
```

```
//   METODO  QUE  ORDENA EL ARRAY EN UN ORDEN
//=======================================================

public void ordenar(boolean tipo)

{
   // Si tipo = true  --> Se quiere ordenar de Menor a Mayor
   // Si tipo = false --> Se quiere ordenar de Mayor a Menor
   int Aux;

   for (int I1=0; I1<Numeros.length-1; I1++)
      for (int I2=I1+1; I2<Numeros.length; I2++)
         if ( Numeros[I1] > Numeros[I2])
         {
            Aux = Numeros[I2] ;
            Numeros[I2] = Numeros[I1] ;
            Numeros[I1] =  Aux ;
         }
   //fin del IF y de los 2 FOR
   System.out.println("\n IMPRESION DE LA LA TABLA EN ASCENDENTE");
   System.out.println("=============================================");
   for (int I=0; I<Numeros.length ; I++)
      System.out.println("\n EL ELEMENTO [ " +I+" ]= " + Numeros[I] );


   if (!tipo)      // (tipo==false)  INVERTIR EL ARRAY
   {               // ================================
      for (int Izq=0,Dcha=Numeros.length-1; Izq<Dcha; Izq++, Dcha--)
      {
         Aux=Numeros[Izq];
         Numeros[Izq]=Numeros[Dcha];
         Numeros[Dcha]=Aux;
      }
      System.out.println("\n IMPRESION DE LA LA TABLA EN DESCENDENTE");
      System.out.println("=============================================");
      for (int I=0; I<Numeros.length ; I++)
         System.out.println("\n EL ELEMENTO [ " +I+" ]= " + Numeros[I] );
   }
}
```

```
    //   METODO  QUE  NOS DICE CUANTOS NUMEROS HAY EN EL ARRAY
    //============================================================
    public int total()
    {
        return Numeros.length;
    }

    //   METODO  QUE  NOS MUESTRA EL VALOR DE UNA POSICIÓN DEL ARRAY
    //================================================================
    public int elemento (int pos)
    {
        return Numeros[pos];
    }
}
```

SOLUCION: EJECUCIÓN DE LA CLASE ALEATORIA

```
//  CLASE  QUE CARGA "X" NUMEROS ALEATORIOS A PARTIR DE UN RANGO
//                       INTRODUCIDO POR  EL USUARIO.
   import java.io.*;

public class  Ejer042_Main1NumAleatorios

   {

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

      {
       int Menor, Mayor, Cantidad;
       BufferedReader bf= new BufferedReader(new InputStreamReader(System.in));
       System.out.print("CUANTOS NUMEROS ALEATORIOS QUIERES GENERAR: ");
       Cantidad = new Integer(bf.readLine()).intValue();  // Desglosado quedaria:
                                                          //  String  cadena=bf.readline();
                                                          //  Integer I=new Integer(cadena);
                                                          //  int  Cantidad= I.intValue();
       System.out.println("INTRODUCE EL RANGO MENOR: ");
       Menor = new Integer (bf.readLine()).intValue();
       System.out.println("INTRODUCE EL RANGO MAYOR: ");
       Mayor = new Integer (bf.readLine()).intValue();

       Ejer042_NumAleatorios   Ejecutar =  new   Ejer042_NumAleatorios( );

       //SE LLAMA AL METODO QUE CARGA EL ARRAY CON NUMEROS ALEATORIOS e imprime desordenados
       //=================================================================================
       Ejecutar.generar(Cantidad, Menor, Mayor);


       //SE LLAMA AL METODO QUE ORDENA EL ARRAY ASCENDENTE o DESCENDENTE y LOS IMPRIME
       //=================================================================================
       boolean orden;
       String CaracterTipoOrden;     // char letra = (char) System.in.read();
       System.out.println(" DIME COMO DESEAS ORDENAR ??? ==> 'A'=Ascendente'D'=Descendente\n");
       CaracterTipoOrden = (bf.readLine());
       if ( CaracterTipoOrden.equals("A") || CaracterTipoOrden.equals("a") )
       {   orden = true;
           Ejecutar.ordenar(orden);
       }
       else if  ( CaracterTipoOrden.equals("D") || CaracterTipoOrden.equals("d") )
       {   orden = false;
           Ejecutar.ordenar(orden);
       }
       else
           System.out.print("\n EL ORDEN SELECCIONADO ES ERRONEO");

       //SE LLAMA AL METODO QUE NOS DICE CUANTOS ELEMENTOS TIENE EL ARRAY
       //=================================================================================
       System.out.print("\n\n EL NUMERO TOTAL DE ELEMENTOS ==> " + Ejecutar.total()+ "\n" );

       //SE LLAMA AL METODO PARA ANALIZAR EL CONTENIDO DE ALGUNOS ELEMENTOS DEL ARRAY
       //=================================================================================
       System.out.println("\n EL CONTENIDO DEL ELEMENTO[3] ==> " + Ejecutar.elemento(3) );
       System.out.println("\n EL CONTENIDO DEL ELEMENTO[6] ==> " + Ejecutar.elemento(6) );
      }
   }
```

PROGRAMA A REALIZAR POR LOS ALUMNOS (Pila de cadenas de caracteres):

Crear una clase llamada "Pila" que contenga **cadenas** de caracteres (String):

Métodos propuestos para la clase PILA DE CADENAS:

Inicializa: sirve para asignarle un tamaño a la pila:
int inicializar(int tamaño);
Poner: sirve para añadir elementos a la pila, si la pila esta llena, no se permitirá añadir más elementos a la pila y devolvera un false:
boolean agregar(String cadenaAañadir);
Sacar: devuelve el último elemento de la pila el borrado (y la pila tendrá un elemento menos): **String sacarCadena( );**
Cuantos quedan: devuelve la cantidada de elementos que quedan en la pila:
int cuantosQuedan( );
Lugar: devuelve la cadena de una posición solicitada de la pila:
String posicionN(int);

Crear Programa de prueba:

Crear un MENU:

- **Si se teclea 1**.- Se creará la pila de cadenas de tamaño N.
- **Si se teclea 2**.- Meter (añadir) una cadena de la pila:
  Se solicitará que se introduzca una cadena.
  - Si me devuelve "true" ==> acción OK
  - Si me devuelve "false" ==> Error no caben más
- **Si se teclea 3**.- SacarCadena (Borrar) una cadena de la pila:
  Se dirá "LA CADENA BORRADA ES =>" cadena
- **Si se teclea 4**.- Consultar una posición de la pila:
  Se dirá "La posición(n)= " cadena
- **Si se teclea 5**.- Consulta cuantos elementos quedan en la pila.
- **Si se teclea 6**.- Salir.

SOLUCION: CREACIÓN DE LA CLASE PILA DE E/S.

