
ORACLE
Academy

Java Fundamentals

7-5:
Polimorfismo

ORACLE
Academy

Objetivos

- Esta lección abarca los siguientes temas:
  - Aplicar referencias superclase a subclase objetos
  - Escritura de un código para invalidar los métodos
  - Uso del despacho del método dinámico para brindar soporte al polimorfismo
  - Creación de métodos y clases abstractos
  - Reconocimiento de una anulación de método correcta

ORACLE Academy

Overview

- Esta lección abarca los siguientes temas:
  - Uso del modificador final
  - Explicación del objetivo y la importancia de la clase del objeto
  - Escritura del código para un Applet que muestre dos triángulos de diferentes colores
  - Describir las referencias a objetos

Revisión de la herencia

- Cuando una clase hereda de otra, la subclase “is a” (es un) tipo de la superclase
- Los objetos de una subclase se pueden referenciar usando una referencia o tipo de superclase

Más información

- Visite las páginas del tutorial de Oracle para obtener más información:
- Herencia:
  - http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
- Polimorfismo:
  - http://docs.oracle.com/javase/tutorial/java/IandI/Polimorfismo.html

Ejemplo de herencia

- Si las clases se crean para una clase Bicicleta o una clase RoadBike (bicicleta de carretera) como extensión de Bicicleta, la referencia del tipo Bicicleta puede referirse a un objeto RoadBike (consultar más adelante)
  - Dado que RoadBike “is a” (es un) tipo de Bicicleta, está perfectamente permitido almacenar un objeto RoadBike como una referencia de Bicicleta
  - El tipo de una variable (o referencia) no determina el tipo real del objeto se refiere

Ejemplo de herencia

- Por lo tanto, una referencia o variable de Bicicleta puede contener o no un objeto del tipo de superclase Bicicleta, dado que contiene cualquier subclase de Bicicleta

```
Bicycle bike = new RoadBike();
```

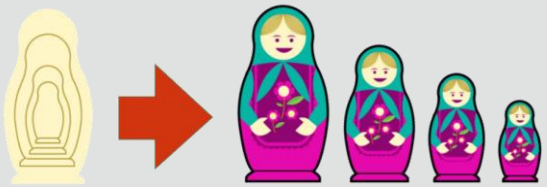
ORACLE Academy

Polimorfismo

- Cuando una variable o referencia se puede referir a diferentes tipos de objetos se denomina polimorfismo
- Polimorfismo es un término que significa “muchas formas”
- En programación, el polimorfismo permite variables para hacer referencia a varios tipos de objetos diferentes y significa que pueden tener múltiples formas
- Por ejemplo, porque RoadBike “is a” Bicicleta, hay dos referencias posibles que definen el tipo de objeto que es (Bicicleta o RoadBike (bicicleta de carretera))

Polimorfismo y muñecas rusas

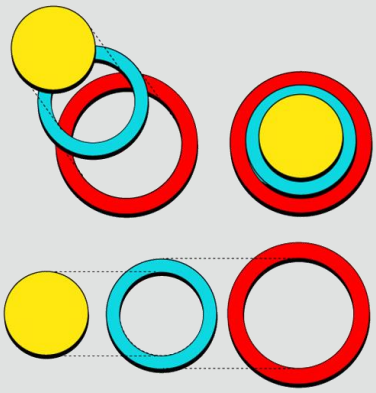
- El polimorfismo se puede visualizar como un conjunto de muñecas rusas: el conjunto de muñecas comparte un tipo y una apariencia, pero son todas únicas de alguna manera
- Cada muñeca tiene la misma forma, con un tamaño determinado por la muñeca dentro de la cual debe encajar
- Cada muñeca más pequeña se guarda dentro de la muñeca más grande siguiente
- Desde afuera no puede ver a las muñecas más pequeñas, pero puede abrir cada muñeca para encontrar una muñeca más pequeña adentro



ORACLE Academy

Súper clases y sub clases

- De manera similar, las subclases se pueden “adaptar” al tipo de referencia de una superclase
- La variable de una superclase puede sostener o guardar el objeto de una subclase, mientras mira y actúa como la superclase



ORACLE Academy

Este concepto nos permite comprender la limitación de las arreglas que solo pueden albergar un tipo de datos. Una arreglo de Bicicletas podría albergar los objetos Bicicleta y Bicicleta de montaña, puesto que la bicicleta de montaña "es una" bicicleta. Este concepto llevado al extremo nos permite decir que una arreglo de objetos en realidad podría albergar cualquier tipo de objeto, puesto que el objeto es la última superclase.

