

# Métodos y Clases

## (Tema 4 y 7 Prog.)

# Métodos:

- ▶ Un **método** es un bloque de código que sólo se ejecuta cuando se llama.
- ▶ A un método se le puede pasar datos, conocidos como parámetros.
- ▶ Los métodos se utilizan para realizar determinadas acciones y también se conocen como **funciones** .
- ▶ ¿Por qué utilizar métodos? Para reutilizar código: define el código una vez y utilízalo muchas veces.

- ▶ Se debe declarar un método dentro de una clase. Se define con el nombre del método, seguido de paréntesis () . Java proporciona algunos métodos predefinidos, como `System.out.println()`, pero también puedes crear tus propios métodos para realizar determinadas acciones:

```
public class Main {  
    static void miMetodo() {  
        // Código a ejecutar  
    }  
}
```

- ▶ `miMetodo()` es el nombre del método
- ▶ `Static` significa que el método pertenece a la clase `Main` y no a un objeto de la clase `Main`. Esto significa que puedes llamar a un método `static` sin crear un objeto de la clase. Por ejemplo, si tienes una clase llamada `Matematicas` que tiene un método `static` llamado `sumar`, puedes invocar ese método así: `Matematicas.sumar(2, 3)`. No necesitas crear un objeto de la clase `Matematicas` para usar el método `sumar`.
- ▶ `Void` significa que este método no tiene un valor de retorno. Por ejemplo, si el método `sumar` devuelve la suma de dos números, debes escribir `int` en vez de `void`. Así: `static int sumar(int a, int b)`.

```
public class Persona {  
String nombre; int edad; //Atributos
```

```
public void saludar() {  
System.out.println("Hola, me llamo " + nombre + " y tengo " + edad + " años.");  
}  
}
```

Para usar el método saludar, necesitas crear un objeto de la clase Persona y asignarle un nombre y una edad. Por ejemplo:

```
Persona p1 = new Persona();  
p1.nombre = "Ana";  
p1.edad = 20;  
p1.saludar(); // Imprime: Hola, me llamo Ana y tengo 20 años.
```

```
Persona p2 = new Persona();  
p2.nombre = "Luis";  
p2.edad = 25;  
p2.saludar(); // Imprime: Hola, me llamo Luis y tengo 25 años.
```

**El método saludar no es static, porque depende de los atributos de cada objeto.**

```
public class Persona {  
String nombre; int edad;  
static int contador = 0; // Contador de objetos creados  
static int suma = 0; // Suma de las edades de los objetos creados  
public void saludar() {  
System.out.println("Hola, me llamo " + nombre + " y tengo " + edad + " años.");  
}  
public static double promedio() {  
return (double) suma / contador; // Calcula el promedio de edad }  
}
```

```
Persona p1 = new Persona(); p1.nombre = "Ana"; p1.edad = 20; Persona.contador++; // Incrementa  
el contador de objetos //p1 es un objeto  
Persona.suma += p1.edad; // Añade la edad del objeto a la suma  
Persona p2 = new Persona(); p2.nombre = "Luis"; p2.edad = 25; Persona.contador++; Persona.suma  
+= p2.edad;  
double prom = Persona.promedio(); // Llama al método static promedio  
System.out.println("El promedio de edad es " + prom); // Imprime: El promedio de edad es 22.5  
El método promedio se puede invocar sin crear un objeto de la clase Persona, solo usando el  
nombre de la clase: Persona.promedio().
```

# Resumiendo:

“objeto”.metodo()-> no es estático

“clase”.metodo()-> si es estático

Los métodos **No estáticos** permiten modelar el **comportamiento específico** de objetos, mientras que los métodos **estáticos** son independientes de las instancias y se utilizan para **funcionalidades comunes** o utilidades.

```
public class Persona {
```

```
    String nombre; int edad; //Atributos
```

```
    public void saludar() {  
        System.out.println("Hola, me llamo " + nombre + " y tengo " + edad + " años.");  
    }
```

```
    public static void main(String[] args) {  
        Persona p1 = new Persona();  
        p1.nombre = "Ana";  
        p1.edad = 20;  
        p1.saludar(); // Imprime: Hola, me llamo Ana y tengo 20 años.  
  
        Persona p2 = new Persona();  
        p2.nombre = "Luis";  
        p2.edad = 25;  
        p2.saludar(); // Imprime: Hola, me llamo Luis y tengo 25 años.  
    }  
}
```



# Llamar a un método

Para llamar a un método en Java, escriba el nombre del método seguido de dos paréntesis () y un punto y coma ;

En el siguiente ejemplo, miMetodo() se utiliza para imprimir un texto (la acción), cuando se llama:

```
public class Main {  
    static void miMetodo() {  
        System.out.println("Vamos!");  
    }  
  
    public static void main(String[] args) {  
        miMetodo();  
    }  
}
```

Un método se puede usar tantas veces como queramos.

