

Conceptos fundamentales de Java

7-5: Polimorfismo

Actividades prácticas

**Objetivos de la lección:**

* Aplicar referencias superclase a subclase objetos
* Redactar un código para sobrescribir métodos
* Usar el enlace dinámico para soportar el polimorfismo
* Crear métodos y clases abstract
* Reconocer el proceso de sobrescritura de método correcto
* Usar el modificador final
* Explicar el prototipo y la importancia de la clase Objeto
* Redactar un código para un applet que muestra dos triángulos de diferentes colores • Describir las referencias a objetos

**Vocabulario:**

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. |
| Sobreescritura de metodos | 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 | Una propiedad de una case estática que hace que la clase no pueda ser extendida o que sus datos sean cambiados. |
| Sobrecarga de metodos | 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 dinamico | 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. |

Copyright © 2020 Oracle y/o sus filiales. Todos los derechos reservados. Oracle y Java son marcas comerciales registradas de Oracle y sus filiales. Todos los demás nombres pueden ser marcas comerciales de sus respectivos propietarios.

**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[]) {

1. a = new A();
2. b = new B();
3. c = new C();

A ref;

ref = b;

ref.callthis();

ref = c;

ref.callthis();

ref = a;

ref.callthis();

}

}

**R// El retorno seria:**

**Inside Class B's Method!**

**Inside Class C's Method!**

**Inside Class A's Method!**

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

R// En Java, una clase abstract es una clase que no puede ser instanciada y que puede contener tanto métodos con implementación como métodos sin implementación (métodos abstractos), mientras que una interfaz es una colección de métodos abstractos que no tienen implementación.

La principal diferencia entre una clase abstract y una interfaz es que una clase abstract puede proporcionar una implementación predeterminada para algunos de sus métodos, mientras que en una interfaz todos los métodos son abstractos y no tienen implementación. Además, una clase solo puede extender una clase abstract, mientras que una clase puede implementar múltiples interfaces.

El uso de una clase abstract se justifica cuando se desea proporcionar una implementación predeterminada para algunos de los métodos de una clase base y permitir que las subclases extiendan y sobrescriban dichos métodos para personalizar su comportamiento. Las clases abstractas también pueden contener campos y métodos concretos que son compartidos por todas las subclases.

Por otro lado, el uso de una interfaz se justifica cuando se desea proporcionar un conjunto común de métodos para diferentes clases que no necesariamente comparten una relación de herencia. Las interfaces permiten definir contratos de comportamiento que deben ser implementados por las clases que las implementan, lo que facilita la creación de código modular y extensible. Las interfaces también se utilizan para definir tipos de datos abstractos.

Copyright © 2020, Oracle y/o sus filiales. Todos los derechos reservados. Oracle y Java son marcas comerciales registradas de Oracle y/o sus filiales. Todos los demás nombres pueden ser marcas comerciales de sus respectivos propietarios

2

1. 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:

* 1. a = new...
  2. b = new... C c = new...
  3. d = new...
  4. 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(); | A veces compila |
| d = new C(); | No compila |
| b.methodA(); | No compila |
| e.methodA(); | Siempre compila |
| c = new C(); | Siempre compila |
| (D)c.methodC(); | No compila |

1. Sobrescribir el método toString() para la clase más abajo para generar los resultados, haciendo coincidir con el resultado dado. El método toString() debería imprimir todos los valores desde 1 al número especificado en num y luego imprimir el valor final usando el método getFactorial provisto.

Suponga que el num int variable es un valor global público:

“Factorial: 10! = 1 \* 2 \* 3 \* 4 \* 5 \* 6 \* 7 \* 8 \* 9 \* 10 = 3628800”

int getFactorial(){

int factorial;

for(i = num; num > 0; i--){

factorial \*= num;

}

return factorial;

}

public String toString() {

}

**Respuesta:**

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append("Factorial: ");

sb.append(num);

sb.append("! = ");

int factorial = 1;

for (int i = 1; i <= num; i++) {

sb.append(i);

factorial \*= i;

if (i != num) {

sb.append(" \* ");

}

}

sb.append(" = ");

sb.append(factorial);

return sb.toString();

}

Copyright © 2020, Oracle y/o sus filiales. Todos los derechos reservados. Oracle y Java son marcas comerciales registradas de Oracle y/o sus filiales. Todos los demás nombres pueden ser marcas comerciales de sus respectivos propietarios

3