```
// CLASE QUE CREA, CARGA, DESCARGA ELEMENTOS DE UNA PILA
//=======================================================
public class Ejer050_ClasePila
{
 private String [] PilaEntradaSalida;
 private int I=0;
 private int TopeElementos=0;

 // INICIALIZA LA PILA ASIGNANDOLE EL TAMAÑO
 public int inicializa(int tamanio)
 {
  TopeElementos=tamanio;
  I=0;
  PilaEntradaSalida = new String [tamanio];
  return  TopeElementos;
 }
 // AÑADE NUEVOS VALORES A LA PILA
 public boolean agregar(String  cadena)
 {
  if (I == TopeElementos)
   return false;
  else
  {
   PilaEntradaSalida[I]=cadena;
   I++;
   return true;
  }
 }
```

```
// BORRA CADENAS DE LA PILA
public String sacar()
{
    String CadenaABorrar;
    if (I > 0)
    {
        I--;
        CadenaABorrar = PilaEntradaSalida[I] ;
        PilaEntradaSalida[I]="E L E M E N T O --- B O R R A D O";
        return CadenaABorrar;
    }
    else
    {
        return "BORRADO => PilaEntradaSalida[0] => YA NO HAY MAS ELEMENTOS";
    }
}
// MUESTRA LA CADENA CONTENIDA EN UNA POSICIÓN X DE LA PILA
public String posicion(int lugar)
{
    if (lugar >=0 && lugar < I )
        return PilaEntradaSalida[lugar];
    else
        return "EL ELEMENTO BUSCADO DE PILA NO EXISTE ===> FUERA DE RANGO";
}

// NOS DICE CUANTAS CADENA SE HAN CARGADO YA EN LA PILA
public int cuantos()
{
    return I;
}

}  //FIN  CLASE
```

SOLUCION: PRUEBA DE LA CLASE PILA DE E/S.

```
/**
 *  CLASE QUE CREA, CARGA, DESCARGA ELEMENTOS DE UNA PILA
 *  ====================================================
 */
  import java.io.*;

public class Ejer050_PruebaPila

{

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

  {
    int OpcionE=6;      //((NO))   String cadena=""; int tamanio=0; boolean Respuesta;
    Ejer050_ClasePila  MiPila = new Ejer050_ClasePila();

    InputStreamReader ip=new InputStreamReader(System.in);
    BufferedReader bf=new BufferedReader(ip);
   // BufferedReader bf=new BufferedReader(new InputStreamReader(System.in)); JUNTO
    do                                                                    //=======
    {
      System.out.println("M E N U   DE   P I L A  ---> Que desea realizar"  );
      System.out.println("================================================");
      System.out.println("Pulse 1 para CREAR LA PILA");
      System.out.println("Pulse 2 para METER (ANIADIR) UNA CADENA");
      System.out.println("Pulse 3 para SACAR (BORRAR) UNA CADENA");
      System.out.println("Pulse 4 para VER LA CADENA DE UNA POSICION");
      System.out.println("Pulse 5 para CONSULTAR CUANTOS ELEMENTOS QUEDAN EN LA PILA");
      System.out.print("Pulse 6 para   S A L I R \n\n =======> TECLEE OPCION:\t");
      OpcionE=Integer.parseInt(bf.readLine());  //<======LEE OPCION
      System.out.println(" ");

      if (OpcionE==1)       //con switch case tb.
      {
        System.out.println(" TAMANIO DE LA PILA A CREAR ????? ==> Teclee un numero");
        int tamanio=Integer.parseInt(bf.readLine());
        int total=MiPila.inicializa(tamanio);
        System.out.println("SE HA CREADO UNA PILA CON " + total + "  ELEMENTOS");
      }
      else if (OpcionE==2)
      {
        System.out.println(" TECLEA UNA CADENA PARA METER EN LA PILA");
        String cadena = bf.readLine();
        boolean Respuesta = MiPila.agregar(cadena);
        if (Respuesta)
        {  System.out.println(" LA CADENA SE HA GUARDADO SATISFACTORIAMENTE EN LA PILA");
           System.out.println(" AHORA HAY UN TOTAL DE = " + MiPila.cuantos()+ " CADENAS EN LA
        }
        else
           System.out.println(" ¡¡ERROR!! PILA LLENA --- NO CABEN MAS ELEMENTOS");
      }
      else if (OpcionE==3)
      {
        String CadenaBorrada = MiPila.sacar();
        System.out.println("SE HA BORRADO LA CADENA SOLICITADA=>  " +  CadenaBorrada );
      }
```

```
        else if (OpcionE==4)
        {
          System.out.println(" TECLEA EL INDICE DE LA CADENA A CONSULTAR e IMPRIMIR");
          int I = Integer.parseInt(bf.readLine());
          System.out.println("SE IMPRIME LA CADENA SOLICITADA=>  " +  MiPila.posicion(I));
        }
        else if (OpcionE==5)
        {
          int NumCadenasQuedanAun = MiPila.cuantos();
          System.out.println(" QUEDAN AUN => " +NumCadenasQuedanAun+ " cadenas en PILA");
        }
        else if (OpcionE != 6)
        {  System.out.println(" OPCION ERRONEA O NO PREVISTA");
        }
      // ============= FIN DEL IF ANIDADO ==========================

        System.out.println("\n <========PRIMER CICLO CONCLUIDO ===========>");
        System.out.println("\n \n   SI DESEA CONTINUAR ??? SELECCIONE OPCION");
      }while(OpcionE != 6) ;

    System.out.println("\n\n FIN DE PROGRAMA SATISFACTORIO");
    }
}
```

Creación de Métodos estáticos (ampliación de lo visto más arriba)

Un método estático es aquel que **no está asociado a un objeto** y **depende directamente de la clase.** Para llamarlos se pone el nombre de la clase, un punto y el nombre del método con sus parámetros.

```
num = Integer.parseInt(string)
num = Math.random( )
```

Los métodos estáticos operan directamente con los valores de sus parámetros, no con los objetos.

Para crear un método estático se hace al definir el método después de la palabra clave **public** con el modificador **static**, antes de indicar el valor que devuelve.

Los métodos estáticos tienen sus limitaciones, por ejemplo, no pueden utilizar los datos miembro de la clase ya que estos son características internas de los objetos de la clase y los métodos estáticos no dependen de ningún objeto, salvo que los datos miembro estén definidos como estáticos.

Ejemplo de Método Estático:

```
class Prueba  {
        public static double cuadrado (int num)  {
                return   num*num;
          }
 }
.............
.............
Prueba.cuadrado(20);
```

Un dato miembro estático no tiene mucho sentido excepto si ese dato miembro queremos que sea una constante. Por Ej. el valor de pi(π). Ese valor lo pueden utilizar cualquier método estático. Es útil si queremos definir una constante que será utilizada por varios métodos estáticos diferentes.

```
class prueba    {
      private static int iva = 16;
      public static float calculo( )
        {  ……  }
  }
```

En el ejemplo de arriba, **por ser static "iva"** todos los objeto que usen esta clase, compartiran dicha variable (es como una constante).
Si no fuese static cada objeto que use esta clase trabajará **con su propia copia** de la variable "iva".

Sí en una clase tiene que llamar a un método de la propia clase es suficiente con poner el nombre del método:

```
class aleatorio {
     public int total( ) {
          ……
    }
     public void generar( ) {                                          ó
          for (int i=0; i <  total( ) ; i++)  ⇔  for (int i=0; i< this.total( ); i++)
          ……
     }
}
```

This ⇒ Es una palabra que representa al propio objeto (a sí mismo).

Un método estático no puede llamar a ningún método de su propia clase a no ser que ese método sea estático, ya que si el método no es estático es porque está asociado al propio objeto.

Desde un método no estático se puede llamar a variables estáticas y a métodos estáticos, pero al contrario no.

DATOS MIEMBRO DE CLASE	MÉTODOS DE CLASE
• Normalmente los atributos declarados existen en cada objeto y no se mezclan. • **Los atributos static**: o sólo existe uno para todos los objetos o todos los objetos lo comparten o si un objeto lo modifica todos lo ven modificado.	• Los métodos de una clase son compartidos por todos los objetos de esa clase. • **Un método static**: o se puede llamar haciendo referencia a la clase o no hace falta crear el objeto para llamarlo o solo pueden trabajar con atributos **static** ya que los atributos normales no estan creados.

Constantes

Una **constante** es una variable que tiene un valor fijo y se crea con la palabra "**final**" delante del tipo de dato. La palabra **final** delante del tipo de dato en la definición de una variable la convierte en una constante por la que a partir de entonces no se le puede cambiar su valor.

