

# POO 2

Juan Carlos Figueroa Duran


Universidad del Bío-Bío  
Departamento de Ciencias de la Computación y Tecnología de la  
Información

- 1 Implementando clase en Java
  - Atributos
  - Constructores
  - Métodos
  - Precondiciones
  - Ambitos de variables
- 2 Creación y uso de objetos en Java
- 3 Uso de parámetros
- 4 Atributos Estáticos
- 5 Constantes
- 6 Métodos estáticos

# Las clases en Java

- En Java, la unidad de programación es la clase de la cual en algún momento se ejemplarizan (crean) objetos.
- La definición de una clase consta de:

**public:** se pueden crear objetos e invocar métodos desde cualquier clase,  
**private:** se pueden crear objetos e invocar métodos sólo en la misma clase.



```
<control de acceso> class <nombre de la clase>{  
    //comentario de autodocumentación  
    <atributos>  
    <constructores>  
    <métodos>  
}
```

- Desde la perspectiva de un lenguaje de programación, los atributos son variables, ubicadas en memoria principal, por lo que se les llama también variables de instancia.
- Ellas implementan las características de una clase y serán usadas por los constructores y métodos de esta.

- Los atributos se definen con la siguiente sintaxis:

`<control de acceso><tipo de dato><nombre atributo>;`

- Donde:

`tipo de dato` puede ser cualquiera, incluyendo otra clase.

`control de acceso` puede ser: `private`, `public` o `protected`

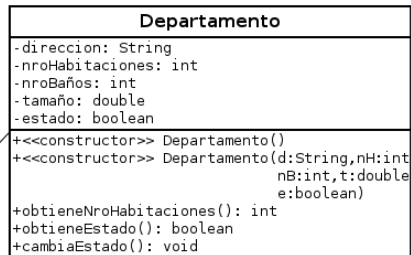
`nombre del atributo` corresponde al nombre con el que se conocerá la característica de la clase que implementa (iniciarse con minúscula y es deseable que tenga un significado que dé cuenta de la característica que representa).

- Un atributo privado en una clase implica que sólo los métodos de esa clase (de ninguna otra) pueden usar (consultar o modificar) dichos atributos.
- Un atributo público, por el contrario, puede ser usado (consultado o modificado) por los métodos de cualquier clase.
- Un atributo protegido tiene sentido cuando se trabaja con clases relacionadas mediante herencia.

Se debe procurar definir atributos privados, de manera de respetar el principio del ocultamiento de la información, una de las ventajas del enfoque orientado a objeto.

# Las clases en Java

```
public class Departamento {  
    //Atributos  
    private String dirección;  
    private int nroHabitaciones;  
    private int nroBaños;  
    private double tamaño;  
    private boolean estado;  
    ...  
}
```



## ¿Qué es un método?

Un método es una porción de código que representa una operación o servicio de una clase. Una clase puede tener tantos métodos como operaciones requiera. Estas operaciones pueden ser servicios para otras clases y/o servicios de uso propio y exclusivo.



Departamento
-direccion: String -nroHabitaciones: int -nroBaños: int -tamaño: double -estado: boolean
+<<constructor>> Departamento() +<<constructor>> Departamento(d:String,nH:int nB:int,t:double e:boolean)  +obtieneNroHabitaciones(): int +obtieneEstado(): boolean +cambiaEstado(): void

- En Java, la creación de un objeto debe realizarse a través de constructores.
- Los constructores son un tipo especial de método que se deben definir como parte de la clase.

## Características de los constructores :

- Es un método que lleva el mismo nombre de la clase.
- Se invoca una sola vez, cuando se crea un nuevo objeto
- Puede usarse para inicializar los atributos de un objeto. Sin embargo, puede no contener instrucción alguna, en este caso, su propósito es permitir la obtención de memoria para las variables de instancia.
- Puede incluir parámetros
- Una clase puede incluir varios constructores con distintas definiciones de parámetros formales.

La sintaxis para definir un constructor es:

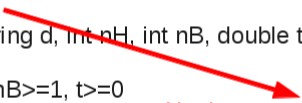
```
<tipo de acceso> <nombre clase>([<lista de parámetros>]) {  
    [<sentencias>] }
```

Un constructor siempre debe ser definido como público, de manera que sea posible, para cualquier clase, crear objetos de cualquier otra clase (definida como pública).