# Parámetros y Argumentos

La información se puede pasar a los métodos como parámetro. Los parámetros actúan como variables dentro del método.

Los parámetros se especifican después del nombre del método, dentro del paréntesis. Puedes agregar tantos parámetros como quieras, simplemente sepáralos con una coma.

El siguiente ejemplo tiene un método que toma un `fnameString` llamado como parámetro. Cuando se llama al método, pasamos un nombre, que se usa dentro del método para imprimir el nombre completo:

```
public class Main {  
  
    static void miMetodo(String nombre) {  
        System.out.println(nombre + " García");  
    } //fin miMetodo()  
  
    public static void main(String[] args) {  
        miMetodo("Marcos");  
        miMetodo("Pedro");  
        miMetodo("Ana");  
    } //fin main  
} //fin class
```

Cuando se pasa un parámetro al método, se le llama argumento . Entonces, según el ejemplo anterior: `nombre` un parámetro , mientras `Pedro` `Marcos` y `Ana` son argumentos.

# Múltiples Parámetros

```
public class Main {  
    static void miMetodo(String nombre, int edad) {  
        System.out.println(nombre + " tiene " + edad+"años");  
    }  
  
    public static void main(String[] args) {  
        miMetodo("Marcos", 5);  
        miMetodo("Juan", 8);  
        miMetodo("Ana", 31);  
    }  
}
```

Ten en cuenta que cuando trabaja con varios parámetros, la llamada al método debe tener la misma cantidad de argumentos que parámetros, y los argumentos deben pasarse en el mismo orden.

# Valores de retorno

Void, utilizada en los ejemplos anteriores, indica que el método no debe devolver un valor. Si deseas que el método devuelva un valor, puedes usar un tipo de datos primitivo (como int, string, char, etc.) en lugar de void y usar la palabra ***return*** clave dentro del método:

```
public class Main {  
    static int miMetodo(int x) {  
        return 5 + x;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(miMetodo(3));  
    }  
} // Salida 8
```

# Valores de retorno

Este ejemplo devuelve la suma de los dos parámetros de un método :

```
public class Main {  
    static int miMetodo(int x, int y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(miMetodo(5, 3));  
    }  
} // Salida 8
```

# Valores de retorno

También puedes almacenar el resultado en una variable (recomendado, ya que es más fácil de leer y mantener):

```
public class Main {  
    static int miMetodo(int x, int y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        int z = miMetodo(5, 3);  
        System.out.println(z);  
    }  
}
```

# Un método con If...Else

```
public class Main {  
  
    //Verifica que tenga más de 18 años  
    static void comprobaredad(int edad) {  
  
        if (edad < 18) {  
            System.out.println("Acceso denegado");  
  
        } else {  
            System.out.println("Bienvenido");  
        }  
  
    }  
  
    public static void main(String[] args) {  
        comprobaredad(20);  
    }  
}
```

# Alcance del Método

En Java, solo se puede acceder a las variables dentro de la región en la que se crean. Esto se llama **alcance** .

Las variables declaradas directamente dentro de un método están disponibles en cualquier parte del método después de la línea de código en la que fueron declaradas:

```
public class Main {  
    public static void main(String[] args) {  
  
        // X no existe  
  
        int x = 100;  
  
        // X existe  
        System.out.println(x);  
    }  
}
```



# Alcance del Método

```
public class Main {  
    public static void main(String[] args) {  
  
        // X no existe  
  
        { //Entrada al bloque  
            // X no existe  
  
            int x = 100;  
            // X existe  
            System.out.println(x);  
        } //Salida del bloque  
        // X no existe  
    }  
}
```