```
class prueba
  {
     private final float  iva = 16.0;
     iva =17;    Error⇒ No se puede volver a modificar su valor por ser una constante.
  }
```

Las constantes pueden ser públicas o privadas. Si una constante es pública lo normal es que sea estática ya que no se encuentra asociada a ningún objeto:

```
[public/private] static  final  float  pi=3.141592;
```

No es imprescindible poner las dos con static sería suficiente.

LOS MODIFICADORES DE ACCESO EN JAVA

(ÁMBITO DE ACCESO ó VISIBILIDAD DE LOS OBJETOS)

LAS CLASES:

ESTRUCTURA DE UNA CLASE					
Ámbito Visibilidad	**Modifica-dores**	**objeto CLASE**	**Nombre CLASE**	**Si Hereda de otra Clase**	**Si implementa un Interface**
public **private** ó nada (package)	**abstract** **final** ó nada	**class**	**ClsMiaX**	**extends ClaseX** ó nada	**implements interfacX,...** ó nada

Clases: Ambito, Visibilidad

El control de acceso o el ambito de una clase determina la relaccion que tiene esa clase con otras clases de otros paquetes. El cuerpo de la clase consta en general de una serie de modificadores de acceso(public, protected y private), atributos, mensajes y metodos.

a) public: Tiene un nivel de acceso publico, es decir, se puede hacer instancia desde cualquier otro sitio

b) private: Solo se puede aplicar al fichero en el que está

c) nada(asume package): Una clase con nivel de acceso de paquete puede ser utilizada solo por las clases de su paquete (no estará disponible para otros paquetes, ni siquiera para los subpaquetes de ese paquete.

REFERENCIA " this "

Cada objeto mantiene su propia copia de los atributos, pero no de los métodos de su clase, de los cuales sólo existe una copia para todos los objetos de esa clase. Esto es, cada objeto almacena sus propios datos, pero para acceder y operar con ellos, todos comparten los mismos métodos definidos en su clase.

Por lo tanto, para que un método conozca la identidad del objeto particular para el que ha sido invocado, Java proporciona una referencia denominada **this** , que referencia al propio objeto en el que estamos. **this** hace referencia a la propiedad (dato miembro) o al método de la clase en la que estamos.
super: Hace referencia a la superclase de la que se ha heredado.

Clases: Modificadores

Un modificador es una palabra clave que modifica el nivel de protección, ya sea de una clase, de un método o de un dato miembro.

a) **abstract**: Cuando una clase se diseña para ser generica es casi seguro qe no necesitaremos crear objetos de ella; la razón de su existencia es la de proporcionar los atributos y métodos que serán compartidos por todas sus subclases. Siempre habrá alguna de sus funciones miembro que no tenga código. La subclase perteneciente a esa clase abstracta tendra que implementar el código. ***Una clase será abstract cuando algunos de sus metodos sea abstract. Un metodo abstract lo reconocemos porque va definido por el modificador abstract, no lleva código y termina en ";"*** .

b) **final**: Cuando una clase se declara final estaremos impidiendo que de esa clase se puedan derivar subclases. No se puede heredar de una clase final. Además todos sus métodos se convierten automáticamente en final y no se pueden sobrescribir.

c) **sin modificadores:** Se puede aplicar siempre la herencia y crear subclases.

FUNCIONES O MÉTODOS:

ESTRUCTURA DE FUNCIONES ó MÉTODOS					
Ámbito Visibilidad	**Modifica-dores**	**tipo retorno**	**Nombre MÉTODO**	**(parámetros) que recibe**	**{sentencias}**
public **private** ó nada (package)	**static** **final** **abstract** ó nada	**TipoDatoX** ó void	**metodoX**	**(tipo,tipo,...)** ó **(vacio)**	**{** **sentencias** **}** ó { vacio}

Funciones miembro o métodos: Ambito, Visibilidad.

a) **public**: Desde cualquier ámbito. Ej: el método **Main**

b) **private**: Solo dentro de la clase

c) **ó nada** **(package)**: su ambito es solo dentro del paquete

Funciones miembro o métodos: Modificadores

a) **static**: Accede al metodo sin hacer instancia de la clase

b) **final**: Si es final el método no se puede sobreescribir por la subclase.
Sobreescribir: Es heredar el método de la superclase y modificarlo.

c) **abstract**: Es un método sin codigo. En las subclases estamos obligados a meter el código apropiado para este método.

d) **nada**: Es un método normal. Sin modificadores.

PROPIEDADES O DATOS MIEMBROS:

ESTRUCTURA DE PROPIEDADES o DATOS MIEMBRO					
Ámbito Visibilidad	**Modifica-dores**	**tipo DATO**	**Nombre DATO**	**Valor Inicial del dato**	**Fin de dato**
public **protected** **private** ó nada (package)	**static** **final** ó nada	**tipoX**	**nombreX**	**= valorInicial** ó nada	**;**

Propiedades o datos miembro. Ámbito, Visibilidad.

a) **public**: acceso desde cualquier lugar del proyecto

c) **protected**: Solo pueden acceder la clases que son subclases de ella

d) **private**: Solo dentro de la clase

b) **ó nada (package)**: desde cualquier lugar del paquete.

Propiedades o Datos Miembro y métodos: Modificadores.

a) **static**: Con static hacemos que el último valor se conserve y almacene información común a todos los objetos de esa clase. Lo llamaríamos **atributo de la clase** y podremos conservar el valor sin hacer instancia de la clase. No necesitamos crear un objeto de la clase con el operador **new**, para acceder al dato. Basta con: **NombreClase.metodo();**. Los métodos static solo pueden llamar a métodos de su misma clase que sean también static y solo pueden acceder a datos miembro que también sean estáticos.

b) **final**: declararemos una propiedad final cuando queramos que sea una constante, no se podrá modificar. Esa referencia solo podra utilizarse para referenciar a ese objeto.

AMBITO DE VISIBILIDAD (Resumen).

sin modificadores: Solo es accesible desde dentro de la misma clase o paquete.

public: Para que el objeto sea accesible desde el exterior. LO VE TODO EL MUNDO. (Se puede crear un objeto y luego se puede acceder con "punto" : objeto.xxxx).

```
ClaseObjeto   objMio = new ClaseObjeto( );
ObjMio.metodos( );
```

private: Solo se puede acceder al objeto desde el interior de la clase. Desde el exterior de la clase NO LO VE NADIE.

protected: (protegido). Solo accesible si se hereda. Solo lo ven las clases que lo extienden.

static: Se le puede llamar sin crear un objeto de la clase: "clase.método". Los métodos static solo pueden llamar a métodos de su misma clase que sean también static.