Variables de superclases

- Si “abre” la variable de una superclase o invoca uno de sus métodos, encontrará que efectivamente tiene el objeto de una subclase guardado adentro
- Por ejemplo
  - con las muñecas rusas, no puede ver la muñeca más pequeña hasta que abra a la muñeca más grande
  - Su tipo puede ser ambiguo
  - Cuando se compila el código Java, Java no verifica qué tipo (supertipo o subtipo) de objeto se encuentra dentro de una variable
  - Cuando se ejecuta el código Java, Java “abrirá” para ver qué tipo de objeto se encuentra dentro de la referencia e invocará a los métodos que sean de ese tipo

Dudas al referenciar objetos

- Se puede dudar al hacer referencia a objetos diferentes en el momento de compilarlos
- Por ejemplo:
  - Usted escribe un programa que calcula las diferentes longitudes de los tubos del marco de una bicicleta, teniendo en cuenta las medidas del ciclista y el tipo de bicicleta deseada (RoadBike o MountainBike)
  - Usted desea obtener una lista para realizar el seguimiento de la cantidad de objetos bicicleta que ha construido
  - Desea solo una lista, no dos listas por separado para cada tipo de bicicleta
  - ¿Cómo prepara esa lista? ¿Una arreglo, tal vez?
  - ¿Cuál es el problema si usa una arreglo para preparar la lista?

¿Por qué no usar una arreglo de objetos de clase?

- Las arreglas son un conjunto de elementos del mismo tipo, como un conjunto de enteros, dobles o bicicletas
- Mientras que es posible tener una arreglo de objetos, deben ser del mismo tipo
- Es correcto para las clases que no han sido extendidas

¿Por qué no usar una arreglo de objetos de clase?

- El polimorfismo resuelve este problema
- Dado que RoadBike y MountainBike son tipos de objetos Bicicleta, use una arreglo de referencias de Bicicleta para almacenar la lista de bicicletas que ha desarrollado
- Cualquier tipo de bicicleta se puede agregar a esta arreglo

```
Bicycle[] bikes = new Bicycle[size];
```

Referencias de objetos

- La clase objeto es la superclase más alta en Java, dado que no se extiende a otras clases
- Como resultado de ello, cualquier clase se puede almacenar en una referencia de objeto

```
Object[] objects = new Bicycle[size];
```

- En el ejemplo de esta arreglo, también es válido almacenar nuestras bicicletas en una arreglo de referencias de objetos
- Sin embargo, esto hace que el tipo de nuestra arreglo sea más ambiguo, por lo que se deberá evitar hacerlo, a menos que exista una razón para ello

Invalidación de métodos de objeto

- Dado que objeto es la superclase de todas las clases, sabemos que todas las clases heredan métodos del objeto
- Dos de estos métodos son muy útiles, como el método equals() y el método toString()
- El método equals() nos permite verificar si dos referencias se refieren al mismo objeto
- El método toString() devuelve un String que representa el objeto
- El String entrega información básica sobre el objeto como su clase, nombre y hashcode único

Invalidación o redefinición de métodos

- Aunque los métodos equals() y toString() son útiles, no incluyen la funcionalidad para un uso más específico
- Para la clase Bicicleta, podemos desear que se genere un String que contenga el número de modelo, el color, el tipo de marco y el precio
- El uso del método del objeto no devolverá esta información
- El método toString() en la clase Objeto devuelve una representación del String de la ubicación del objeto en la memoria
- En lugar de crear un método con otro nombre, podemos invalidar el método toString() y redefinirlo para que se ajuste a nuestras necesidades