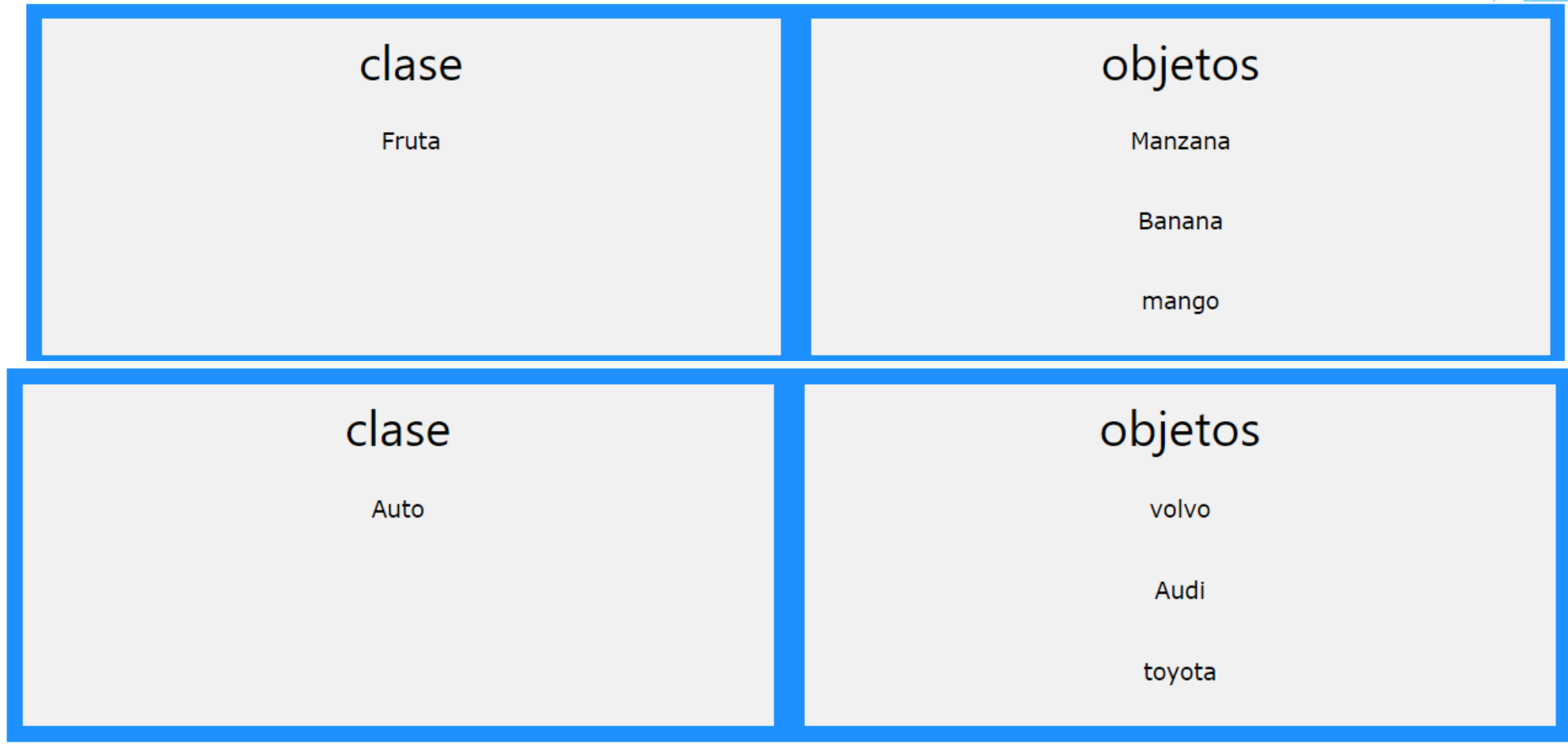
# Clases:

¿Qué es la programación orientada a objetos?

POO significa **Programación Orientada a Objetos** .

- Es más rápida y fácil de ejecutar
- Proporciona una estructura clara para los programas.
- Ayuda a mantener el código Java SECO "No lo repita" y hace que el código sea más fácil de mantener, modificar y depurar.
- Permite crear aplicaciones totalmente reutilizables con menos código y un tiempo de desarrollo más corto.

# ¿Qué son las clases y los objetos?



Cuando se crean los objetos individuales, heredan todas las variables y métodos de la clase.

