
ORACLE
Academy

Java Fundamentals

7-4:

Herencia

ORACLE
Academy

Objetivos

- Esta lección abarca los siguientes temas:
  - Demostración y explicación de los diagramas de clase UML (Unified Modeling Language)
  - Uso de la palabra clave extends para heredar una clase
  - Comparación y contraste de superclases y subclases
  - Descripción de la manera en que afecta la herencia al acceso a los miembros
  - Uso de super para invocar al constructor de una superclase
  - Uso de super para acceder a los miembros de una superclase
  - Creación de una jerarquía de clases de niveles múltiples

Descripción general

- Esta lección abarca los siguientes temas:
  - Reconocimiento del momento en que se invocan los constructores en la jerarquía de una clase
  - Demostración de la comprensión de la herencia mediante el uso de Applets
  - Reconocimiento de los cambios de parámetros correctos en una Applet existente

¿Qué es herencia?

- La herencia es una herramienta simple pero poderosa de lenguajes orientados a los objetos que permite a las clases heredar métodos y campos de otras clases
- Se entiende por heredar, recibir u obtener algo de su predecesor o padre
- En Java, el concepto de herencia es similar a la genética
  - Los genes y los rasgos genéticos se traspasan del padre al hijo
  - Como resultado de ello, los hijos generalmente se parecen y actúan como sus padres

Más información sobre herencia

- Para obtener más información sobre herencia, visite:
  - http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Superclase frente a Subclase

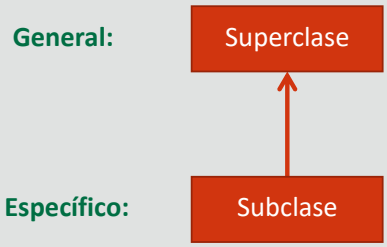
- Las clases se pueden derivar de, o evolucionar desde, las clases de padres, o sea, que contienen los mismos métodos y campos que sus padres, pero se pueden considerar una forma más especializada de las clases de sus padres
- La diferencia entre una subclase y una superclase es la siguiente:

Superclase	Subclase
Es la clase más general desde la que otras clases derivan sus métodos y datos	La clase más específica que se deriva o hereda de otra clase (la superclase)

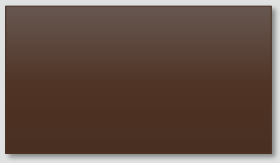
La superclase se denomina a veces clase base.

Superclase frente a Subclase

- Las superclases contienen métodos y campos que se traspasan a todas sus subclases
- Subclases:
  - Heredan métodos y campos de sus superclases
  - Pueden definir métodos o campos adicionales que la superclase no tiene



Ejemplo de herencia



- Crear una clase Forma con una variable, un color y un método que devuelva el color
  - Crear la clase Rectángulo que herede la variable y el método de la Forma y pueda tener sus propios métodos y variables

```
Shape (superclass)
public String color
public String getColor()
```

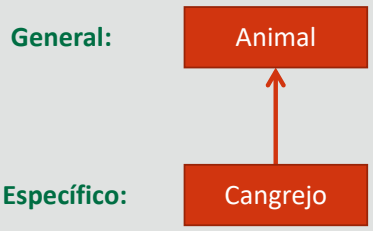
```
Rectangle (subclass)
public String color
public String getColor()

//Rectangle-only data
public int length
public int width
public int getLength()
public int getWidth()
```

ORACLE Academy

La API de Java tiene una interfaz Shape y una clase Rectangle. Los ejemplos de las diapositivas no hacen referencia a la API.

Ejemplo de Superclase frente a Subclase

- Considere las clases Animal y Cangrejo de Greenfoot
- Animal es un término más general que Cangrejo y se puede aplicar a más criaturas que Cangrejo
- Un Cangrejo es un tipo de Animal y ese Cangrejo se aplica a un tipo específico de Animal
- Por lo tanto, el Cangrejo es la subclase y Animal es la superclase



ORACLE Academy

La subclase debe cumplir la prueba "Es un/una" con la superclase. Un cangrejo "es un" animal. Este concepto se analiza en la diapositiva 26.

Método move de la clase Cangrejo

- ¿De dónde proviene el método move() en Cangrejo?
- No existe un código visible que muestre la lógica para el método move() en la clase Cangrejo

```
public class Crab extends Animal
{
   public void act()
   {
      move(1);
   }//end method act
}//end class Crab
```

Método move heredado

- Aunque el código no está escrito en la clase Cangrejo, sabemos que un objeto Cangrejo puede invocar al método move()
- Por lo tanto, el código debe ser heredado desde la superclase Animal, de la siguiente manera:

```
public class Animal
{
   public void move(int d)
   {
      //Logic for move()
   }//end method move
}//end class Animal
```