```
NombreClase.Metodo();          Ej: Math.random();
NombreClase.DatoMiembro;       Ej: Math.PI;
```

RESUMEN:

(mayor visibilidad) **PUBLIC >PROTECTED > PRIVATE** (menor visibilidad)

7.2- programación orientada a objetos en java:

CONSTRUCTORES

7.2. CONSTRUCTORES

- Concepto de constructor
- Cómo se crea un constructor
- Constructores con parámetros

Un constructor es un método que hay dentro de una clase pero que tiene una característica especial, que es que **se ejecuta automáticamente cuando se crea un objeto de esa clase**.

La sintaxis de un constructor es:

```
public   nombre_clase ( ) {...}        ó
public   nombre_clase ( parámetros)  {...}
```

Los constructores pueden recibir parámetros pero no pueden devolver nunca nada, por eso no hay que poner ni **void**, ni **int,** ni nada. El nombre del constructor tiene que ser el mismo que el de la clase.

```
class  MiClase
{
        String   nombre;
        public MiClase (String c)
        {
                nombre = c;
        }
}
```

Cuando se crea un objeto con el operador **new** es cuando se ejecuta **el constructor**.

```
MiClase   p  =   new   MiClase (s)
    s es un parámetro que debe ser del tipo del parámetro del constructor
```

Los constructores se utilizan para inicializar los datos miembro de la clase.
Los datos miembro de una clase deben ser privados.

En java es obligatorio que todas las clases tengan un constructor, si creamos una clase y no ponemos ningún constructor, java lo crea automáticamente, ahora bien, lo crea sin parámetros y no hace nada. Esto es lo que se conoce como el constructor por defecto:

Formato de un constructor por defecto de una clase, creado por java:

```
public  MiClase( )
  {
  }
```

En el momento que nosotros creamos un constructor en una clase, java ya no creará el constructor por defecto, y si este constructor tiene algún parámetro, al crear un objeto a partir de esa clase será obligatorio crearlo con los mismos parámetros que los que tengan los constructores definidos en la clase, ya que si no es así, dará error.

Por ello, si creamos objetos a partir de la clase "MiClase" definida más arriba debemos de introducir algún parámetro de tipo String, sino dará error:

MiClase objeto = new MiClase ("Esteban");
MiClase objeto = new MiClase ( ); ⇒ daría error por faltar el parámetro.

EJEMPLO DE USO DE CONSTRUCTORES: CLASE CUENTA BANCARIA

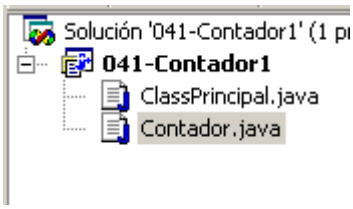
```
//CLASE GESTIÓN CUENTA BANCARIA
//=============================
  class Cuenta
  {  // Datos miembro
     float saldo;

  //CONSTRUCTORES
     public Cuenta  ()        {
        saldo=0; }
     public Cuenta (float n) {
        saldo=n; }
     public Cuenta (int   n) {
        saldo=n; }

     //METODOS DE ACCESO
     public void ingresar    (float cantidad) {
        saldo+=cantidad; }
     public boolean sacar    (float cantidad) {
        if (cantidad <= saldo) {
           saldo-=cantidad;
           return true;
        }
        else
           return false;
     }
     public float saldoActual () {
        return saldo; }
  }  //FIN CLASE
```

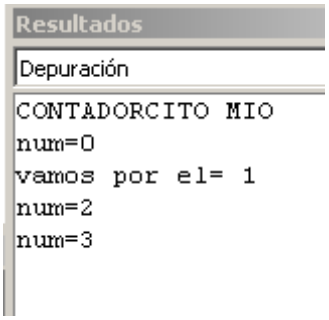
```
//==========PRUEBA DE LA CLASE CUENTA=========================

   public class PruebaCuenta
   {
      public static void main (String [] arg)
      {
         Cuenta C1  = new Cuenta();
         Cuenta C2  = new Cuenta(1500.50f);
         Cuenta C3  = new Cuenta(8234);

         C1.ingresar(300);
         C1.ingresar(500.88f);
         System.out.println("El SALDO ES = " + C1.saldoActual() );      //TB. OK
      // System.out.println("El SALDO ES = " + String.valueOf(C1.saldoActual()));
         if ( C1.sacar(200) )
            System.out.println("El SALDO ES = " + C1.saldoActual() );
         else
            System.out.println("ERROR SALDO MENOR CANTIDAD SOLICITADA");

         if ( C1.sacar(300.38f) )
            System.out.println("El SALDO ES = " + C1.saldoActual() );
         else
            System.out.println("ERROR SALDO MENOR CANTIDAD SOLICITADA");

         C2.ingresar(8499.50f);
         System.out.println("El SALDO ES = " + C2.saldoActual() );

         boolean Respuesta=C3.sacar(13234);
         if ( Respuesta )
            System.out.println("El SALDO ES = " + C3.saldoActual() );
         else
            System.out.println("ERROR SALDO MENOR CANTIDAD SOLICITADA");
      }

   } // FIN CLASE
```

PRUEBA DE CLASE CUENTA BANCARIA INTERACTIVA

```
import java.io.*;

public class PruebaCuentaES {

   public static void main (String [] arg) throws IOException

   {
      Cuenta MiCuenta  = new Cuenta();
      System.out.println("RECIEN CREADA El SALDO ES = " + MiCuenta.saldoActual());
      BufferedReader bf=new BufferedReader(new InputStreamReader(System.in));
      float cantidadES=0;
      int   OpcionE=1;

      while(OpcionE == 1 || OpcionE == 2) {
         System.out.println("=================================================");
         System.out.println("==      M E N U   C U E N T A   CORRIENTE       ==");
         System.out.println("=================================================");
         System.out.println("Pulse 1 para INGRESAR EUROS ");
         System.out.println("Pulse 2 para SACAR EUROS");
         System.out.println("---- CERO PARA SALIR ----");
         System.out.print(" \n\n ========> TECLEE UNA OPCION:\t");

         OpcionE=Integer.parseInt(bf.readLine());  //<======LEE OPCION
         System.out.println("  \n\n\n\n");

         if (OpcionE==1) {
            System.out.println(" ? CUANTO DESEA INGRESAR ==> Teclee cantidad");
            cantidadES=Float.parseFloat(bf.readLine());
            MiCuenta.ingresar(cantidadES);
            System.out.println("SE HA INGRESADO [ "+ cantidadES +" ] EUROS");
            System.out.println("El SALDO ACTUAL ES = " + MiCuenta.saldoActual() );
            System.out.println(" _____________________________________________\n\n");
         }
         else if  (OpcionE==2) {
            System.out.println(" ? CUANTO DESEA  S A C A R ==> Teclee cantidad");
            cantidadES=Float.parseFloat(bf.readLine());

            if ( MiCuenta.sacar(cantidadES) )
               System.out.println("El SALDO ACTUAL ES = " + MiCuenta.saldoActual() );
            else
               System.out.println("ERROR SALDO MENOR CANTIDAD SOLICITADA");
            System.out.println("El SALDO ACTUAL ES = " + MiCuenta.saldoActual() );
            System.out.println(" _____________________________________________\n\n");
         }
         else
            System.out.println(" ADIOS y GRACIAS EXCELENTISIMO CLIENTE");
      }  // FIN while
   }  // FIN main
} // FIN CLASE
```

041-Contador1 (otro ejemplo)



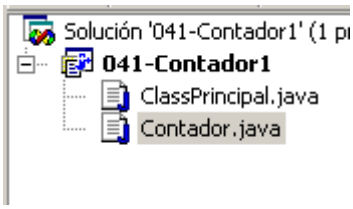


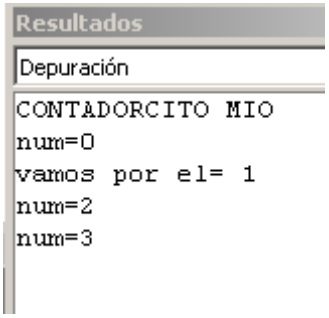

