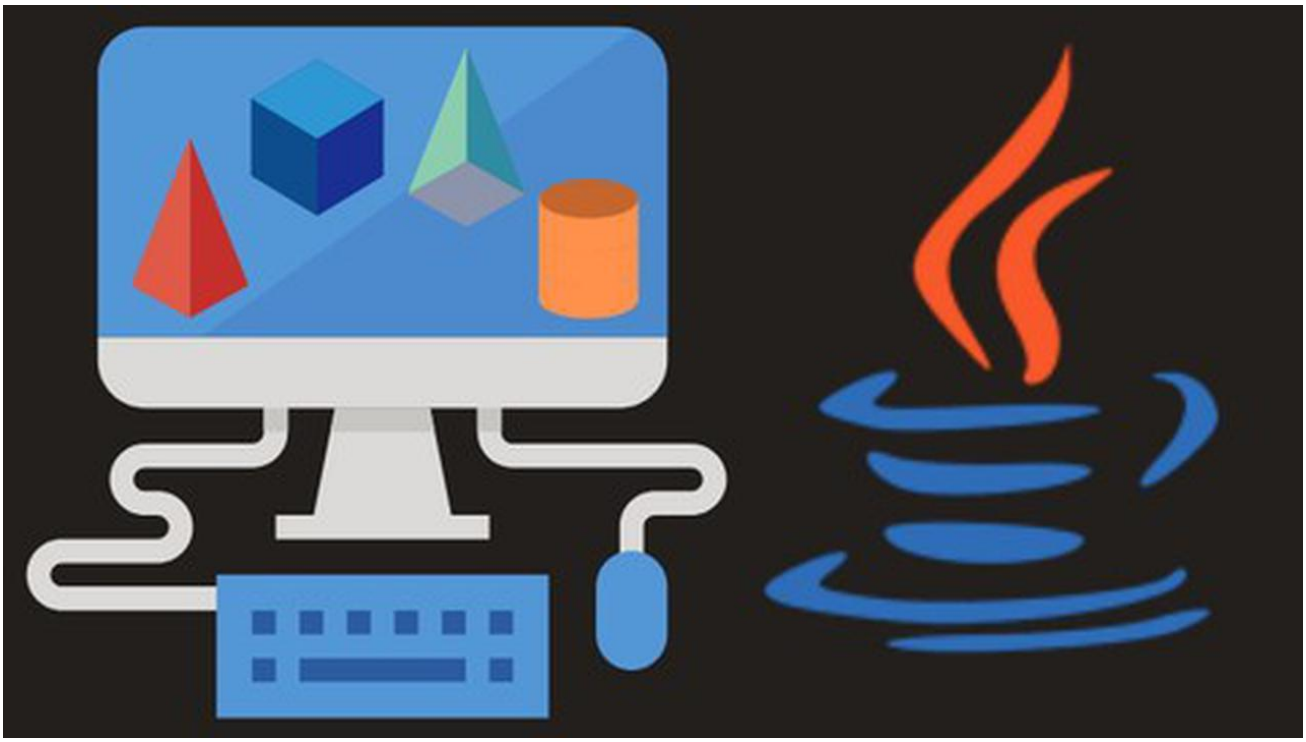
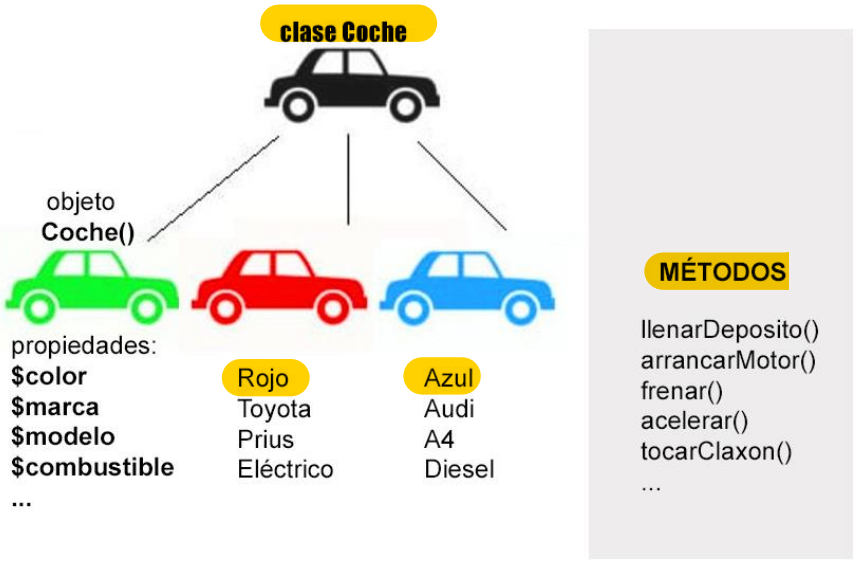
Java - ¿Qué es OOP? Programación Orientada a Objetos