# ¿Qué son las clases y los objetos?

Todo en Java está asociado con clases y objetos, junto con sus atributos y métodos. Por ejemplo: en la vida real, **un coche es un objeto**. El automóvil **tiene atributos** , como **peso** y **color**, y **métodos** , como **conducción** y **freno**.

Una **clase** es como un constructor de objetos o un "modelo" **para crear objetos**.

## Crear una clase

Para crear una clase, usa la palabra clave `class`:

*Crea una clase llamada "Main" con una variable x:*

```
public class Main {  
    int x = 5;  
}
```

# Crear un objeto

En Java, un objeto se crea a partir de una clase. Ya hemos creado la clase denominada Main, por lo que ahora podemos usarla para crear objetos.

Para crear un objeto de Main, especifique el nombre de la clase, seguido del nombre del objeto y utilice la palabra clave **new**:

## *Ejemplo*

*Crea un objeto llamado " myObj" e imprime el valor de x:*

```
public class Main {  
    int x = 5;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        System.out.println(myObj.x);  
    }  
}
```

# Usando múltiples clases

También puedes crear un objeto de una clase y acceder a él en otra clase. Esto se utiliza a menudo para una mejor organización de las clases (una clase tiene todos los atributos y métodos, mientras que la otra clase contiene el `main()` método (código a ejecutar)).







Recuerde que el nombre del archivo java debe coincidir con el nombre de la clase. En este ejemplo, hemos creado dos archivos en el mismo directorio/carpeta:

Main.java

secundaria.java

```
public class secundaria {  
    int x = 5;  
}  
  
class Main {  
    public static void main(String[] args) {  
        secundaria myObj = new secundaria();  
        System.out.println(myObj.x);  
    }  
}
```

# Usando múltiples clases

- ▼  programacion
  - ▼  src
    - >  programacion
    - ▼  segundo\_trimestre
      - >  Main.java
      - >  secundaria.java

```
package segundo_trimestre;

public class Main {

    public static void main(String[] args) {

        secundaria myObj = new secundaria();
        System.out.println(myObj.x);
    }

}
```

```
package segundo_trimestre;

public class secundaria {
    int x = 5;
}
```

# Atributos de clase Java

En el ejemplo anterior, utilizamos el término “int x=5;”. En realidad, es un atributo de la clase. Se podría decir que los atributos de clase son variables dentro de una clase:

```
public class Main {  
    int x = 5;//atributo  
    int y = 3;//atributo  
}
```



# Accediendo a los atributos

Puede acceder a los atributos creando un objeto de la clase y utilizando “.”:

El siguiente ejemplo creará un objeto de la clase Main, con el nombre myObj. Usamos el atributo x del objeto para imprimir su valor:

## *Ejemplo*

*Cree un objeto llamado " myObj" e imprima el valor de x:*

```
public class Main {  
    int x = 5;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        System.out.println(myObj.x);  
    }  
}
```

# Modificar atributos

También se puede modificar los valores de los atributos o anular los valores existentes:

*Ejemplo*

*Establezca el valor de x en 40:*

```
public class Main {  
    int x;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        myObj.x = 40;  
        System.out.println(myObj.x);  
    }  
}
```

*Ejemplo*

*Cambie el valor de x a 25:*

```
public class Main {  
    int x = 10;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        myObj.x = 25; // x es ahora 25  
        System.out.println(myObj.x);  
    }  
}
```

# Modificar atributos

También puedes mantener un atributo constante:

*final* es útil cuando desea que una variable almacene siempre el mismo valor, como PI (3.14159...).

```
public class Main {  
    final int x = 10;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        myObj.x = 25; // generará un error: cannot assign a  
        value to a final variable  
        System.out.println(myObj.x);  
    }  
}
```

# Múltiples objetos

Si crea varios objetos de una clase, puede cambiar los valores de los atributos en un objeto, sin afectar los valores de los atributos en el otro:

## *Ejemplo*

*Cambie el valor de x a 25 para myObj2 y deja a myObj1 sin cambios:*

```
public class Main {  
    int x = 5;  
  
    public static void main(String[] args) {  
        Main myObj1 = new Main(); // creas el objeto 1  
        Main myObj2 = new Main(); // creas el objeto 2  
  
        myObj2.x = 25;  
        System.out.println(myObj1.x); // salida 5  
        System.out.println(myObj2.x); // salid 25  
    }  
}
```