```
//LLAMADA A LOS METODOS DE CONTADOR
import java.io.*;            // Uso la biblioteca de entradas/salidas

public class ClassPrincipal  // IMPORTANTE:
                             // Nombre de la clase igual al nombre del archivo!
{
    // ESTE METODO, MAIN, ES EL QUE HACE QUE ESTO SE COMPORTE COMO APLICACION.
    // Es donde arranca el programa cuando se ejecuta "ClassPrincipal"
    // NOTA: main debe ser public & static.
    static int num;    //entero para asignarle el valor de contador e imprimirlo
    static Contador laCuenta;
    public static void main (String[] args)
    {
        System.out.println ("CONTADORCITO MIO");           // Imprimo el título
        laCuenta = new Contador();                         // Creo una instancia del Contador
        System.out.println ("num="+ laCuenta.getCuenta()); // 0)Imprimo el valor actual (cero!)

        num = laCuenta.incCuenta();                        // 1) Asignación e incremento
        System.out.println ("vamos por el= "+num);         // Ahora imprimo num

        laCuenta.incCuenta();                             // 2)Lo incremento (no uso el valor…
        System.out.println ("num="+laCuenta.getCuenta()); // …de retorno) y lo imprimo
        System.out.println ("num="+laCuenta.incCuenta()); // 3 - Ahora todo en un paso!
    }                       //de esta forma num no sería necesario
}
```

```
/*  Cuando, desde una aplicación u otro objeto, se crea una instancia de la
 clase Contador,  mediante la instrucción:    new Contador()
el compilador busca un método con el mismo nombre de la clase y que se
corresponda con la llamada en cuanto al tipo y número de parámetros.
Dicho método se llama Constructor, y una clase puede tener más de un constructor
(no así un objeto o instancia, ya que una vez que fue creado no puede recrearse
sobre sí mismo).
En tiempo de ejecución, al encontrar dicha instrucción, el intérprete reserva
espacio para el objeto/instancia, crea su estructura y llama al constructor.
O sea que el efecto de new Contador() es, precisamente, reservar espacio para
el contador e inicializarlo en cero.
En cuanto a los otros métodos, se pueden llamar desde otros objetos (lo que
incluye a las aplicaciones) del mismo modo que se llama una función desde C.
*/
```

Ver página siguiente.....

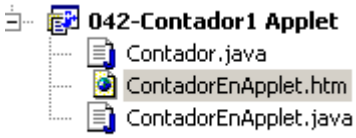
Continuación:

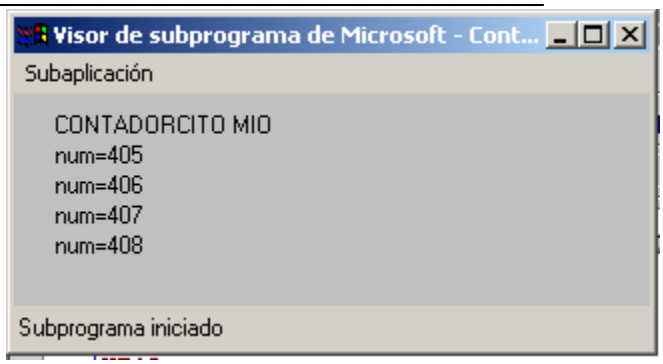





```
public class Contador                //se define la clase contador
{
    //definicion de atributos
    int conta;           //se define un entero para guardar el valor actual

    //constructor
    public Contador()
    {
        conta=0;          //el const. simplemente inicializa el contador
    }


    //METODO-1 ------------------------------------------------------
    public int incCuenta()  // Un método para incrementar el contador
    {
        conta++;              // incrementa cnt
        return conta;         // y de paso devuelve el nuevo valor
    }


    //METODO-2 ------------------------------------------------------------
    public int getCuenta()
    {
        return conta;      // Este sólo devuelve el valor actual del contador
    }

}
```

042-ContadorEnApplet





solo desde Applet

```
// Applet de acción similar a la aplicación 041-Contador1
// GRABAR EN ARCHIVO: "ClassPrincipal.java"
// COMPILAR CON: "javac ClassPrincipal.java"
// PARA EJECUTAR: Crear una página HTML como se indica luego

/*
Para terminar, observemos las diferencias entre la aplicación standalone y el applet:
· La aplicación stanalone usa un método MAIN, desde donde arranca
· El APPLET, en cambio, se arranca desde un constructor (método con el mismo nombre
  que la clase)
Además:
· En la aplicación utilizamos System.out.println para imprimir en la salida estándar
· En el applet necesitamos "dibujar" el texto sobre un fondo gráfico, por lo que usamos
 el método  "g.drawString"  dentro del método "paint" (que es llamado cada vez que es
 necesario redibujar el applet)
        Con poco trabajo se pueden combinar ambos casos en un solo objeto,
        de modo que la misma clase sirva para utilizarla de las dos maneras.
 */

import java.awt.*;
import java.applet.*;

public class ContadorEnApplet  extends Applet
{
    //definicion de atributos
    static int num;    //entero para asignarle el valor de contador e imprimirlo
    static Contador laCuenta;

    //constructor
    public ContadorEnApplet()
    {
        laCuenta = new Contador();  //el constructor simplemente inicializa el contador
    }
```

```
    // El método paint se ejecuta cada vez que hay que redibujar
    // NOTAR EL EFECTO DE ESTO CUANDO SE CAMBIA DE TAMAÑO LA VENTANA DEL NAVEGADOR!

    public void paint (Graphics g)
    {
        g.drawString ("CONTADORCITO MIO", 20, 20);
        g.drawString ("num="+String.valueOf(laCuenta.getCuenta()), 20, 35 );

        num = laCuenta.incCuenta();               // 1) Asignación e incremento
        g.drawString ("num="+ String.valueOf(num), 20, 50 );
        laCuenta.incCuenta();
        g.drawString ("num="+ String.valueOf(laCuenta.getCuenta()), 20, 65 );
        g.drawString ("num="+ String.valueOf(laCuenta.incCuenta()), 20, 80 );
    }
}
```

```
public class Contador                //se define la clase contador
{
    //definicion de atributos
    int conta;           //se define un entero para guardar el valor actual

    //constructor
    public Contador()
    {
        conta=0;          //el const. simplemente inicializa el contador
    }


    //METODO-1 ------------------------------------------------------
    public int incCuenta()  // Un método para incrementar el contador
    {
        conta++;              // incrementa cnt
        return conta;         // y de paso devuelve el nuevo valor
    }


    //METODO-2 ------------------------------------------------------------
    public int getCuenta()
    {
        return conta;     // Este sólo devuelve el valor actual del contador
    }
}
```