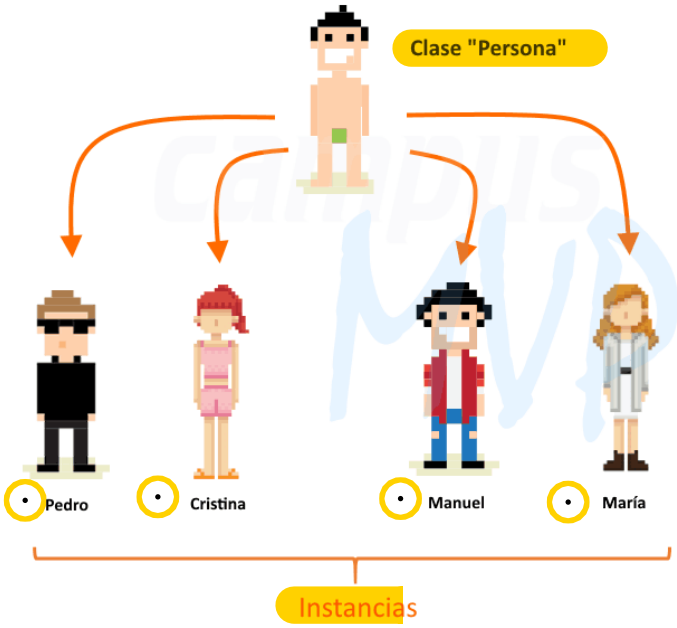
OOP significa **Programación Orientada a Objetos en inglés**.

- OOP es más rápido y fácil de ejecutar sobre la programación estándar
- OOP proporciona una estructura clara para los programas
- OOP ayuda a mantener el código Java SECO "No se repita", y hace que el código sea más fácil de mantener, modificar y depurar
- OOP permite crear aplicaciones reutilizables completas con menos código y menor tiempo de desarrollo

Java - ¿Qué son las clases y los objetos?



Otro ejemplo:



Por lo tanto, una clase es una plantilla para objetos y un objeto es una instancia de una clase.

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

Piensa, crea, comenta... una clase

Clases/Objetos Java

Crear una clase

Para crear una clase, utilice la palabra clave : class Es como un molde para crear objetos

Main.java

Cree una clase denominada "Main" con una variable x:

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

Crear un objeto

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);
  }

}
```

Objeto creado/instanciado a partir de una clase

Múltiples objetos

Ejemplo

Crear dos objetos de Main:

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

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

Atributo al que queremos acceder

Uso de varias clases

- Main.java
- Second.java

El nombre de la clase debe ser el mismo nombre que el del archivo

Main.java

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

Second.java

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

}
```

Cuando se hayan compilado ambos archivos:

```
C:\Users\Your Name>javac Main.java
C:\Users\Your Name>javac Second.java
```

Ejecute el archivo Second.java:

```
C:\Users\Your Name>java Second
```

Y el resultado será:

```
5
```

Atributos de la clase

Ejemplo

Cree una clase llamada " Main " con dos atributos: x y y:

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

Atributos de la clase

Otro término para los atributos de clase es **campos**.

