SOBRECARGA DE MÉTODOS

A menudo interesa que existan diferentes versiones del mismo método para aceptar distintos tipos de conjuntos de parámetros. Sobrecargar un método es volverlo a escribir con parámetros diferentes.

Ejemplo:

```
class SobreCarga{
   void miMetodo(){
      System.out.println("no hay parámetros");
   }
   void miMetodo(int a){
      System.out.println("un parámetro: " + a);
   }
   void miMetodo(int a,int b){
      System.out.println("dos parámetros: " + a +" y " + b);
   }
   void miMetodo(double a,int b){
      System.out.println("dos parámetros: " + a +" y " + b);
   }
}
class Unidad2{

   public static void main(String[] args) {
      SobreCarga ob = new SobreCarga();
      ob.miMetodo();
      ob.miMetodo(5);
      ob.miMetodo(5, 10);
      ob.miMetodo(6.7, 16);
   }
}
```

En terminología Java una ***firma***, es el nombre de un método más su lista de parámetros. Para realizar una sobrecarga correcta todas las versiones del método tienen que tener firmas diferentes como así ocurre en el caso anterior:

```
miMetodo()
miMetodo(int a)
miMetodo(int a,int b)
miMetodo(double a,int b)
```

Ten en cuenta:

- la última versión, `miMetodo(double a,int b)`, es legal, ya que aunque ya hay una firma con dos parámetros, sus tipos son al menos diferentes en uno.
- El compilador decide qué versión del método debe ejecutar examinando los argumentos, número y tipo, con los que invocamos al método. Recuerda que "argumento" es el nombre técnico de los valores que se envían a los parámetros de un método al invocar a dicho método.

Ejemplo: Añade el siguiente código y observa el error señalado.

```
void miMetodo(int c){
      System.out.println("otro método con un parámetro: " + c);
}
```

Sobrecarga de métodos y tipo de retorno.
El compilador **no** considera el tipo devuelto por un método como parte de la firma y por tanto no se puede utilizar el tipo devuelto para conseguir una firma distinta:

Ejemplo: Reformamos el método anterior cambiando el tipo devuelto y seguimos observando error.

```
int miMetodo(int c){
      System.out.println("otro método con un parámetro: " + c);
        return c*2;
 }
```

El tipo de retorno no se puede tener en cuenta para conseguir una firma distinta
Este comportamiento, no incluir el tipo de retorno en la firma es totalmente lógico, por ejemplo imagina que el compilador nos permitiera definir estos dos métodos en la misma clase

```
int miMetodo(int c){
      System.out.println("otro método con un parámetro: " + c);
        return c*2;
 }

void miMetodo(int c){
      System.out.println("otro método con un parámetro: " + c);
}
```

Ahora sitúate en el main(), puedo escribir una instrucción tal que así:
miMetodo(4);
El compilador sería incapaz de distinguir que versión del método quiero ejecutar.
Fíjate que al invocar un método que devuelve un valor no estoy obligado a utilizarlo
int x=miMetodo(4); //utilizo el valor devuelto por el método
ó
miMetodo(4); //no lo utilizo
en este último caso aprecio que el compilador no sabría qué versión usar

Lo que sí hay que tener claro es que cada versión puede devolver un tipo diferente. Insistiendo en que esto no influye a la hora de obtener firmas diferentes.
Ejemplo: Prueba el ejercicio inicial modificado de la siguiente forma:

```
class SobreCarga{
   void miMetodo(){
      System.out.println("no hay parámetros");
   }
   int  miMetodo(int a){
      System.out.println("un parámetro: " + a);
      return a*2;
   }
   char   miMetodo(int a,int b){
      System.out.println("dos parámetros: " + a +" y " + b);
      return (char) (a+b);
   }
   double  miMetodo(double a,int b){
      System.out.println("dos parámetros: " + a +" y " + b);
       return a*b;
   }
}
class Unidad2{
   public static void main(String[] args) {
```

```
        int i;
        char c;
        double d;

        SobreCarga ob = new SobreCarga();
        ob.miMetodo();
        i=ob.miMetodo(5);
        System.out.println("Variable i: " +i);
        c=ob.miMetodo(92, 10);
        System.out.println("Variable c: " +c);
        d=ob.miMetodo(6.7, 16);
         System.out.println("Variable d: " +d);
    }
}
```

println() es un método sobrecargado

consulta en el API JAVA que la clase PrintStream tiene un método que se llama ¡println()! y que está sobrecargado

Sobrecarga de constructores

Idem métodos ya que un constructor no es más que un método con algunas características especiales.