```
/* ============================================================================
 Cuando, desde una aplicación u otro objeto, se crea una instancia de la
 clase Contador,  mediante la instrucción:    " laCuenta = new Contador(); "
el compilador busca un método con el mismo nombre de la clase y que se
corresponda con la llamada en cuanto al tipo y número de parámetros.
Dicho método se llama Constructor, y una clase puede tener más de un constructor
(no así un objeto o instancia, ya que una vez que fue creado no puede recrearse
sobre sí mismo).
En tiempo de ejecución, al encontrar dicha instrucción, el intérprete reserva
espacio para el objeto/instancia, crea su estructura y llama al constructor.
O sea que el efecto de new Contador() es, precisamente, reservar espacio para
el contador e inicializarlo en cero.
En cuanto a los otros métodos, se pueden llamar desde otros objetos (lo que
incluye a las aplicaciones) del mismo modo que se llama una función desde C.
*/
```

Contenido del Fichero ContadorEnApplet.htm

```
<HTML>
<HEAD>
<TITLE> CONTADOR EN UN APPLET</TITLE>
</HEAD>
<BODY>
<applet code="ContadorEnApplet.class" width=170 heigth=150>
</applet>
</BODY>
</HTML>
```

OTRO EJEMPLO DE CONSTRUCTORES:

```
 /*****************************************************************************\
 ****  Aqui, Caja define tres constructores para inicializar de varias formas      *****
 ****  las dimensiones de una caja                                                  *****
 \*****************************************************************************/

class Caja {
  double Anchura;
  double Altura;
  double Profundidad;
  // Constructor utilizado cuando se especifican todas las dimensiones
  Caja(double Anc, double Alt, double Pro) {
      Anchura = Anc;
      Altura = Alt;
      Profundidad = Pro;
  }
  // Este constructor se usa cuando no se especifica ninguna dimension
  Caja() {
      Anchura = -1;       // Usa -1 para inicializar
      Altura = -1;        // una caja no inicializada
      Profundidad = -1;
  }
  // Este constructor se usa cuando se crea un cubo
  Caja(double Lado) {
      Anchura = Altura = Profundidad = Lado;
  }
   // METODO que Calcula y devuelve el volumen
  double Volumen() {
      return Anchura * Altura * Profundidad;
  }
} //-------------------------------------------------------------------

public class CajaConstructoresSobrecargados {
    public static void main(String args[]) {
    // Crea cajas utilizando varios constructores
    // SE DECLARA, SE RESERVA ESPACIO E SE INICIALIN LOS OBJETOS
      Caja MiCaja1= new Caja();
      Caja MiCaja2= new Caja(10); //para crear hexaedro o cubo
      Caja MiCaja3= new Caja(40, 20, 50);
      double Volumen;
    // Calcula el volumen de la primera caja
      Volumen = MiCaja1.Volumen();
      System.out.println("El volumen de la primera caja es " + Volumen);
    // Calcula el volumen de la segunda caja
      Volumen = MiCaja2.Volumen();
      System.out.println("El volumen de la segunda caja es " + Volumen);
    // Obtiene el volumen del cubo
      Volumen = MiCaja3.Volumen();
      System.out.println("El volumen de la tercera caja es " + Volumen);
  }
}   // fin clase
```

PROGRAMA A REALIZAR POR LOS ALUMNOS:

Hacer una copia de la clase PILA DE CADENAS y modificarlo para que utilize CONSTRUCTORES en lugar del Metodo "Inicializa". Lo demás se puede respetar:

- Se eliminará el MÉTODO INICIALIZA
- Se le pide al usuario que sea él el que decida si nos introduce el numero de elementos que se van a guardar en la PILA o bién, en su defecto el programa creará una pila de 10 elementos.

SOLUCION:

```
// CLASE QUE CREA, CARGA, DESCARGA ELEMENTOS DE UNA PILA
//====== C O N  C O N S T R U C T O R E S ==============
   public class  Pila
   {
   // D A T O S    M I E M B R O
   //============================
      private String [] PilaEntradaSalida;
      private int I;
      private int TopeElementos;

   // C O N S T R U C T O R E S
   //============================
      public   Pila()
      {
         I=0;
         TopeElementos=0;
         PilaEntradaSalida = new String[10];
      }

      public   Pila(int TotalElementos)
      {
         I=0;
         TopeElementos=TotalElementos;
         PilaEntradaSalida = new String[TotalElementos];
      }
```

RESTO DE MÉTODOS:

```
// AÑADE NUEVOS VALORES A LA PILA
   public boolean agregar(String  cadena)
   {
      if (I == TopeElementos)
         return false;
      else
      {
         PilaEntradaSalida[I]=cadena;
         I++;
         return true;
      }
   }

// BORRA CADENAS DE LA PILA
   public String sacar()
   {
      String CadenaABorrar;
      if (I > 0)
      {
         I--;
         CadenaABorrar = PilaEntradaSalida[I] ;
         PilaEntradaSalida[I]="E L E M E N T O --- B O R R A D O";
         return CadenaABorrar;
      }
      else
      {
          return "BORRADO => PilaEntradaSalida[0] => YA NO HAY MAS ELEMENTOS";
      }
   }


// MUESTRA LA CADENA CONTENIDA EN UNA POSICIÓN X DE LA PILA
   public String posicion(int lugar)
   {
      if (lugar >=0 && lugar < I )
         return PilaEntradaSalida[lugar];
      else
         return "EL ELEMENTO BUSCADO DE PILA NO EXISTE ===> FUERA DE RANGO";
   }

// NOS DICE CUANTAS CADENA SE HAN CARGADO YA EN LA PILA
   public int cuantos()
   {
      return I;
   }
} //FIN  CLASE
```

```
/**
 *  CLASE QUE CREA, CARGA, DESCARGA ELEMENTOS DE UNA PILA
 *  ====================================================
 */
  //import MisClases.Pila;   //donde busca la clase: Pila
  import java.io.*;
  public class PruebaPila
  {
     public static void main (String[] args)  throws IOException
     {
        BufferedReader bf=new BufferedReader(new InputStreamReader(System.in));
        int OpcionE=0;
        Pila MiPila=null;    //DECLARACION DE LA PILA iniciada a null por que es un obj.
        do{
           System.out.println("=================================================");
           System.out.println("==     MENU  INICIAL  DE  CREACION  DE  PILA    ==");
           System.out.println("=================================================");
           System.out.println("Pulse 1 para CREAR LA PILA USTED MISMO de 'N' elementos");
           System.out.println("Pulse 2 para que la cree JAVA con '10' elementos");
           System.out.print(" \n\n =====> TECLEE UNA OPCION:\t");

           OpcionE=Integer.parseInt(bf.readLine());  //<======LEE OPCION
           System.out.println("  \n\n\n\n");

           if (OpcionE==1)
           {
              System.out.println(" TAMANIO DE LA PILA A CREAR ????? ==> Teclee un numero");
              int tamanio=Integer.parseInt(bf.readLine());
              MiPila = new Pila(tamanio);    // <==========> SE LLAMA AL CONSTRUCTOR
              System.out.println("SE HA CREADO UNA PILA CON [" + tamanio + "] ELEMENTOS");
           }
           else if  (OpcionE==2)
           {
              MiPila = new Pila();         // <==================> SE LLAMA AL CONSTRUCTOR
              System.out.println("SE HA CREADO UNA PILA CON [10]  ELEMENTOS");
           }
           else   System.out.println(" OPCION INCORRECTA: INTENTALO OTRA VEZ");
        }while(OpcionE != 1 && OpcionE != 2) ;

       do
       {
          System.out.println("=============================================================");
          System.out.println("===   MENU  UTILIZACION DE PILA ---> Que desea realizar  ===");
          System.out.println("=============================================================");