Acceso a atributos

Puede tener acceso a los atributos creando un objeto de la clase y utilizando la sintaxis de punto (.):

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 puede modificar los valores de los atributos:

Ejemplo

Establezca el valor de x a 40:

```
public class Main {
 int x;

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

O sobreescribir los valores existentes:

Ejemplo

Cambiar 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 is now 25
  System.out.println(myObj.x);
 }
}
```

Si no desea la cambiar los valores existentes, declare el atributo como final:

Ejemplo

```
public class Main {
 final int x = 10;

 public static void main(String[] args) {
  Main myObj = new Main();
  myObj.x = 25; //generará un error: no se puede asignar un valor a una varuable final
  System.out.println(myObj.x);
 }
```

```
}
```

La palabra clave final es útil cuando se quiere que una variable almacene siempre el mismo valor, como PI (3.14159...).

La palabra clave final se denomina "modificador".

Múltiples objetos

Ejemplo

Cambie el valor de x a 25 en myObj2 , y deje x en myObj1 sin cambios:

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

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

Múltiples atributos

Puede especificar tantos atributos como desee:

Ejemplo

```
public class Main {
  String fname = "John";
  String lname = "Doe";
  int age = 24;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
```

Métodos de la clase

Los métodos se declaran dentro de una clase y se utilizan para realizar ciertas acciones:

Ejemplo

Cree un método denominado myMethod() en Main:

```
public class Main {
 static void myMethod() {
   System.out.println("Hello World!");
 }
}
```

Ejemplo

En el interior de main , llame myMethod():

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}
// Outputs "Hello World!"
```

Estático vs. no estático

Ejemplo

Un ejemplo para demostrar las diferencias entre **los métodos** static y public:

```
public class Main {
  // Static method
 static void myStaticMethod() {    Podemos acceder al método sin generar ningún objeto
   System.out.println("Método Static puede ser  llamado sin crear el objeto");
  }

  // Public method
 public void myPublicMethod() {
   System.out.println("Método Public debe ser llamado creando objetos");
  }

  // Main method
  public static void main(String[] args) {
   myStaticMethod(); // Llama al método Static
    // myPublicMethod(); Debería dar error de compilación.
    Main myObj = new Main(); // Crea un objeto de  Main
   myObj.myPublicMethod(); // Llama al método public en el objecto
  } Tenemos que generar un objeto para acceder al método
}
```

Acceso a métodos con un objeto

Ejemplo

Crear un objeto Car denominado myCar. Llame a los métodos fullThrottle() y speed() en el objeto myCar y ejecute el programa:

```
// Crea una clase Main
public class Main {

  // Crea un método fullThrottle()
  public void fullThrottle() {
    System.out.println("El coche está circulando tan rápido como puede!");
  }

  // Crea un método speed() y añade un parámetro.
  public void speed(int maxSpeed) {
    System.out.println("Max velocidad es: " + maxSpeed);
  }

  // Dentro de main, llame a los métodos en el objeto myCar
```

```
  public static void main(String[] args) {
    Main myCar = new Main();   // Crea un objeto myCar
    myCar.fullThrottle();     // Llama al método fullThrottle()
    myCar.speed(200);         // Llama al método speed()
  }
}