Palabra clave extends

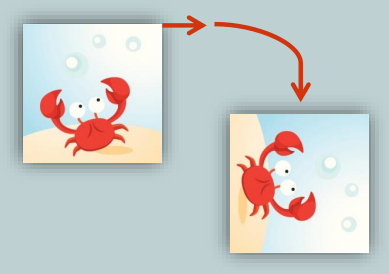
- En Java, puede elegir qué clases desea heredar al usar la palabra clave extends (extiende)
- La palabra clave extends le permite designar la superclase que posee los métodos que desea heredar o de quiénes son los métodos y datos que desea extender
- Por ejemplo, para heredar métodos de la clase Forma, use extends al crear la clase Rectángulo

```
public class Rectangle extends Shape
{
   //code
}//end class Rectangle
```

Ejemplo de la palabra clave extiende

- Deseamos que la clase Cangrejo extienda los métodos y datos en Animal y herede métodos como move(), turn(), etc
  - Al extender la clase Animal, puede invocar los métodos move() y turn() aunque no aparezcan en el código de la clase Cangrejo

```
public class Crab extends Animal
{
   public void act()
   {
      move(1);
      turn(90);
   }//end method act
}//end class Crab
```



La regla de la herencia única

- Se entiende por herencia única, que no podrá declarar o extender más de una superclase por clase
- El siguiente código no se compilará:

```
public class Crab extends Animal, Crustacean, ...
```

Extender más de una clase

- ¿Por qué no podemos extender más de una clase?
  - Dado que las superclases traspasan sus métodos y datos a todas sus subclases y las subclases de sus subclases, no es realmente necesario extender más de una clase

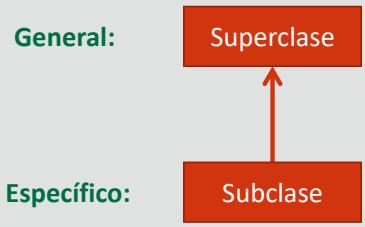
Las interfaces a veces se utilizan para implantar un comportamiento que se forma por la falta de herencia múltiple. Las interfaces se analizan en la Programación Java.

Más información sobre herencia

- La herencia es un camino en una sola dirección
  - Las subclases heredan de las superclases, pero las superclases no pueden acceder o heredar métodos y datos de sus subclases
  - Es por eso que los padres no heredan rasgos genéticos, como el color del cabello o de los ojos, de sus hijos

Objeto: la superclase más alta

- Toda superclase implícitamente extiende el objeto de clase
- Objeto:
  - Se considera el componente más alto y más general de cualquier jerarquía
  - Es la única clase que no tiene una superclase
  - Contiene métodos muy generales que todas las clases heredan



ORACLE Academy

Los métodos de los objetos suelen sustituirse, si se utilizan.

Ejemplo 1 de Objeto

- El objeto contiene métodos que se pueden usar en todas las clases, como toString() o equals()
- Por ejemplo, después de crear una clase y construir una instancia de esta clase, ¿puede invocar al método toString() en su objeto?

```
A_Class class = new A_Class();
class.toString();
```

- Sí
  - Aunque no haya escrito el método toString(), se le permite todavía invocar a este método porque fue heredado del Objeto

Ejemplo 2 de Objeto

- ¿Se permite class.toString() si A_Class extiende explícitamente a Another_Class, una superclase?
- Sí
- Siempre se permite, dado que la superclase A_Class se extiende al Objeto

```
A_Class class = new A_Class();
class.toString();
```

¿Por qué usar herencia?

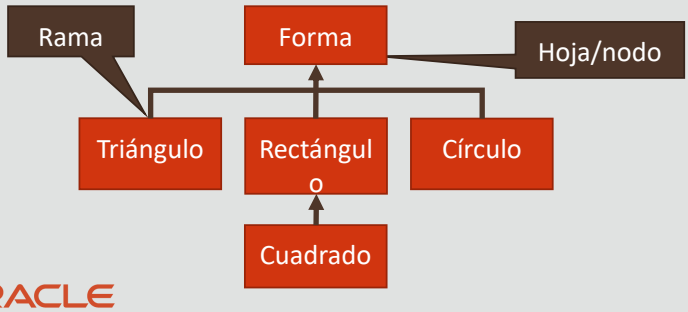
- El beneficio principal para la herencia es la reutilización del código
- Al heredar los métodos de una superclase, le otorga a su clase acceso al código y a los datos de la superclase
- No debe escribir el código dos veces, con lo cual usted ahorra tiempo y se optimiza su código
- Además, ocurren muy pocos errores

Jerarquías de herencias

- En varias situaciones, es habitual clasificar a los conceptos como jerarquías
  - Una jerarquía es una manera de categorizar la relación entre ideas, conceptos o cosas con el componente más general o integral en primer lugar, además del más específico o el componente con el alcance más limitado en la parte inferior
  - Las jerarquías son un concepto útil cuando se trata de herencia y se pueden usar para modelar y organizar la relación entre superclases y subclases

Diagramas de árbol

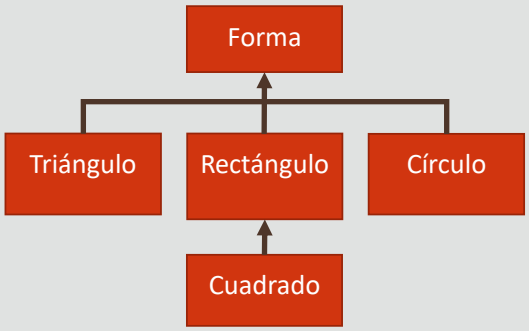
- Las jerarquías se pueden organizar en diagramas de árbol
  - Los ingenieros en sistemas con frecuencia se referirán a los árboles con hojas y ramas o se referirán a las “hojas” como nodos
  - Por ejemplo, las formas se pueden clasificar según diferentes propiedades, por ejemplo, la cantidad de lados