Departamento
-direccion: String -nroHabitaciones: int -nroBaños: int -tamaño: double -estado: boolean
+<<constructor>> Departamento() +<<constructor>> Departamento(d:String,nH:int nB:int,t:double e:boolean) +obtieneNroHabitaciones(): int +obtieneEstado(): boolean +cambiaEstado(): void

# Las constructor en Java

```
public class Departamento {  
    // Atributos  
    private String dirección;  
    private int nroHabitaciones;  
    private int nroBaños;  
    private double tamaño; // metros cuadrados  
    private boolean estado; // true=arrendada, false=disponible;  
    // Constructores  
    public Departamento(){ }  
    public Departamento (String d, int nH, int nB, double t,  
boolean e) {  
        // Precondición: nH>=1, nB>=1, t>=0  
        dirección = d;  
        nroHabitaciones = nH;  
        nroBaños = nB;  
        tamaño = t;  
        estado = e;  
    }  
}
```



No tiene mayor valor definir constructores vacíos, ya que Java considera un constructor por defecto para todas las clases que se definen, asignando valores por defecto a los atributos.

# Las constructor en Java

Producto
-nombre: String
-stockMinimo: int
-stock: int
+<<constructor>> Producto(nom:String,stockMin:int stock:int)

```
public class Producto{
```

```
//Atributos
```

```
private String nombre;
```

```
private int stockMinimo;
```

```
private int stock;
```

```
//Constructor
```

```
public Producto(String nom, int stockMin, int stock){
```

```
//Precondicion: nom<>null, lenght(trim(nom)) <>0, stckMin>0,stck>0
```

```
nombre=nom;
```

```
stockMinimo=stckMin;
```

```
stock=stck;
```

}

## //Métodos

• • •

Mismo nombre de la clase

### Lista de parámetros del constructor

### Asigna valores iniciales a los atributos de la clase


## Métodos :

- Un método es equivalente a un procedimiento (función o subrutina) en un lenguaje de programación tradicional.
- Tienen un nombre, uno o más argumentos y pueden retornar un valor.
- La diferencia es que un método tiene al menos un argumento: *el objeto de invocación*.
- Un método puede acceder variables de instancia e invocar métodos .



Los métodos se definen con la siguiente sintaxis:

```
<control de acceso><tipo valor devuelto><nombre>([<lista de parámetros>]){  
instrucción 1;  
instrucción 2;  
instrucción 3;  
...  
instrucción n-1;  
return [<expresión>;  
}
```



**Cuerpo del método**

## <control de acceso>:

- Puede ser *public*, *private* o *protected*.
- Un método privado sólo puede ser invocado (llamado) por métodos de la misma clase.
- Un método público puede ser invocado por un método en cualquier otra clase. Por lo tanto, se debe considerar que, cuando se define una clase con el fin de que brinde servicios a otras clases, los métodos que implementan tales servicios deben ser **públicos**.

## <tipo valor devuelto>:

- Puede ser void, int, String, double, etc.
- Un método, una vez que es ejecutado, puede devolver o no un valor a quien lo ha invocado.
- Cuando no devuelve valor, se debe indicar void.
- Cuando se devuelve un valor se debe indicar el tipo de dato del valor, por ejemplo String.
- Un método sólo puede devolver un único valor, pudiendo ser este un arreglo, por ejemplo int[].

([<lista de parametros>]):

- La lista de parámetros indica por cada parámetro recibido, el tipo y nombre con que se manejará dentro del método.
- Un método puede recibir muchos parámetros y de tipos diferentes.
- Un método también puede no recibir parámetro alguno, en cuyo caso la lista estará vacía.
- Ejemplos: **obtieneNombre()**, **calculaEdad(int añoActual)**.

## Cuerpo del método:

- Conjunto de instrucciones necesarias para realizar la operación representada por el método.
- Estas instrucciones pueden contener:
  - definición de variables locales (variables sólo válidas dentro del método), asignaciones, ciclos iterativos, condiciones, etc.
  - instrucciones que correspondan a la invocación de otros métodos.
  - uso de los atributos propios de la clase, siendo estos considerados variables globales (variables que son válidas para cualquier método de la clase).

```

1 public class Departamento {
2
3     /* Metodo obtieneNroHabitaciones, devuelve el numero de
       habitaciones */
4     public int obtieneNroHabitaciones( ){
5     return nroHabitaciones;
6     }
7     /* Metodo obtieneEstado, devuelve el estado del departamento*/
8     public String obtieneEstado ( ){
9     if (estado)
10    return "Arrendada";
11    else
12    return "Disponible";
13    }
14    /* Metodo cambioEstado, cambia el atributo estado del departamento
       */
15    public void cambiaEstado ( ){
16    estado = !estado;
17    }
18 }

