

**GUÍA DE LABORATORIO 04**
Paquetes, Herencia y Polimorfismo JAVA

Asignatura	Datos del alumno	Fecha y Firma
Algoritmos y solución de problemas	Apellidos:	
	Nombre:	

Instrucciones:

Desarrollar las actividades que indica el docente en base a la guía de trabajo que se presenta.

1. Objetivos:

- Aprender sobre Herencia.
- Aprender sobre Polimorfismo.

2. Equipos, Herramientas o Materiales

- Computador
- Software: JDK – VSCode - Extension Pack for Java.

3. Fundamento Teórico**3.1. Paquetes de Java****Paquetes y API de Java**

En Java, un paquete se utiliza para agrupar clases relacionadas. Piense en él como una carpeta en un directorio de archivos. Usamos paquetes para evitar conflictos de nombres y para escribir un código más fácil de mantener. Los paquetes se dividen en dos categorías:

- Paquetes integrados (paquetes de la API de Java)
- Paquetes definidos por el usuario (crea tus propios paquetes).

Paquetes integrados

La API de Java es una biblioteca de clases preescritas, de uso gratuito, incluidas en el entorno de desarrollo de Java.

La biblioteca contiene componentes para gestionar entradas, programación de bases de datos y mucho más. La lista completa se puede encontrar en el sitio web de Oracle: <https://docs.oracle.com/javase/8/docs/api/>.

La biblioteca se divide en paquetes y clases. Esto significa que puede importar una sola clase (junto con sus métodos y atributos) o un paquete completo que contenga todas las clases que pertenecen al paquete especificado.

Para utilizar una clase o un paquete de la biblioteca, debe utilizar la palabra clave import:

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



Importar una clase

Si encuentra una clase que desea utilizar, por ejemplo, la clase Scanner que se utiliza para obtener la entrada del usuario, escriba el siguiente código:

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

En el ejemplo anterior, java.util es un paquete, mientras que Scanner es una clase del paquete java.util.

Para utilizar la clase Scanner, cree un objeto de la clase y utilice cualquiera de los métodos disponibles que se encuentran en la documentación de la clase Scanner. En nuestro ejemplo, utilizaremos el método `nextLine()` que se utiliza para leer una línea completa:



```
MyClass.java 1 x
MyClass.java > ...
1  import java.util.Scanner;
2
3  class MyClass {
4      Run | Debug | Run main | Debug main
5      public static void main(String[] args) {
6          Scanner myObj = new Scanner(System.in);
7          System.out.println(x:"Enter username");
8
9          String userName = myObj.nextLine();
10         System.out.println("Username is: " + userName);
11     }
12 }
```

Importar un paquete

Hay muchos paquetes para elegir. En el ejemplo anterior, usamos la clase Scanner del paquete java.util. Este paquete también contiene funciones de fecha y hora, generador de números aleatorios y otras clases de utilidad.

Para importar un paquete completo, finalice la oración con un asterisco (*). El siguiente ejemplo importará TODAS las clases del paquete java.util:

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

Paquetes definidos por el usuario

Para crear tu propio paquete, debes comprender que Java utiliza un directorio del sistema de archivos para almacenarlos, al igual que las carpetas de tu computadora:

```
└─ root
  └─ mypack
    └─ MyPackageClass.java
```

Para crear un paquete, utilice la palabra clave `package`:



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



3.2. Herencia de java

Herencia de Java (subclase y superclase)

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

-  subclase (hijo): la clase que hereda de otra clase
-  superclase (padre): la clase que se hereda de

Para heredar de una clase, utilice la palabra clave `extends`.

En el siguiente ejemplo, la clase `Car` (subclase) hereda los atributos y métodos de la clase `Vehicle` (superclase):

```
Car.java - Language Support for Java(TM) by Red Hat > Car
1 class Vehicle {
2     protected String brand = "Ford";           // Vehicle attribute
3     public void honk() {                         // Vehicle method
4         System.out.println(x:"Tuut, tuut!");
5     }
6 }
7
8 class Car extends Vehicle {
9     private String modelName = "Mustang";       // Car attribute
10    Run | Debug | Run main | Debug main
11    public static void main(String[] args) {
12
13        // Create a myCar object
14        Car myCar = new Car();
15
16        // Call the honk() method (from the Vehicle class) on the myCar object
17        myCar.honk();
18
19        // Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class
20        System.out.println(myCar.brand + " " + myCar.modelName);
21    }
22 }
```

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

¿Por qué y cuándo utilizar "herencia"?

Es útil para la reutilización de código: reutilizar atributos y métodos de una clase existente cuando se crea una nueva clase.

3.3. Polimorfismo de Java

Polimorfismo significa "muchas formas" y ocurre cuando tenemos muchas clases que están relacionadas entre sí por herencia.

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, una superclase llamada `Animal` que tiene un método llamado `animalSound()`. Las subclases de `Animales` podrían ser `Cerdos`, `Gatos`, `Perros`, `Pájaros` - Y también tienen su propia implementación de un sonido animal (el cerdo gruñe, el gato maúlla, etc.):



```
Animal.java 4
Animal.java > ...
1 class Animal {
2     public void animalSound() {
3         System.out.println(x:"The animal makes a sound");
4     }
5 }
6
7 class Pig extends Animal {
8     public void animalSound() {
9         System.out.println(x:"The pig says: wee wee");
10    }
11 }
12
13 class Dog extends Animal {
14     public void animalSound() {
15         System.out.println(x:"The dog says: bow wow");
16    }
17 }
18
```

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

```
Animal.java 2 X
Animal.java > Language Support for Java(TM) by Red Hat > Main
1 class Animal {
2     public void animalSound() {
3         System.out.println(x:"The animal makes a sound");
4     }
5 }
6
7 class Pig extends Animal {
8     public void animalSound() {
9         System.out.println(x:"The pig says: wee wee");
10    }
11 }
12
13 class Dog extends Animal {
14     public void animalSound() {
15         System.out.println(x:"The dog says: bow wow");
16    }
17 }
18
19 class Main {
20     Run | Debug | Run main | Debug main
21     public static void main(String[] args) {
22         Animal myAnimal = new Animal(); // Create a Animal object
23         Animal myPig = new Pig(); // Create a Pig object
24         Animal myDog = new Dog(); // Create a Dog object
25         myAnimal.animalSound();
26         myPig.animalSound();
27         myDog.animalSound();
28     }
29 }
```

¿Por qué y cuándo utilizar “herencia” y “polimorfismo”?

Es útil para la reutilización de código: reutilizar atributos y métodos de una clase existente cuando se crea una nueva clase.

4. Ejercicios



Ejercicio parte 01:

Herencia y Métodos Sobrescritos:

Crea una clase base Empleado con atributos nombre y salario. Define un método calcularBono() que devuelva un bono del 10% del salario. Luego, crea una clase derivada Gerente que sobrescriba calcularBono() para que el bono sea del 15% del salario.



Ejercicio parte 02:

Herencia Multinivel:

Crea una jerarquía de clases para representar vehículos. La clase base será Vehiculo, con una clase derivada VehiculoMotorizado y dos clases derivadas finales Coche y Motocicleta. Agrega atributos y métodos relevantes a cada clase.

