

Conceptos fundamentales de Java

7-5: Polimorfismo

Actividades prácticas

**Vocabulario:**

**LUIS MARIO POSSO GARCIA**

Identifique el término correspondiente a cada una de las siguientes definiciones.

| **polimorfismo** | Un concepto en la programación orientada por objetos que les permite a las clases tener varias formas y comportarse como sus superclases. |
| --- | --- |
| **sobrescritura de método, override** | Implementar métodos en una subclase que tiene el mismo prototipo (los mismos parámetros, nombre de método, y tipo de retorno) que otro método en la superclase. |
| **final** | Una palabra clave en Java usada para limitar las subclases a partir de la extensión de una clase, sobrescribiendo métodos o cambiando datos. |
| **Final (o clase sellada)** | Una propiedad de una case estática que hace que la clase no pueda ser extendida o que sus datos sean cambiados. |
| **Sobrecarga** | Implementar un método con el mismo nombre que otro método en la misma clase que tiene diferentes parámetros o un tipo de retorno diferente. |
| **Enlace dinámico** | El proceso por el cual Java es capaz de determinar qué método invocar cuando los métodos han sido sobrescritos. |
| **Abstract** | Una palabra clave en Java que permite que las clases sean extendidas, pero las clases no pueden ser instanciadas (construidas)y cuando se aplican los métodos, se indica que los métodos deberían ser implementados en todas las subclases de la clase. |

**Inténtelo/resuélvalo:**

¿Cuál sería el resultado del siguiente código?

class A

{

void callthis() {

System.out.println(“Inside Class A's Method!”);

}

}

class B extends A

{

void callthis() {

System.out.println(“Inside Class B's Method!”);

}

}

class C extends A

{

void callthis() {

System.out.println(“Inside Class C's Method!”);

}

}

class DynamicDispatch {

public static void main(String args[]) {

A a = new A();

B b = new B();

C c = new C();

A ref;

ref = b;

ref.callthis();

ref = c;

ref.callthis();

ref = a;

ref.callthis();

}

}

R/ El resultado del código sería:

Inside Class B's Method!

Inside Class C's Method!

Inside Class A's Method!

**2. ¿Cuál es la diferencia entre una clase abstract y una interfaz? ¿Cuándo es apropiado el uso de una clase abstract o una interfaz?**

R/ La diferencia entre una clase abstracta y una interfaz es que una clase abstracta puede tener implementaciones de métodos, mientras que una interfaz sólo puede tener declaraciones de métodos y no puede contener implementación.

En cuanto a cuándo es apropiado utilizar una clase abstracta o una interfaz, depende de la situación. Si se desea proporcionar una implementación predeterminada o compartida entre las clases hijas, se debe usar una clase abstracta. Si se desea definir un conjunto de métodos que deben ser implementados por cualquier clase que implemente la interfaz, se debe usar una interfaz.

3. Dada la siguiente información, determine si resultará: “siempre compila”, “a veces compila”, “no compila”. public interface A

public class B implements A

public abstract class C

public class D extends C

public class E extends B

Cada clase ha sido inicializada, pero no queda claro para qué han sido inicializadas:

A a = new...

B b = new...

C c = new...

D d = new...

E e = new...

Se incluyen los siguientes métodos:

interfaz A especifica method void methodA()

clase C tiene el abstract method void methodC()

| **Código:** | **¿Siempre compila, a veces compila, no compila?** |
| --- | --- |
| a = new B(); | **siempre compila** |
| d = new C(); | **no compila** |
| b.methodA(); | **a veces compila** |
| e.methodA(); | **siempre compila** |
| c = new C(); | **a veces compila** |
| (D)c.methodC(); | **a veces compila** |