Ejemplo:

```
class MiClase{
  int x;

  MiClase(int i){
    System.out.println("Dentro de MiClase(int i)");
    x=i;
  }
  MiClase(double d){
    System.out.println("Dentro de MiClase(double d)");
    x= (int) d;
  }
  MiClase(int i, int j){
    System.out.println("Dentro de MiClase(int i, int j)");
    x= i*j;
  }
}
class Unidad2{
    public static void main(String[] args) {
      // MiClase ob1= new MiClase();
       MiClase ob2= new MiClase(88);
       MiClase ob3= new MiClase(17.23);
       MiClase ob4= new MiClase(2,4);

       //System.out.println("ob1.x: "+ ob1.x);
       System.out.println("ob2.x: "+ ob2.x);
       System.out.println("ob3.x: "+ ob3.x);
       System.out.println("ob4.x: "+ ob4.x);
    }

}
```

Ejemplo: con el profesor, consulta en el API JAVA que la clase String tiene un constructor sobrecargado.

sobrecarga y constructor por defecto

Observa que la instrucción

```
// MiClase ob1= new MiClase();
```

está comentada pues provoca error de compilación. Recuerda que a toda clase java le asigna un constructor por defecto que se corresponde a un constructor sin parámetros como MiClase() en el ejemplo anterior, pero, si para una clase defino uno o varios

constructores entonces java “retira” la versión de constructor por defecto.

Si por la razón que fuera también interesa la versión sin parámetros, entonces si ya existen otras versiones escritas, hay que escribir explícitamente la versión sin parámetros en el código

Ejemplo: definimos MiClase() y descomentamos instrucciones que daban error

```
class MiClase{
  int x;
  MiClase(){
    System.out.println("Dentro de MiClase()");
    x=0;
  }
  MiClase(int i){
    System.out.println("Dentro de MiClase(int i)");
    x=i;
  }
  MiClase(double d){
    System.out.println("Dentro de MiClase(double i)");
    x= (int) d;
  }
  MiClase(int i, int j){
    System.out.println("Dentro de MiClase(int i, int j)");
    x= i*j;
  }
}
class Unidad2{
    public static void main(String[] args) {
       MiClase ob1= new MiClase();
       MiClase ob2= new MiClase(88);
       MiClase ob3= new MiClase(17.23);
       MiClase ob4= new MiClase(2,4);

       System.out.println("ob1.x: "+ ob1.x);
       System.out.println("ob2.x: "+ ob2.x);
       System.out.println("ob3.x: "+ ob3.x);
       System.out.println("ob4.x: "+ ob4.x);
    }

}
```

Ejercicio U2_B4_E1:

Añade a la clase Bicicleta los constructores necesarios para que funcione el siguiente main()

```
class Bicicleta {
    int velocidade = 0;
    int marcha = 1;
    void cambiarMarcha(int novoValor) {
        marcha = novoValor;
    }
    void acelerar(int incremento) {
        velocidade = velocidade + incremento;
    }
    void frear(int decremento) {
        velocidade = velocidade - decremento;
    }
    void imprimirEstado() {
        System.out.println("Velocidade: "+velocidade+" Marcha: "+marcha);
    }
}
```

```
class Unidad2{
   public static void main(String[] args) {
      // Crea dous obxectos bicicleta
      Bicicleta bicicleta1 = new Bicicleta();
      Bicicleta bicicleta2 = new Bicicleta();
      // Invoca os métodos destes obxectos
      bicicleta1.acelerar(10);
      bicicleta1.cambiarMarcha(2);
      bicicleta1.imprimirEstado();
      bicicleta2.acelerar(10);
      bicicleta2.cambiarMarcha(2);
      bicicleta2.acelerar(10);
      bicicleta2.cambiarMarcha(3);
      bicicleta2.imprimirEstado();
      //un objeto bicicleta inicializado con velocidad 11 y marcha 2
      Bicicleta bicicleta3 = new Bicicleta(11,2);
      bicicleta3.imprimirEstado();
   }
}
```

Ejercicio U2_B4_E2: Escribe los constructores de la clase Persona para que el main produzca la salida indicada

```
class Persona {
   String nombre;
   int edad;
}
class Unidad2{

   public static void main(String args[]) {
      Persona p1= new Persona();//crea una persona con nombre "don nadie" y edad 0 años
      Persona p2= new Persona("Juan");//se crea una persona con el nombre indicado y edad 27 años
      Persona p3= new Persona("Juan",30);//se crea persona con nombre y edad indicados

      System.out.println("p1.nombre: "+ p1.nombre+"  p1.edad: "+p1.edad);
      System.out.println("p2.nombre: "+ p2.nombre+"  p2.edad: "+p2.edad);
      System.out.println("p3.nombre: "+ p3.nombre+"  p3.edad: "+p3.edad);
   }
}
```

SALIDA

```
p1.nombre: Don Nadie  p1.edad: 0
p2.nombre: Juan  p2.edad: 27
p3.nombre: Juan  p3.edad: 30
```