          System.out.println("Pulse 1 para METER (ANIADIR) UNA CADENA");
          System.out.println("Pulse 2 para SACAR (BORRAR) UNA CADENA");
          System.out.println("Pulse 3 para VER LA CADENA DE UNA POSICION");
          System.out.println("Pulse 4 para CONSULTAR CUANTOS ELEMENTOS QUEDAN EN LA PILA");
          System.out.print("Pulse 5 para   S A L I R \n\n =======> TECLEE OPCION:\t");
          OpcionE=Integer.parseInt(bf.readLine());  //<======LEE OPCION
          System.out.println(" ");

         if (OpcionE==1)
         {
            System.out.println(" TECLEA UNA CADENA PARA METER EN LA PILA");
            String cadena = bf.readLine();
            boolean Respuesta = MiPila.agregar(cadena);
            if (Respuesta)
            {  System.out.println(" LA CADENA SE HA GUARDADO SATISFACTORIAMENTE EN LA PILA");
               System.out.println(" AHORA HAY UN TOTAL DE = " + MiPila.cuantos()+ " CADENAS EN LA PILA");
            }
            else
               System.out.println(" ¡¡ERROR!! PILA LLENA --- NO CABEN MAS ELEMENTOS");
```

```
        }
        else if (OpcionE==2)
        {
           String CadenaBorrada = MiPila.sacar();
           System.out.println("SE HA BORRADO LA CADENA SOLICITADA=>  " +  CadenaBorrada );
        }
        else if (OpcionE==3)
        {
           System.out.println(" TECLEA EL INDICE DE LA CADENA A CONSULTAR e IMPRIMIR");
           int I = Integer.parseInt(bf.readLine());
           System.out.println("SE IMPRIME LA CADENA SOLICITADA=>  " +  MiPila.posicion(I));
        }
        else if (OpcionE==4)
        {
           int NumCadenasQuedanAun = MiPila.cuantos();
           System.out.println(" QUEDAN AUN => " +NumCadenasQuedanAun+ " cadenas en PILA");
        }
        else if (OpcionE != 5)
        {  System.out.println(" OPCION ERRONEA O NO PREVISTA");
        }
      // ============= FIN DEL IF ANIDADO ==========================

        System.out.println
              ("\n <############################## CICLO CONCLUIDO ######################>");
        System.out.println("\n \n   SI DESEA CONTINUAR ??? SELECCIONE OPCION");
      }while(OpcionE != 5) ;
       System.out.println("\n\n FIN DE PROGRAMA SATISFACTORIO");
       System.in.read();    //para que pare hasta pulsar una tecla
   }
}     // FIN PROGRAMA
```

this es una referencia especial predefinida para cada objeto:

- Dentro de un objeto existe siempre
- Referencia al propio objeto
- Es util, cuando hay variables con el mismo nombre.

```
// Utilizacion redundante de this.
    Caja(double Anc, double Alt, double Pro) {
        this.Anchura = Anc;
        this.Altura = Alt;
        this.Profundidad = Pro;
    }

// This resuelve colisiones del espacio de nombres.
   Caja(double Anchura, double Altura, double Profundidad) {
        this.Anchura = Anchura;
        this.Altura = Altura;
        this.Profundidad = Profundidad;
    }
```

- **SOBRECARGA DE MÉTODOS**

Una clase puede tener varios **métodos con el mismo nombre**.
Ya hemos visto como se pueden crear varios constructores con el mismo nombre (un constructor es un método que se llama igual que la clase).

De la misma forma podemos tener varios METODOS diferentes con **el mismo nombre**. Pero tienen que tener **diferente número de parámetros**, o si tienen el mismo número de parámetros, éstos serán de **distinto tipo**. Los tipos de los datos que se retornan no se tienen en cuenta.

Las conversiones automáticas de tipos se mantienen en la sobrecarga de métodos.

EJEMPLOS DE SOBRECARGA DE MÉTODOS:

```
// EJEMPLO DE SOBRECARGA DE MÉTODOS.
//==================================
class Sobrecarga {
 void pueba() {
   System.out.println("Sin parametros");
 }
 // Sobrecarga del metodo pueba con un parametro entero.
 void pueba(int a) {
   System.out.println("a: " + a);
 }
 // Sobrecarga del metodo pueba con dos parametros enteros
 void pueba(int a, int b) {
   System.out.println("a y b: " + a + " " + b);
 }
 // Sobrecarga del metodo pueba con un parametro double
 double pueba(double a) {
   System.out.println("a: " + a);
   return a*a;
 }
} // FIN CLASE
//============================================================
public class DemostracionDeSobrecarga1 {
 public static void main(String args[]) {
   Sobrecarga Objeto = new Sobrecarga();
   double Resultado;

   // Llamamos a todas las versiones de pueba()
   Objeto.pueba();
   Objeto.pueba(10);
   Objeto.pueba(10, 20);
   Resultado = Objeto.pueba(123.2);
   System.out.println("Resultado de Objeto.pueba(123.2): " + Resultado);
 }
} // FIN CLASE
```

CONVERSIONES IMPLICITAS DE TIPOS:

Debemos definir los tipos de los datos con mucho cuidado para evitar errores o resultados imprevisibles.

```
// Las conversiones automaticas de tipos se aplican en la sobrecarga.

class Sobrecarga {
 void prueba() {
   System.out.println("Sin parametros");
 }

 // Sobrecarga el metodo prueba con dos parametros enteros
     void prueba(int a, int b) {
        System.out.println("a y b: " + a + " " + b);
 }

 // Sobrecarga el metodo prueba con un parametro double
     void prueba(double a) {
        System.out.println("Dentro de prueba(double) a: " + a);
 }
}
//=============================================================
class DemostracionDeSobrecarga2 {
  public static void main(String args[]) {
   Sobrecarga Objeto = new Sobrecarga();
   int entero = 88;

   Objeto.prueba();
   Objeto.prueba(10, 20);

   Objeto.prueba(entero); // Esto llama a prueba(double)
   Objeto.prueba(123.2);  // Esto llama a prueba(double)
  }
}
```

PROGRAMA A REALIZAR POR LOS ALUMNOS (CLASE TRIANGULOS):

Crear una clase que sirva para hallar ÁREA DE TRIÁNGULOS.

DATOS MIEMBRO:

La clase contendrá 2 datos miembro: **base** y **altura** de tipo entero.
y además para el perímetro: **lado1, lado2 y lado3**.

CONSTRUCTORES:

1º: Un método inicializa cada dato miembro con un valor por defecto.
Triangulo( );

2º: Otro método inicializa la base y la altura con el mismo valor.
Triangulo(int);

3º: Otro método inicializa la base y la altura con su valor.
Triangulo(int, int);

4º: Otro método inicializa los 3 lados con su valor.
Triangulo(int, int, int);

MÉTODOS:

1º: Cada dato miembro tiene un método GET y un método SET.
Para modificar o para ver los datos miembro.

2º: Habrá los siguientes métodos **sobrecargados** de área y de perímetro:

```
int     area( );
int     perimetro( );
int     area( int, int);        no usa los datos miembro, usa los datos que recibe
int     perimetro(int, int, int);   no usa los datos miembro
```