Invalidación de métodos

- La invalidación de métodos es la manera de redefinir los métodos con el mismo tipo de devolución y parámetros, al agregar o invalidar la lógica existente en el método de una subclase
- La invalidación es diferente de la sobrecarga de un método
- La sobrecarga de un método significa que el programador conserva el mismo nombre (por ej.: toString()), pero cambia los parámetros de ingreso (firma del método)
- La invalidación esencialmente oculta el método del padre con la misma firma y no será invocado en el objeto de una subclase a menos que la subclase use la palabra clave super

Métodos de invalidación

- La invalidación no cambia los parámetros
- Solo cambia la lógica dentro del método definido en la superclase

Técnicamente, un método de sustitución en la subclase podría, con ciertas limitaciones, tener un tipo de retorno diferente siempre que los tipos de parámetros y el número de parámetros sean los mismos. No se recomienda esta práctica.

Tutoriales de Java sobre métodos de invalidación

- Visite los tutoriales de Java de Oracle para obtener más información sobre los métodos de invalidación:
  - http://docs.oracle.com/javase/tutorial/java/IandI/override.html

Invalidación de toString()

- Podemos invalidar toString() para devolver un String que entregue información sobre el objeto en lugar de la ubicación del objeto en la memoria
  - En primer lugar, comience por el prototipo:

```
public String toString()
```

  - No existe motivo para cambiar el tipo de devolución o los parámetros, por eso, invalidamos toString()
  - Considerando nuestros datos privados (número de modelo, color, tipo de marco y precio), podemos devolver el siguiente String:

```
return "Model: " + modelNum + " Color: " + color +
       " Frame Type: " + frameType + " Price: " + price;
```

Invalidación de toString()

- El resultado de nuestro método invalidado toString():

```
public String toString(){
   return "Model      : " + modelNum +
          "Color      : " + color +
          "Frame Type : " + frameType +
          "Price      : " + price;
}//end method toString
```

- Es muy común y muy útil cuando se crean clases de Java, invalidar el método toString() para probar sus métodos y datos

Comprensión del modelo de objeto

- El polimorfismo, como la herencia, es esencial para el modelo del objeto y la programación orientada a objetos
- El polimorfismo brinda versatilidad para trabajar con objetos y referencias, mientras se mantienen los objetos como discretos o distintos
- Como esencia de esta filosofía, el modelo del objeto convierte a los programas en un conjunto de objetos, con comparación con un conjunto de tareas, encapsulando los datos y creando partes más pequeñas de un programa, en lugar de un fragmento más grande del código

Objetivos del modelo de objeto

- El modelo de objeto tiene varios objetivos:
  - Abstracción de datos
  - Protección de información y limitación de la capacidad de otras clases para cambiar o dañar datos
  - Ocultar la implementación
  - Brindar un código modular que pueda ser utilizado nuevamente por otros programas o clases

Polimorfismo y métodos

- ¿De qué manera se ven afectados los métodos en la subclase por el polimorfismo?
- Recuerde que las subclases pueden heredar métodos de sus superclases
- Si la variable Bicicleta puede mantener al tipo de objeto de las subclases, ¿cómo sabe Java qué métodos debe invocar cuando se invoca a un método invalidado?

Polimorfismo and Methods

- Los métodos invocados en una referencia (Bicicleta) siempre se refieren a los métodos en el tipo de objeto (RoadBike)

```
Bicycle bike = new RoadBike();
```

- Imagine que la clase Bicicleta contiene un método setColor(Color color) para establecer el color de la bicicleta que el ciclista desea
- RoadBike hereda este método
- ¿Qué ocurre cuando hacemos lo siguiente?

```
bike.setColor(new Color(0, 26, 150));
```


La primera línea de código no es demasiado práctica pero ilustra el concepto. La aplicación práctica sería la arreglo de objetos Bicycle, algunos de los cuales pueden ser RoadBikes.

Distribución dinámica de un método

- Java puede determinar qué método invocar en base al tipo de objeto al que se hace referencia en el momento de invocar el método
- La distribución dinámica de un método, conocida como enlace dinámico, le permite a Java determinar de manera correcta y automática qué método invocar en base al tipo de referencia y tipo de objeto

Esto se realiza en el tiempo de ejecución.

