

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.

|  |  |
| --- | --- |
| Porlimorfismo | Un concepto en la programación orientada por objetos que les permite a las clases tener varias formas y comportarse como sus superclases. |
| Sobreescritura | 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 | 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 código proporcionado tiene tres clases, A, B y C. Las clases B y C extienden la clase A, lo que significa que heredan sus métodos y atributos. Cada clase tiene un método llamado "callthis" que imprime un mensaje diferente. Luego, en la clase DynamicDispatch, se crean tres objetos: uno de la clase A, otro de la clase B y otro de la clase C. A continuación, se crea una referencia de tipo A llamada "ref". Se asigna el objeto de la clase B a la referencia "ref" y se llama al método "callthis". Como "ref" es de tipo A, pero se asignó un objeto de la clase B, Java utiliza el enlace dinámico para determinar que se debe llamar al método "callthis" de la clase B. ¡Entonces, se imprime “Inside Class B’s Method!". Luego, se asigna el objeto de la clase C a la referencia "ref" y se llama al método "callthis". De manera similar, Java utiliza el enlace dinámico para determinar que se debe llamar al método "callthis" de la clase C. ¡Entonces, se imprime “Inside Class C’s Method!". Finalmente, se asigna el objeto de la clase A a la referencia "ref" y se llama al método "callthis". Dado que "ref" es de tipo A y se asignó un objeto de la clase A, se llama al método "callthis" de la clase A. ¡Entonces, se imprime “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// Una clase abstracta es una clase que no puede ser instanciada, sino que se utiliza como una plantilla para crear subclases que heredan sus propiedades y métodos. Al menos uno de los métodos de una clase abstracta es declarado abstracto, lo que significa que no tiene una implementación y debe ser definido por sus subclases. Una interfaz, por otro lado, es una colección de métodos abstractos (es decir, sin implementación) y constantes que se definen en una clase separada. Las clases que implementan una interfaz deben proporcionar implementaciones concretas de todos los métodos de la interfaz.

Y teniendo en cuenta la información anterior osea tengo unos puntos :

* Una clase abstracta es apropiada cuando se quiere proporcionar una estructura básica y común a varias subclases, y se desea compartir código común entre ellas.
* Una interfaz es apropiada cuando se quiere definir un conjunto de métodos que una clase debe implementar, pero no se quiere especificar una implementación concreta. Las interfaces se utilizan a menudo para definir contratos para las clases que las implementan.
* Las clases abstractas también pueden ser utilizadas en situaciones en las que una subclase solo necesita proporcionar una implementación para algunos de los métodos de la clase padre, mientras que los demás métodos ya tienen una implementación predeterminada en la clase abstracta.
* Las interfaces, por otro lado, pueden ser utilizadas para definir comportamientos comunes a diferentes clases, incluso si no están relacionadas en una jerarquía de clases.

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(); | Siempre compila |
| d = new C(); | No compila |
| b.methodA(); | No compila |
| e.methodA(); | A veces compila |
| c = new C(); | Siempre compila |
| (D)c.methodC(); | A veces 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() {

}

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