```



# Ejemplo del uso de Precondiciones

- Las precondiciones son útiles para especificar las condiciones que deben reunir los parámetros de un método o constructor para asegurar el correcto funcionamiento de estos, dichas condiciones se refieren a los valores permitidos a los parámetros.
- El no cumplimiento de una precondición es causal de errores, por lo cual quien invoca un método o constructor debe conocer y respetar las precondiciones establecidas por dicho método o constructor.



```
1 public Producto(String nom, int stckMin, int stck) {  
2 // Precondicion: nom<>null, length(trim(nom))<>0,  
3 // stckMin>0, stck>0  
4 nombre = nom;  
5 stockMinimo = stckMin;  
6 stock = stck;  
7 }
```

Las precondiciones siempre se especifican como un comentario.

- Las variables tienen un ámbito en el cual existen y pueden ser usadas.
- Las variables pueden tener:
  - ámbito local: existen y son válidas en un espacio acotado.
  - ámbito global: pueden ser usadas en cualquier parte de una clase (o programa).
- En Java el ámbito de una variable queda determinada por el lugar en el que fue declarada, siendo los paréntesis { } los que delimitan dicho ámbito.

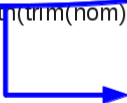
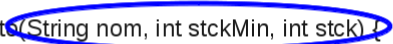
# Los ambito de las variables en Java

```
public class Producto{  
    //Atributos  
    private String nombre;  
    private int stockMinimo;  
    private int stock;
```



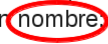
Variables globales, pueden ser usadas en todos los métodos de la clase.

```
    //Constructor  
    public Producto(String nom, int stckMin, int stck) {  
        / Precondición: nom<>null, length(trim(nom))<>0, stckMin>0, stck>0  
        nombre=nom;  
        stockMinimo=stckMin;  
        stock=stck;  
    }
```



Variables locales, sólo pueden ser usadas en el constructor.

```
    //Metodos  
    public String obtieneNombre() {  
        return nombre;  
    }
```



Variable global

# Los ambito de las variables en Java

```
public class Producto{  
    //Atributos  
    private String nombre;  
    private int stockMinimo;  
    private int stock;  
    //Constructor  
    //Metodos  
    public int nroUnidadesBajoStockMinimo() {  
        int nroUnidades;  
        if (!estaBajoStockMinimo()) // Verifica si no está bajo stock  
            nroUnidades = 0;  
        else  
            nroUnidades = stockMinimo - stock;  
        return nroUnidades;  
    }  
}
```

Variables globales, pueden ser usadas en todos los métodos de la clase.

Variables locales

Variable global

# Empleo de la referencia this

- Cuando a un objeto se le invoca un método, dicho objeto es pasado como parámetro en forma implícita.
- Para referenciar tal objeto se usa this, tanto para hacer referencia a los atributos como a los métodos del objeto.
- Cuando se utiliza this en un constructor, se hace referencia al objeto que está siendo creado.

# Empleo de la referencia this

```
public class Producto{  
    //Atributos  
    private String nombre;  
    private int stockMinimo;  
    private int stock;
```



Variables globales, pueden ser usadas en todos los métodos de la clase.

```
    //Constructor  
    public Producto(String nombre, int stockMinimo, int stock) {  
        // Precondición: nom<>null, length(trim(nom))<>0, stckMin>0,  
        stck>0  
        this.nombre=nombre;  
        this.stockMinimo=stockMinimo;  
        this.stock=stock;  
    }  
    //Metodos  
    ...  
}
```

# Creación de objetos en Java

- En primer lugar, dada una clase ya definida, deben crearse objetos. A estos objetos se les podrá solicitar servicios invocando los métodos definidos en la clase a la que pertenecen.
- Un objeto se crea con la siguiente sintaxis:

`<nombre objeto> = new <nombre constructor>([<lista de  
parametros>]);`

- Donde: la variable nombre objeto ha sido previamente declarada, siendo su tipo de dato la clase respectiva.