// El coche está circulando tan rápido como puede!
// Max velocidad es: 200
```

Ejemplo explicado

1) Creamos una clase Main personalizada con la palabra clave class.

2) Creamos los métodos fullThrottle() y speed() en la clase Main.

3) El método fullThrottle() y el método speed()imprimirán algo de texto, cuando se llamen.

4) El método speed() acepta un parámetro int llamado maxSpeed - usaremos esto en **8)**.

5) Para usar la clase Main y sus métodos, necesitamos crear un **objeto** de la Clase Main.

6) Luego, vaya al método main(), que ya sabe que es un método Java incorporado que ejecuta su programa (se ejecuta cualquier código dentro de main).

7) Usando la palabra clave new creamos un objeto con el nombre myCar.

8) Luego, llamamos a los métodos fullThrottle() y speed() en el objeto myCar, y ejecutamos el programa usando el nombre del objeto (myCar), seguido de un punto (**.**), seguido del nombre del método (fullThrottle(); y speed(200);). Observe que agregamos un parámetro int de **200** dentro del método speed().

Recuerda que…

El punto (**.**) se utiliza para acceder a los atributos y métodos del objeto.

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

Una clase debe tener un nombre de archivo coincidente (Main y **Main.java**).

Constructores de Java

Ejemplo

Crear un constructor: Para inicializar los atributos

```
// Create una clase Main
public class Main {
 int x;  // Crear un atributo

 // Crea un constructor para la clase  Main
 public Main() {   Tiene que coicidir con el nombre de la clase!!!
  x = 5;  // Asigna el valor inicial para el atributo x
 }

 public static void main(String[] args) {
  Main myObj = new Main(); // Crea un objecto de la clases Main (Se llamará el constructor)
  System.out.println(myObj.x); // Imprimimos el valos de x
 }
}
// Outputs 5
```

Parámetros del constructor

Ejemplo

```
public class Main {
 int x;

              Parámetro para inicializar la variable
 public Main(int y) {
  x = y;
 }

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

// Outputs 5
```

Puede tener tantos parámetros como desee:

Ejemplo

```
public class Main {
 int modelYear;
```

```
  String modelName;

  public Main(int year, String name) {
    modelYear = year;
    modelName = name;
  }

  public static void main(String[] args) {
    Main myCar = new Main(1969, "Mustang");
    System.out.println(myCar.modelYear + " " + myCar.modelName);
  }
}

// Outputs 1969 Mustang
```

Modificadores

La palabra clave public que aparece en casi todos los ejemplos:

```
public class Main
```

Modificadores de acceso

Para **las clases**, puede usar cualquiera de los *siguientes;* public *o default*

Modificador	Descripción
public	La clase es accesible por cualquier otra clase
default	Solo se puede acceder a la clase mediante clases del mismo paquete. Esto se usa cuando no se especifica un modificador.

Para **atributos, métodos y constructores**, puede utilizar uno de los siguientes:

Modifier	Description
public	El código es accesible para todas las clases
private	Solo se puede acceder al código dentro de la clase declarada
default	Solo se puede acceder al código en el mismo paquete. Esto se usa cuando no se especifica un modificador.
protected	Se puede acceder al código en el mismo paquete y subclases.

Modificadores sin acceso

Para **las clases**, puede usar final o abstract:

Modifier	Description
final	La clase no puede ser heredada por otras clases (aprenderá más sobre la herencia en el capítulo Herencia)

abstract	La clase no se puede utilizar para crear objetos (para acceder a una clase abstracta, debe heredarse de otra clase.

Para **los atributos y métodos**, puede utilizar uno de los siguientes:

Modifier	Description
final	Los atributos y métodos no se pueden reemplazar/modificar
static	Atributos y métodos pertenece a la clase, en lugar de a un objeto
abstract	Sólo se puede utilizar en una clase abstracta, y sólo se puede utilizar en métodos. El método no tiene un cuerpo, por ejemplo, abstract void run();. El cuerpo es proporcionado por la subclase (heredada de).
transient	Los atributos y métodos se omiten al serializar el objeto que los contiene
synchronized	Solo se puede acceder a los métodos mediante un subproceso a la vez
volatile	El valor de un atributo no se almacena en caché localmente y siempre se lee desde la "memoria principal"

Final

Si no desea la capacidad de invalidar los valores de atributo existentes, declare los atributos como final:

Ejemplo

```
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: no se puede asignar un valor a una variable final
    myObj.PI = 25; generará un error: no se puede asignar un valor a una variable final
    System.out.println(myObj.x);
  }
}
```

Estático

Ejemplo

Un ejemplo para demostrar las diferencias entre los métodos static y public:

```
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println("Se puede llamar a métodos estáticos sin crear objetos ");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Se debe llamar a los métodos públicos mediante la creación de objetos ");
  }
  // Main method
  public static void main(String[ ] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would output an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method
  }
}
```

Abstracto

Ejemplo

```
// Code from filename: Main.java
// abstract class
abstract class Main {
  public String fname = "John";
  public int age = 24;
  public abstract void study(); // abstract method
}

// Subclass (inherit from Main)
class Student extends Main {
  public int graduationYear = 2018;
  public void study() { // the body of the abstract method is provided here
    System.out.println("Studying all day long");
  }
}
// End code from filename: Main.java

// Code from filename: Second.java
class Second {
  public static void main(String[] args) {
    // create an object of the Student class (which inherits attributes and methods from Main)
    Student myObj = new Student();

    System.out.println("Name: " + myObj.fname);
    System.out.println("Age: " + myObj.age);
    System.out.println("Graduation Year: " + myObj.graduationYear);
    myObj.study(); // call abstract method
  }
}
```

Encapsulación

Get y Set

Ejemplo

```
public class Person {
 private String name; // private = restricted access

 // Getter
 public String getName() {
  return name;
 }

 // Setter
 public void setName(String newName) {
  this.name = newName;
 }
}
```

Ejemplo explicado

El método getname devuelve el valor de la variable .

El método set toma un parámetro (newName) y lo asigna a la variable name. La palabra clave this se utiliza para hacer referencia al objeto actual.

Sin embargo, como la variable name se declara como private, **no podemos** acceder a ella desde fuera de esta clase:

Ejemplo

```
public class Main {
 public static void main(String[] args) {
  Person myObj = new Person();
  myObj.name = "John";  // error
  System.out.println(myObj.name); // error
 }
}
```

Si la variable se declarara como public, esperaríamos el siguiente resultado:

```
John
```

Sin embargo, al intentar acceder a una variable private, obtenemos un error:

```
MyClass.java:4: error: name has private access in Person
  myObj.name = "John";
      ^
MyClass.java:5: error: name has private access in Person
  System.out.println(myObj.name);
                     ^
2 errors
```

En su lugar, utilizamos los métodos getName()y setName()para acceder y actualizar la variable:

Ejemplo

```
public class Main {
  public static void main(String[] args) {
    Person myObj = new Person();
    myObj.setName("John"); // Set the value of the name variable to "John"
    System.out.println(myObj.getName());
  }
}

// Outputs "John"
```

¿Por qué encapsulación?

- Mejor control de los atributos y métodos de clase
- Los atributos de clase se pueden hacer **de solo lectura** (si solo usa el método get) o **de solo escritura** (si solo usa el método set)
- Flexible: el programador puede cambiar una parte del código sin afectar a otras partes
- Mayor seguridad de los datos

Paquetes Java y API

Paquetes integrados

Sintaxis

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package
```

Importar una clase

Ejemplo

```
import java.util.Scanner;
```

Ejemplo

Uso de la clase Scanner para obtener la entrada del usuario:

```
import java.util.Scanner;

class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter username");

    String userName = myObj.nextLine();
    System.out.println("Username is: " + userName);
  }
}
```

Importar un paquete

Ejemplo

```
import java.util.*;
```

Paquetes definidos por el usuario

Para crear su propio paquete, debe comprender que Java utiliza un directorio del sistema de archivos para almacenarlos. Al igual que las carpetas en su computadora:

Ejemplo
└── root
  └── mypack
    └── MyPackageClass.java

Para crear un paquete, utilice la palabra clave package:

```
MyPackageClass.java
package mypack;
class MyPackageClass {
  public static void main(String[] args) {
    System.out.println("This is my package!");
  }
}
```

Guarde el archivo como MyPackageClass.java y compílelo:

```
C:\Users\Your Name>javac MyPackageClass.java
```

A continuación, compile el package:

```
C:\Users\Your Name>javac -d . MyPackageClass.java
```

Esto obliga al compilador a crear el paquete "mypack".