# Múltiples atributos

Puede especificar tantos atributos como desee:

```
public class Main {  
    String nombre= "Marta";  
    String apellido = "García";  
    int edad = 24;  
  
    public static void main(String[] args) {  
        Main persona = new Main();  
        System.out.println("Nombre: " + persona.nombre + " "  
+ persona.apellido);  
        System.out.println("Edad: " + persona.edad);  
    }  
}
```

# Métodos de la clase Java

## *Ejemplo*

*Cree un método nombrado miMetodo():*

```
public class Main {  
    static void miMetodo() {  
        System.out.println("Vamos!");  
    }  
  
    public static void main(String[] args) {  
        miMetodo();  
    }  
}
```

# Estático VS Público

A menudo verá programas Java que tienen atributos y métodos `static` `public`

En el ejemplo anterior, creamos un método ***static***, lo que significa que se puede acceder a él sin crear un objeto de la clase, a diferencia de ***public***, al que solo se puede acceder mediante objetos:

# Estático VS Público

Ejemplo:

```
public class Main {  
    // Static  
    static void miMetodoStatic() {  
        System.out.println("Los metodos estáticos se usan sin  
crear objetos");  
    }  
  
    // Public  
    public void miMetodoPublic() {  
        System.out.println("Los métodos públicos se usan  
creando objetos");  
    }  
  
    // Main  
    public static void main(String[] args) {  
        miMetodoStatic(); // Llama al método static  
        // miMetodoPublic(); Esto daría un error  
  
        Main myObj = new Main(); // Creas un objeto Main  
        myObj.miMetodoPublic(); // Llama al método public  
    }  
}
```



# Métodos de acceso con un objeto

Ejemplo:

Crea un objeto llamado micoche. Llama a los métodos atodogas() y velocidad().

```
public class Coche {  
  
    // Crea el metodo atodogas()  
    public void atodogas() {  
        System.out.println("El V8 va a explotar!");  
    }  
  
    // Crea el método velocidad() y pasa la velocidad como  
    // parámetro  
    public void velocidad(int maxvelocidad) {  
        System.out.println("La velocidad max es: " + maxvelocidad);  
    }  
  
    public static void main(String[] args) {  
        Coche miCoche = new Coche();  
        miCoche.atodogas();    // Llamada a función  
        miCoche.velocidad(200);    // Llamada a función  
    }  
}  
  
// El V8 va a explotar!  
// La velocidad max es : 200
```

# Usando múltiples clases

Ejemplo:

## Clase Secundaria

```
public class Secundaria {  
    public void atodogas() {  
        System.out.println("Llamando al 112");  
    }  
  
    public void velocidad(int maxvelocidad) {  
        System.out.println("Velocidad max: " + maxvelocidad);  
    }  
}
```

## Clase Principal

```
class Main {  
    public static void main(String[] args) {  
        Secundaria miCoche = new Secundaria();  
        // Crea un objeto miCoche  
        miCoche.atodogas(); // Llamada al metodo  
  
        miCoche.velocidad(200); // Llamada al metodo  
    }  
}
```

# Constructores de Java

Un constructor en Java es un método especial que se utiliza para inicializar objetos. Se llama al constructor cuando se crea un objeto de una clase. Se puede utilizar para establecer valores iniciales para los atributos del objeto:

```
public class Main {  
    int x; // Atributo de clase  
  
    // Constructor de clase Main  
    public Main() {  
        x = 5; // inicializa x  
    }  
  
    public static void main(String[] args) {  
        Main myObj = new Main(); // (Esto llama al  
        constructor)  
        System.out.println(myObj.x); // Imprime x  
    }  
}
```

# Constructores de Java

- Tenga en cuenta que el nombre del constructor debe coincidir con el nombre de la clase y no puede tener un tipo de retorno como void.
- También ten en cuenta que se llama al constructor cuando se crea el objeto.
- Todas las clases tienen constructores de forma predeterminada: si no creas uno, Java crea uno por defecto. Sin embargo, entonces no podrás establecer valores iniciales para los atributos del objeto.

# Parámetros del constructor

Los constructores también pueden tomar parámetros, que se utilizan para inicializar atributos.