```
static  float   area(float, float)   SERÁ ESTÁTICO
static  float   perimetro(float, float, float)   SERÁ ESTÁTICO
```

SOLUCIÓN AL EJERCICIO PROPUESTO (CLASE TRIANGULOS):

```
/*****************************************************\
****      C L A S E    T R I A N G U L O S         *****
****  CONSTRUCTORES Y METODOS SOBRECARGADOS         *****
\*****************************************************/

public class Triangulos
{
// DATOS MIEMBRO  base=lado1
 private int base;
 private int altura;
 private int lado1, lado2, lado3;

// CONSTRUCTORES
 public Triangulos()
 {
  base=0;       // base=lado1
  altura=0;
  lado1=base;
  lado2=0;
  lado3=0;
 }
 public Triangulos(int pvalor)
 {
  base=lado1=pvalor;
  altura=pvalor;

 }
 public Triangulos(int pbase,int paltura)
 {
  base=pbase;
  altura=paltura;
  lado1=pbase;
 }

 public Triangulos(int plado1, int plado2, int plado3)
 {
  lado1=base=plado1;
  lado2=plado2;
  lado3=plado3;
 }
```

```
//METODOS

public void setBase(int pbase)
{
    base=lado1=pbase;
}

public int getBase()
{
    return base;
}

public void setAltura(int paltura)
{
    altura=paltura;
}

public int getAltura()
{
    return altura;
}

public void setLado1(int plado1)
{
    lado1=base=plado1;
}

public int getLado1()
{
    return lado1;
}

public void setLado2(int plado2)
{
    lado2=plado2;
}
```

```
public int getLado2()
{
    return lado2;
}

public void setLado3(int plado3)
{
    lado3=plado3;
}

public int getLado3()
{
    return lado3;
}

// ============> M E T O D O S   S O B R E C A R G A D O S <===============
/* ========================================================================
*  Método Area NO ESTATICO, es decir que depende de un objeto, así para   *
* llamarlo sería:          double ResulArea;                               *
*                          Triangulos tri= new Triangulos();               *
*                          ResulArea=tri.area();                           *
========================================================================*/

public int area()
{
    return (base*altura)/2; // return(getBase()*getAltura());
                               // return(this.getBase()*this.getAltura());
}

public int area(int _base, int _altura)
{
    return (_base * _altura)/2;
}

public int perimetro()  //USANDO LOS LADOS ALMACENADOS
{
    return (lado1 + lado2 + lado3);    //SUMAS DE LADOS
    // Math.sqrt(Math.pow(base,2)+ Math.pow(caltura,2));//HIPOTENUSA
}
```

```
    public int  perimetro(int L1, int L2, int L3)
    {
        return (L1 + L2 + L3);    //NOS PASAN LOS LADOS
        // Math.sqrt(Math.pow(base,2)+ Math.pow(caltura,2));//HIPOTENUSA
    }

    /* ================================================================
     *  Métodos ESTÁTICOS, así NO depende de ningún objeto, sino      *
     * directamente de la CLASE TRIANGULO llamarlo sería:             *
     *      Así para llamarlo seria:                                  *
     *                ResulArea = Triangulo.area(int pBase,int pAltura);  *
     ================================================================*/
    public static double area(double _base, double _altura)
    {
        return ( _base  * _altura ) / 2;
    }

    public static float perimetro(float L1, float L2, float L3)
    {
        return (L1 + L2 + L3);    //NOS PASAN LOS LADOS
    }
} // FIN CLASE TRIANGULOS
```

```
/*****************************************************\
****      C L A S E  PRUEBA  T R I A N G U L O S  *****
****  CONSTRUCTORES Y METODOS SOBRECARGADOS        *****
\*****************************************************/
public class PruebaTriangulos  {
  public static void main (String [] argumentos)  {
     Triangulos Tri1 = new Triangulos(100,200,300);    // los 3 lados
     System.out.println("\n\nDATOS RELACIONADOS CON TRIANGULO PRIMERO");
     System.out.println("=======================================");
     System.out.println("BASE= "+ Tri1.getBase()+" = LADO1= "+ Tri1.getLado1());
     Tri1.setAltura(400);
     System.out.println("ALTURA DEBE SER 400 y SACAMOS = "+ Tri1.getAltura());
     System.out.println("EL PERIMETRO DEBE SER 600 y SACAMOS="+ Tri1.perimetro());
     System.out.println("EL AREA DEBE SER 20000 y SACAMOS = "+ Tri1.area());

     System.out.println("EL AREA DEBE SER 20 y SACAMOS = "+ Tri1.area(5,8) );
     System.out.println("EL AREA DEBE SER 23.11 y SALE="+Tri1.area(5.55f,8.33f));
     System.out.println("EL PERIMETRO SERA 55 y SALE="+ Tri1.perimetro(8,32,15));

     //UTILIZANDO LOS METODOS ESTATICOS
     float PeriEstatico = Triangulos.perimetro(10.11f, 20.55f, 30.33f);
     System.out.println("PERI estatico SERA 60.98 y SACAMOS = "+ PeriEstatico );
     double AreaEstatica = Triangulos.area(5.55d, 3.33d);  //DOUBLE
     System.out.println("AREA estatica SERA 9.24 (double) = "+ AreaEstatica );
     AreaEstatica = Triangulos.area(5.55f, 3.33f);  //FLOAT
     System.out.println("AREA estatica SERA 9.24  (float) = "+ AreaEstatica );

     Triangulos Tri2 = new Triangulos();
     System.out.println("\n\nDATOS RELACIONADOS CON TRIANGULO SEGUNDO");
     System.out.println("=======================================");
     Tri2.setBase(100);
     Tri2.setAltura(400);
     Tri2.setLado2(200);
     Tri2.setLado3(300);
     System.out.println("BASE ="+ Tri2.getBase()+" ALTURA= "+ Tri2.getAltura());
     System.out.println("LADO2="+ Tri2.getLado2()+" ALTURA= "+ Tri2.getLado3());
     System.out.println("EL PERIMETRO DEBE SER 600 y SACAMOS= "+Tri2.perimetro());
     System.out.println("EL AREA DEBE SER 20000 y SACAMOS   = "+ Tri2.area());
  } // FIN MAIN
} // FIN CLASE PRUEBA TRIANGULOS
```

RESULTADOS EN PANTALLA

```
DATOS RELACIONADOS CON TRIANGULO PRIMERO
=======================================
BASE= 100 = LADO1= 100
ALTURA DEBE SER 400 y SACAMOS = 400
EL PERIMETRO DEBE SER 600 y SACAMOS = 600
EL AREA DEBE SER 20000 y SACAMOS = 20000
EL AREA DEBE SER 20 y SACAMOS = 20
EL AREA DEBE SER 23.11 y SACAMOS= 23.115750582695
EL PERIMETRO SERA 55 y SACAMOS = 55
PERI estatico SERA 60.98 y SACAMOS = 60.989998
AREA estatica SERA 9.24 (double) = 9.24075
AREA estatica SERA 9.24  (float) = 9.2407501058578


DATOS RELACIONADOS CON TRIANGULO SEGUNDO
=======================================
BASE = 100 ALTURA= 400
LADO2= 200 ALTURA= 300
EL PERIMETRO DEBE SER 600 y SACAMOS = 600
EL AREA DEBE SER 20000 y SACAMOS    = 20000
```