- Una vez creado un objeto, se podrá invocar alguno de sus métodos mediante una instrucción con la siguiente sintaxis:

`<referencia a objeto>.<nombre método>([<lista de  
parámetros>]);`



# Ejemplo creación de objetos

- Para la clase Departamento, se creará el objeto dep mediante la siguiente instrucción:

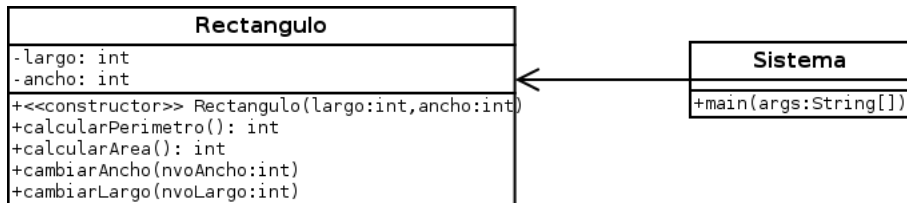
```
Departamento dep;//Declaracion de la variable dep  
dep = new Departamento("Brasil 105",2,1,35.5,false);
```

- La siguiente instrucción escribe la disponibilidad del departamento dep, la disponibilidad se obtiene invocando el método obtieneEstado:

```
System.out.println("El departamento está "+  
dep.obtieneEstado());
```

- La creación de objetos se realiza normalmente en métodos de otra clase. A continuación se ejemplifica, con la definición de la clase Sistema, la creación de objetos de la clase Departamento.

# Diagrama de una aplicación simple




- Antes de crear un objeto se debe declarar la variable que lo referenciará, donde el tipo de la variable corresponderá a la clase del objeto que se creará.
- Para crear un objeto usamos un operador especial llamado **new**.
- El operador **new** invoca al constructor correspondiente y retorna la dirección de memoria que le es asignada durante la ejecución.
- Sólo cuando usamos el operador **new** se asigna memoria al objeto.
- Una vez que un objeto ha sido creado, será posible invocar sus métodos, los que se encuentran definidos en la clase a la que pertenece el objeto.

# Uso de parámetros

- Los parámetros son el mecanismo utilizado por un método para intercambiar datos con constructores u otros métodos de la misma clase o de otras clases. Se distinguen 2 tipos de parámetros, según el lugar donde se usan.
- Se denominan parámetros formales a la lista de parámetros presentes en la cabecera de un método o constructor. Los parámetros formales siempre incluyen tipo de dato y nombre.

```
public void aumentaStock(int cantidad) {  
    //Precondicion cantidad > 0  
    stock = cantidad;  
}
```

 Parámetros formales  
que incluyen tipo y  
nombre.

# Uso de parámetros

- Se denominan **parámetros reales** a la lista de parámetros incluidos en la invocación de un método o constructor. Esta lista debe tener el mismo orden especificado en la lista de parámetros formales del método o constructor correspondiente.
- Los parámetros reales nunca incluyen el tipo de dato, sólo los valores que se asociarán a cada parámetro formal. Pueden ser constantes, variables o expresiones.

```
public void aumentaStock(int cantidad) {  
    //Precondicion cantidad > 0  
    stock = cantidad;  
}
```

Parámetros formales  
que incluyen tipo y  
nombre.

```
prod1.aumentaStock(70);
```

```
...
```

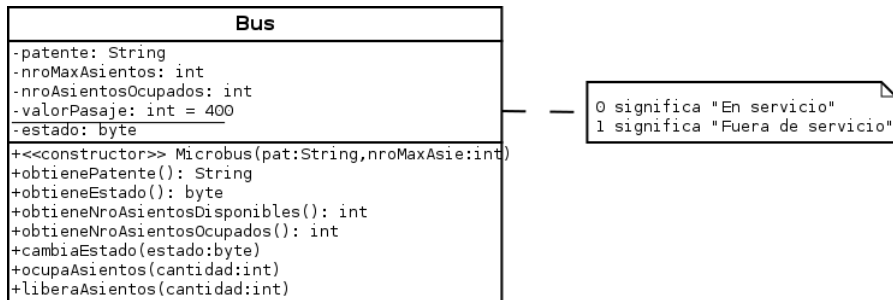
```
prod2.aumentaStock(50);
```

70 y 50 son parámetros reales

- Se denominan **parámetros reales** a la lista de parámetros incluidos en la invocación de un método o constructor. Esta lista debe tener el mismo orden especificado en la lista de parámetros formales del método o constructor correspondiente.
- Los parámetros reales nunca incluyen el tipo de dato, sólo los valores que se asociarán a cada parámetro formal. Pueden ser constantes, variables o expresiones.