Clases abstractas

- ¿Es realmente necesario definir una clase `Bicicleta, si solo vamos a crear objetos de sus subclases:
  - roadBikes
  - mountainBikes?
- Las clases abstractas son una alternativa que se ocupa de esta problema
- Una clase abstracta es aquella para la que no se puede crear una instancia:
  - Esto significa que usted no puede crear objetos de este tipo
  - Y que es posible crear variables o referencias de este tipo

Puede resultar difícil para los alumnos que estudian por primera vez la programación orientada al objeto
(u OPP) entender el valor de las clases abstractas, las clases finales y otros temas relacionados. Se dirán: "Pero no crearé ninguna bicicleta. ¿Por qué hacer que la clase sea abstracta?" No existe una respuesta sencilla a esa pregunta, en realidad se reduce a cumplir el diseño global que construyeron los arquitectos de Java. Al hacer que la Bicicleta sea abstracta, otro programador no tardará en crear objetos Bicicleta puros e "interrumpirá" el intento de los diseñadores del programa original.

Clases abstractas

- Si declaramos que la clase Bicicleta es abstracta, todavía podemos usar la siguiente sintaxis, pero no podemos efectivamente crear un objeto Bicicleta
- Esto significa que todas las referencias al tipo Bicicleta se referirán a los objetos de la subclase MountainBike o RoadBike

```
Bicycle bike2 = new mountainBike();
```

```
Bicycle bike  = new RoadBike();
```

Clases abstractas

- Las clases abstractas pueden contener métodos totalmente implementados que se “pasan” a cualquier clase que los extienda
- Puede hacer que una clase sea abstracta mediante la palabra clave abstract

```
public abstract class Bicycle
```

Clases abstractas

- Las clases abstractas también pueden declarar por lo menos un método abstracto (que no contiene ninguna implementación)
- Esto significa que las subclases deben usar prototipo del método (esquema) y deben implementar estos métodos
- Los métodos abstractos se declaran con la palabra clave abstract

```
abstract public void setPrice();
```

Declare como público abstracto No use corchetes "{}"

Métodos abstractos

- Métodos abstractos:
  - No pueden tener el cuerpo de un método
  - Deben ser declarados en una clase abstracta
  - Deben estar invalidados en una subclase
- Esto obliga a los programadores a implementar y redefinir los métodos
- En general, las clases abstractas contienen métodos abstractos, métodos parcialmente implementados o métodos totalmente implementados

Métodos implementados parcialmente

- Recuerde que las subclases pueden invocar al constructor y a los métodos de su superclase mediante la palabra clave super
- Con las clases abstractas, las subclases también pueden usar super para utilizar el método de su superclase
- En general, esto se hace invalidando primero el método de la superclase, luego invocando el método super o el método invalidado, y luego agregando el código
- Por ejemplo:
  - invalidemos el método equals() de la clase abstracta Bicicleta, que ha sido implementada parcialmente
  - Esto significa que el método equals() en Bicicleta no es abstracto

Métodos implementados parcialmente

- Aquí se comparan dos objetos Bicicleta en base al precio y el número de modelo
  - Observe que este método invalida al método equals() de objeto, porque tiene los mismos parámetros y tipo de devolución

```
public boolean equals(Object obj) {
   if(this.price == obj.price &&
      this.modelNum == obj.modelNum) {
      return true;
   }
   else {
      return false;
   }//end if
}//end method equals
```

Métodos implementados parcialmente

- Podemos invalidar el método en nuestra subclase MountainBike y verificar si existe equivalencia en otros atributos

```
public boolean equals(Object obj) {
    if(super.equals(obj)) {
       if(this.suspension == obj.suspension)
       return true;
    }//end if
    return false;
}//end method equals
```

Subclasificación de clases abstractas

- Al heredar de una clase abstracta, debe optar por uno de los siguientes puntos:
  - Declarar la clase hijo como abstracta
  - Invalidar todos los métodos abstractos heredados de la clase padre
  - Si no lo hace se generará un error de tiempo de compilación