El siguiente ejemplo agrega un parámetro *int* y al constructor. Dentro del constructor configuramos x e y (x=y). Cuando llamamos al constructor, le pasamos un parámetro al constructor (5), que establecerá el valor de x en 5:

```
public class Main {  
    int x;  
  
    public Main(int y) {  
        x = y;  
    }  
  
    public static void main(String[] args) {  
        Main myObj = new Main(5);  
        System.out.println(myObj.x);  
    }  
}
```

# Parámetros del constructor

```
public class Main {  
    int año_modelo;  
    String nombre_modelo;  
  
    public Main(int año, String nombre) {  
        año_modelo = año;  
        nombre_modelo = nombre;  
    }  
  
    public static void main(String[] args) {  
        Main miCoche = new Main(1969, "Mustang");  
        System.out.println(miCoche.año_modelo + " " +  
miCoche.nombre_modelo);  
    }  
}
```

# Modificadores de Java

Public-> La clase es accesible por cualquier otra clase: `public class Main {`

Default-> La clase solo es accesible por clases del mismo package: `class Main{`

Final-> Una clase marcada como final no puede ser subclase. Es decir, no se puede extender o heredar de una clase final. No se puede crear una clase que extienda una clase final

Para atributos:

Public-> la variable es accesible por cualquier otra clase

Default-> la variable solo es accesible por clases del mismo package

Private-> la variable solo es accesible en la clase declarada

Protected-> la variable solo es accesible en el mismo package y por las subclases.

```
public class Main {  
    final int x = 10;  
    final double PI = 3.14;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        myObj.x = 50; // generará un error  
        myObj.PI = 25; // generará un error  
        System.out.println(myObj.x);  
    }  
}
```

# Encapsulación

El significado de Encapsulación es garantizar que los datos "sensibles" estén ocultos a los usuarios. Para lograr esto, debes:

Declarar variables/atributos de clase como private.

Proporcionar métodos públicos de obtención y configuración para acceder y actualizar el valor de una variable private.

Son los famosos: GETTERS Y SETTERS

```
public class Persona {  
    private String nombre; // private = clases de fuera no  
    pueden acceder  
    // Getter  
    public String getNombre() {  
        return nombre;  
    }  
    // Setter  
    public void setNombre(String nomb) {  
        this.nombre = nomb;  
    }  
}
```



# Herencia de Java

En Java es posible heredar atributos y métodos de una clase a otra. Agrupamos el “concepto de herencia” en dos categorías:

subclase (secundaria): la clase que hereda de otra clase

superclase (padre): la clase de la que se hereda

Para heredar de una clase, se usa la palabra `extends`.

En el siguiente ejemplo, la Coche (subclase) hereda los atributos y métodos de la Vehículo (superclase):

# Herencia de Java

```
class Vehiculo {  
    protected String marca = "Chabas";  
    public void inicio() {  
        System.out.println("En marcha!");  
    }  
}  
class Coche extends Vehiculo {  
    private String nombreModelo = "Dam1";  
    public static void main(String[] args) {  
  
        // Creamos objeto coche  
        Coche miCoche = new Coche();  
  
        // Llamamos a inicio() (de la clase Vehiculo)  
        miCoche.inicio();  
  
        System.out.println(miCoche.marca + " " + miCoche.nombreModelo);  
    }  
}
```

# Herencia de Java

Establecemos el atributo de marca en Vehículo como un protected. Si estuviera configurado en private, la clase Coche no podría acceder a él.

¿Por qué y cuándo utilizar la "herencia"?

- Es útil para la reutilización del código: reutiliza atributos y métodos de una clase existente cuando creas una nueva clase.

Si no deseas que otras clases hereden de una clase, utiliza: final

# Polimorfismo de Java

La herencia nos permite heredar atributos y métodos de otra clase. El polimorfismo utiliza esos métodos para realizar diferentes tareas. Esto nos permite realizar una misma acción de diferentes maneras.