La palabra clave especifica el destino de dónde guardar el archivo de clase. Puede usar cualquier nombre de directorio, como c:/user (windows), o, si desea mantener el paquete dentro del mismo directorio, puede usar el signo de punto "", como en el ejemplo anterior.-d.

Nota: El nombre del paquete debe escribirse en minúsculas para evitar conflictos con los nombres de clase.

Cuando compilamos el paquete en el ejemplo anterior, se creó una nueva carpeta, llamada "mypack".

Para ejecutar el archivo MyPackageClass.java, escriba lo siguiente:

```
C:\Users\Your Name>java mypack.MyPackageClass
```

El resultado será:

```
This is my package!
```

Herencia (Subclase y Superclase)

Ejemplo

```
class Vehicle {
  protected String brand = "Ford";       // Vehicle attribute
  public void honk() {                   // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from
the Car class
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

¿Has notado el modificador protected en Vehículo?

Establecemos el atributo de marca en Vehículo a un modificador protected de acceso. Si se estableciera en private, la clase Car no podría acceder a él.

¿Por qué y cuándo usar "herencia"?
- Es útil para la reutilización de código: reutilizar atributos y métodos de una clase existente cuando se crea una nueva clase.

Tip: También eche un vistazo al siguiente capítulo, Polimorfismo, que utiliza métodos heredados para realizar diferentes tareas.

La palabra clave final

Si no desea que otras clases hereden de una clase, use la palabra clave final:

Si intenta acceder a una clase final, Java generará un error:

```
final class Vehicle {
  ...
}

class Car extends Vehicle {
  ...
}
```

El resultado será algo como esto:

```
Main.java:9: error: cannot inherit from final Vehicle
class Main extends Vehicle {
                   ^
1 error)
```

Polimorfismo de Java

Ejemplo

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

Recuerde en el capítulo Herencia que usamos la palabra clave para heredar de una clase.extends

Ahora podemos crear y objetos y llamar al método en ambos:Pig Dog animalSound()

Ejemplo

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
```

```
  }
}

class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

¿Por qué y cuándo usar "herencia" y "polimorfismo"?
- Es útil para la reutilización de código: reutilizar atributos y métodos de una clase existente cuando se crea una nueva clase.

Clases internas de Java

Ejemplo

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}

// Outputs 15 (5 + 10)
```

Clase Interior Privada

A diferencia de una clase "regular", una clase interna puede ser o . Si no desea que los objetos externos tengan acceso a la clase interna, declare la clase como :private protected private

Ejemplo

```
class OuterClass {
  int x = 10;

  private class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

Si intenta acceder a una clase interna privada desde una clase externa, se produce un error:

```
Main.java:13: error: OuterClass.InnerClass has private access in OuterClass
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
            ^
```

Clase interna estática

Ejemplo

```
class OuterClass {
  int x = 10;

  static class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
    System.out.println(myInner.y);
```

```
  }
}

// Outputs 5
```

Nota: al igual que los atributos y métodos, una clase interna no tiene acceso a los miembros de la clase externa.static static

Acceda a la clase exterior desde la clase interna

Ejemplo

```
class OuterClass {
  int x = 10;

  class InnerClass {
    public int myInnerMethod() {
      return x;
    }
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.myInnerMethod());
  }
}

// Outputs 10
```

Clases y métodos abstractos

Una clase abstracta puede tener métodos abstractos y regulares:

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

A partir del ejemplo anterior, no es posible crear un objeto de la clase Animal:

```
Animal myObj = new Animal(); // will generate an error
```

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

Recuerde en el capítulo Herencia que usamos la palabra clave para heredar de una clase.extends

Ejemplo

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}
```

```
class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

¿Por qué y cuándo usar clases y métodos abstractos?

Para lograr la seguridad, oculte ciertos detalles y solo muestre los detalles importantes de un objeto.