Uso de Final

- Aunque se aconseja contar con esta opción, en algunos casos, es posible que desee invalidar algunos métodos o extender su clase
- Java le brinda una herramienta para evitar que los programadores invaliden métodos o creen subclases:
  - la palabra clave final

Uso de Final

- Un buen ejemplo es la clase de String
- Se declara:

```
public final class String {}
```

- Los programadores harán referencia a las clases como esta como inmutables, es decir, ninguna puede extender el String y modificar o invalidar sus métodos

Uso de Final

- El modificador Final se puede aplicar a las variables
- Las variables Final no pueden cambiar sus valores después de ser inicializadas

Se trata de constantes. Además, los objetos pueden ser finales, pero probablemente no de la forma que se imaginan de forma intuitiva los alumnos. Consulte la diapositiva 43.

Uso de Final

- Las variables Final pueden ser:
  - Campos de clase
    - Los campos Final con expresiones constantes de tiempo de compilación son variables constantes
    - Static se puede combinar con final para crear una variable siempre disponible e inalterable
  - Parámetros de métodos
  - Variables locales

Cuando se trata con valores como PI y otras constantes matemáticas y científicas, etc., es recomendable asignarlos como variables estáticas finales.

Uso de Final

- Las referencias Final siempre se deben referir al mismo objeto
- El objeto al cual la variable hace referencia no se puede cambiar
- El contenido de ese objeto se puede modificar
- Visite los tutoriales de Java de Oracle para obtener más información sobre el uso de Final:
  - http://docs.oracle.com/javase/tutorial/java/IandI/final.html

Código Applet de triángulo

- El siguiente código muestra los pasos incluidos al escribir un Applet con dos triángulos de diferentes colores

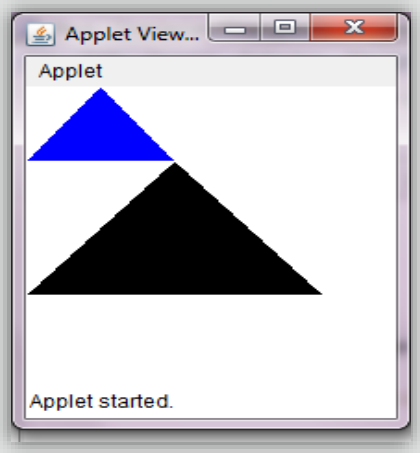
```
public class TrianglesApplet extends Applet{
   public void paint(Graphics g){
      int[] xPoints = {0, 40, 80};
      int[] yPoints = {50, 0, 50};
      g.setColor(Color.blue);
      g.fillPolygon(xPoints, yPoints, 3);
      int[] x2Points = {80, 160, 0};
      int[] y2Points = {50, 140, 140};
      g.setColor(Color.black);
      g.fillPolygon(x2Points, y2Points, 3);
   }//end method paint
}//end class TrianglesApplet
```

Explicación del código Applet de Triángulo

- Paso 1:
  - extienda la clase de Applet para que herede todos los métodos, incluso pintar
- Paso 2:
  - invalide el método pintar para incluir los triángulos
- Paso 3:
  - dibuje el triángulo usando el método heredado fillPolygon
- Paso 4:
  - dibuje el segundo triángulo usando el método heredado fillPolygon
- Paso 5:
  - ejecute y compile su código

Imagen del Applet de Triángulo

- El código del Applet de Triángulo muestra la siguiente imagen:



Terminología

- Los términos clave usados en esta lección son los siguientes:
  - Abstracto
  - Distribución dinámica de un método
  - Final
  - Inmutable
  - Métodos de sobrecarga
  - Invalidación de métodos
  - Polimorfismo

Resumen

- En esta lección, habrá aprendido a:
  - Aplicar referencias superclase a subclase objetos
  - Escritura de un código para invalidar los métodos
  - Uso del despacho del método dinámico para brindar soporte al polimorfismo
  - Creación de métodos y clases abstractos
  - Reconocimiento de una anulación de método correcta
  - Uso del modificador final
  - Explicación del objetivo y la importancia de la clase del objeto
  - Escritura del código para un Applet que muestre dos triángulos de diferentes colores
  - Describir las referencias a objetos


ORACLE
Academy