Por ejemplo, piensa en una superclase llamada `Animal` que tiene un método llamado `sonidoAnimal()`. Las subclases de animales podrían ser cerdos, gatos, perros, pájaros, y también tienen su propia implementación de un sonido animal (el cerdo gruñe y el gato maúlla, etc.):

# Polimorfismo de Java

```
class Animal {  
    public void sonidoAnimal() {  
        System.out.println("El animal hace un sonido");  
    }  
}
```

```
class Gallina extends Animal {  
    public void sonidoAnimal() {  
        System.out.println("La gallina cacarea");  
    }  
}
```

```
class Lobo extends Animal {  
    public void sonidoAnimal() {  
        System.out.println("El lobo aúlla");  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        Animal miAnimal = new Animal();  
        Animal gallina = new Gallina();  
        Animal lobo = new Lobo();  
        //>Atención: si añades Lobo lobo=new Lobo() no es  
        polimorfismo, estarías llamando al método de la clase  
        Animal  
        miAnimal. sonidoAnimal();  
        miGallina. sonidoAnimal();  
        miLobo. sonidoAnimal();  
    }  
}
```

# Polimorfismo de Java

```
class Animal {  
    public void sonidoAnimal() {  
        System.out.println("El animal hace un sonido");  
    }  
}
```

```
class Gallina extends Animal {  
    public void sonidoAnimal() {  
        System.out.println("La gallina cacarea");  
    }  
}
```

```
class Lobo extends Animal {  
    public void sonidoAnimal() {  
        System.out.println("El lobo aúlla");  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        Animal miAnimal = new Animal();  
        Animal gallina = new Gallina();  
        Animal lobo = new Lobo();  
        //>Atención: si añades Lobo lobo=new Lobo() no es  
        polimorfismo, estarías llamando al método de la clase  
        Animal  
        miAnimal. sonidoAnimal();  
        miGallina. sonidoAnimal();  
        miLobo. sonidoAnimal();  
    }  
}
```

# Abstracciones de Java

La abstracción de datos es el proceso de ocultar ciertos detalles y mostrar solo información esencial al usuario.

La abstracción se puede lograr con clases abstractas o interfaces

**Clase abstract:** es una clase restringida **que no se puede utilizar para crear objetos** (para acceder a ella se debe heredar de otra clase).

**Método abstract:** **solo se puede utilizar en una clase abstracta y no tiene cuerpo. El cuerpo lo proporciona la subclase.**

Una clase abstracta puede tener métodos tanto abstractos como regulares/normales:

# Abstracciones de Java

```
abstract class Animal {  
    public abstract void sonidoAnimal();  
    public void dormir() {  
        System.out.println("zzz");  
    }  
}
```

Para acceder a la clase abstracta, se debe heredar de otra clase. Convirtamos la clase `Animal` que usamos en el capítulo Polimorfismo en una clase abstracta:



# Abstracciones de Java

// clase Abstract

```
abstract class Animal {  
    // el método abstract no tiene cuerpo  
    public abstract void animalSound();  
    // metodo normal  
    public void dormir() {  
        System.out.println("zzz");  
    }  
}
```

// Subclass (hereda de Animal)

```
class Gallina extends Animal {  
    public void sonidoAnimal() {  
        System.out.println("La gallina cacarea");  
    }  
}
```

class Main {

```
    public static void main(String[] args) {  
        Gallina miGallina = new Gallina();  
        miGallina.sonidoAnimal();  
        miGallina.dormir();  
    }  
}
```

# Interfaces de Java

Una interfaz es una clase abstracta que se utiliza para agrupar métodos relacionados con cuerpos vacíos

```
interface Animal {  
    public void sonidoAnimal();  
    public void correr();  
  
}
```

Para acceder a los métodos de la interfaz, la interfaz debe ser "implementada" (algo así como heredada) por otra clase con la implements palabra clave (en lugar de extends). El cuerpo del método de interfaz lo proporciona la clase "implementar":

# Interfaces de Java

```
// Interface
interface Animal {
void sonidoAnimal();//si no se especifica nada, el método por defecto es
público
void dormir();
}

class Gallina implements Animal {
    public void sonidoAnimal() {

        System.out.println("La gallina cacarea");
    }
    public void dormir() {
        System.out.println("Siesta de 10 a 11");
    }
}

class Main {
    public static void main(String[] args) {
        Gallina gallina1 = new Gallina();
        gallina1.sonidoAnimal();
        gallina1.dormir();
    }
}
```

# Interfaces de Java

Al igual que las clases abstractas , las interfaces no se pueden utilizar para crear objetos (en el ejemplo anterior, no es posible crear un objeto "Animal" )

Los métodos de interfaz no tienen cuerpo: el cuerpo lo proporciona la clase "implementar"

Al implementar una interfaz, debe anular todos sus métodos.