<tipo de acceso >static <tipo de dato ><nombre del atributo estático>  
=< *valor inicialización* >

# Diagrama con atributos staticos



# Implementacion atributos estaticos

```
1 public class Bus {
2 // Atributos
3 private String patente;
4 private int nroMaxAsientos;
5 private int nroAsientosOcupados;
6 private static int valorPasaje=400;
7 private byte estado;
8 // Constructor
9 public Microbus(String pat, int nroMaxAsie) {
10 // Precondicion: pat<>null, length(trim(pat))>0, nroMaxAsie>0
11 patente = pat;
12 nroMaxAsientos = nroMaxAsie;
13 estado = 0;
14 // En servicio
15 nroAsientosOcupados = 0;
16 }
17 // Metodos
18 ...
19 }
```

Cada vez que se cree un objeto, el sistema asignará memoria para los atributos del nuevo objeto; sin embargo, el atributo **valorPasaje** existirá en una única posición de memoria. Es una variable global cuyo alcance está

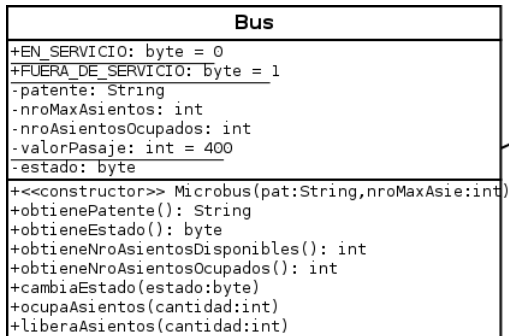


- Pueden ser parte de una clase
- Se pueden utilizar en los métodos de dicha clase durante la programación y es accesible a todos los objetos de esa clase durante ejecución.
- Para ser accedida por métodos de una clase distinta a la que pertenece, una constante debe ser definida con acceso público.

`<tipo de acceso >final static <tipo de dato ><nombre del atributo  
estático>= <valor inicialización >;`

Por convención se usan mayúsculas para su nombre y guión bajo.

# Diagrama con constantes



0 significa "En servicio"  
1 significa "Fuera de servicio"

# Implementacion constantes

```
1 public class Microbus {  
2     // Constantes  
3     public final static byte EN_SERVICIO=(byte) 0;  
4     public final static byte FUERA_DE_SERVICIO=(byte) 1;  
5     ...  
6 }
```

- Una constante no modifica su valor durante ejecución.
- Una vez asignado su valor, mediante inicialización o una operación de asignación, ya no será posible asignarle otro valor.
- En lo posible, la relación entre la constante y su valor deberá realizarse al momento de ser declarada.

# Implementacion constantes

- Por otra parte, cuando un constructor o método de una clase requiera acceder una constante definida en otra clase deberá utilizar la siguiente sintaxis:

<nombre clase >.<nombre constante>

Por ejemplo:

```
1 ...  
2 if (bus1.obtieneEstado() != Bus.FUERA_DE_SERVICIO) {  
3 System.out.println(" y esta en servicio ");  
4 }  
5 ...
```

# Métodos estáticos

- Se usan cuando se desea invocarlo sin que importe el objeto usado para hacer la llamada o bien si se desea hacer la invocación sin necesidad de crear un objeto a través del cual hacer la llamada.

Ejemplo, clase Math:

```
1 double numero = Math.random();
```

- Una clase cuenta con un atributo estático privado que requiere ser accedido o modificado desde métodos de otra(s) clase(s).

<tipo de acceso >static <tipo de dato ><nombre del método  
estático>(

< listadeparametros >

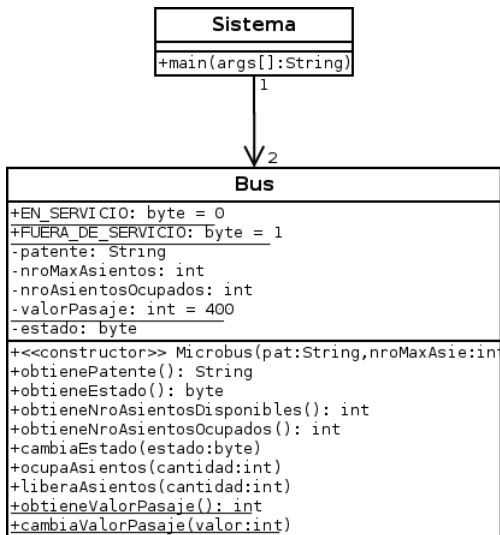
) {

<instrucciones >

return < expresion >

}

# Diagrama con metodos estaticos



# Implementacion metodos estaticos

```
1 // Metodos Estaticos
2 public static int obtieneValorPasaje() {
3 // No debe incluir atributos no estaticos
4 return valorPasaje;
5 }
6 public static void cambiaValorPasaje(int valor){
7 // No incluir atributos no estaticos
8 // Precondicion: nuevoValor>0
9 valorPasaje = nuevoValor;
10 }
```