Los métodos de interfaz son por defecto abstracty public

Los atributos de la interfaz son por defecto publicy staticfinal

Una interfaz no puede contener un constructor (ya que no se puede utilizar para crear objetos)

# Múltiples Interfaces de Java

```
interface Interfaz1 {  
    public void miMetodo1();  
}
```

```
interface Interfaz2 {  
    public void miMetodo2();  
}
```

```
class Prueba implements Interfaz1, Interfaz2 {  
    public void miMetodo1() {  
        System.out.println("Hola");  
    }  
    public void miMetodo2() {  
        System.out.println("Adiós");  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        Prueba prueba = new Prueba();  
        prueba.miMetodo1();  
        prueba.miMetodo2();  
    }  
}
```

# Enumeraciones de Java

Un *enum* es una "clase" especial que representa un grupo de constantes (tipo *final*). También pueden declararse dentro de una clase.

Para crear un enum, ten en cuenta que las letras deben estar en mayúsculas:

```
enum Tipos {  
    APOYO,  
    DISTANCIA,  
    LOGIA  
}
```

```
Tipos miclase=Tipos.APOYO;
```

```
public class Main {  
    enum Tipos {  
        APOYO,  
        DISTANCIA,  
        LOGIA  
    }  
}
```

```
    public static void main(String[] args) {  
        Tipos miclase=Tipos.APOYO;  
  
        System.out.println(miclase);  
    }  
}
```

# Enumeraciones de Java

```
enum Tipos {  
    APOYO,  
    DISTANCIA,  
    LOGIA  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Tipos miclase=Tipos.APOYO;  
  
        switch(miclase) {  
            case APOYO:  
                System.out.println(" Clase Apoyo ");  
                break;  
            case DISTANCIA:  
                System.out.println(" Clase Distancia ");  
                break;  
            case LOGIA:  
                System.out.println("Clase Logia");  
                break;  
        }  
    }  
}
```

# Enumeraciones de Java

Recorrer una enumeración

El tipo de enumeración tiene un método `values()` que devuelve todas las constantes de enumeración. Este método es útil cuando desea recorrer las constantes de una enumeración:

Utiliza enumeraciones cuando tenga valores que sepa que no van a cambiar

```
for (Tipos miclase : Tipos.values()) {  
    System.out.println(miclase);  
}
```

HACER EJERCICIO DE LA BARAJA



# HashMap de Java

Si bien los Arrays con colecciones listas enumeradas por un índice numérico, con los HashMap tenemos que podemos indicarlo con un índice personalizado, como por ejemplo String. Tenemos por un lado la clave/index y por otro el valor.

```
// Import HashMap class
import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        //
        HashMap<String, String> ciudadesCapitales = new HashMap<String, String>();

        // Indice país, ciudad como valor(País, Capital)
        ciudadesCapitales.put("China", "Pekin");
        ciudadesCapitales.put("Nigeria", "Abuya");
        ciudadesCapitales.put("Rusia", "Moscú");
        ciudadesCapitales.put("España", "La villa de Madrid");
        System.out.println(ciudadesCapitales);
        System.out.println(ciudadesCapitales.get("España")); //Imprime La villa de Madrid
        ciudadesCapitales.remove("España"); //borra el registro
        ciudadesCapitales.clear(); //lo borra todo
    }
}
```

# Recorrer un HashMap

Recorrer los elementos de un HashMap con bucle.

Nota: Utiliza el método `keySet()` si solo deseas imprimir el índice y utilice el método `values()` si solo deseas los valores:

```
//Imprimir el índice/key
for (String i : ciudadesCapitales.keySet()) {
    System.out.println(i);
} //Resultado: China Nigeria Rusia España
```

```
// Imprimir el valor
for (String i : ciudadesCapitales.values()) {
    System.out.println(i);
} Pekín Abuya Moscú La villa de Madrid
```

```
// Imprimir index y valor (Países y Capitales)
for (String i : capitalCities.keySet()) {
    System.out.println("País: " + i + " Capital: " +
ciudadesCapitales.get(i));
}
```