ORACLE Academy

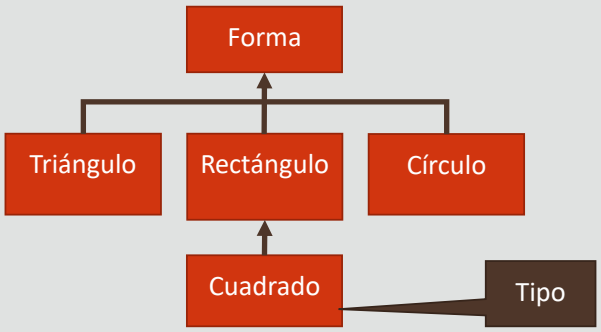
Diagramas de árbol

- Observe que el Círculo, el Triángulo y el Rectángulo tienen una cantidad diferente de lados, de modo que se consideran diferentes ramas del árbol
- Solo los nodos con las mismas propiedades ocuparán la misma rama



Tipo de nodos padres

- Cualquier elemento debajo de un nodo en un árbol es un tipo de nodo padre
- Sabemos que el Cuadrado es un tipo de Rectángulo
- El Triángulo, el Rectángulo y el Círculo son tipos de formas



Jerarquías de herencias: “Is A”

- Con las jerarquías de clases puede usar la frase “is-a” para describir una relación jerárquica
- Un nodo en una rama se puede considerar del mismo tipo que el nodo de la raíz
- Ejemplo: un Rectángulo “is-a” (es una) Forma, dado que posee todas las propiedades de una forma
- Para modelar las relaciones entre clases, usamos UML

Unified Modeling Language: UML

- Los ingenieros en sistemas modelan jerarquías de herencia usando un lenguaje de modelado denominado Unified Modeling Language or UML
- Se entiende por UML a la manera de describir las relaciones entre clases en un sistema o representación gráfica de un sistema
- UML fue desarrollado por Grady Booch, James Rumbaugh e Ivar Jacobson, y se encuentra estandarizado, de modo que se pueda comprender en todos los lenguajes

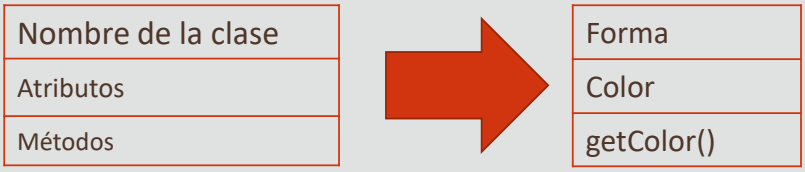
Componentes básicos de UML

- Las jerarquías de la herencia se pueden modelar simplemente con UML
- Se requieren pocos componentes simples para comenzar:
  - Diagrama de clase:
    - muestra el nombre de la clase y cualquier dato importante o métodos en la clase
  - Flechas y líneas:
    - muestran la relación de una clase con otras clases

UML es útil para la creación de modelos en mayor medida de lo que se muestra en este curso.

Diagrama de clases en UML

- Una clase se puede dibujar como una caja que contiene el nombre de la clase, las variables de instancia y los métodos
- Las clases también se pueden dibujar como cajas simples, incluyendo solo el nombre de la clase, aunque puede ser útil incluir los métodos
- No es necesario incluir cada atributo o aquellos atributos que representan conjuntos de datos (como las arreglas) Incluya únicamente los atributos más útiles



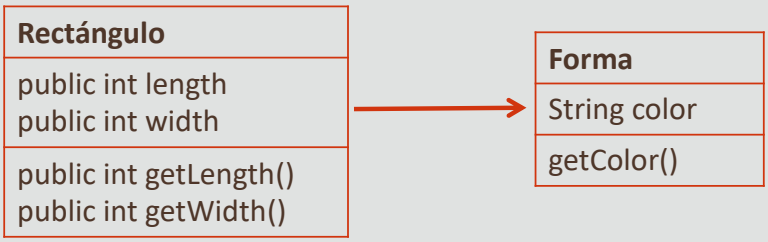
Más detalles sobre UML

- UML es una herramienta útil para que pueda planificar de qué manera organizar las jerarquías de clases de múltiples niveles
- Se alienta el uso de UML para los proyectos de codificación
- Para obtener información más detallada de UML, visite:
  - http://www.oracle.com/technetwork/developer-tools/jdev/gettingstartedwithumlclassmodeling-130316.pdf

Demostración de herencia en UML

- Los diagramas de clase se conectan usando líneas con flechas
- La conexión de las líneas varía según la relación entre las clases
- Para la herencia, se utiliza una línea sólida y una flecha triangular para representar la relación “is-a”

Relación	Símbolo
Herencia	→



Encapsulación

- La encapsulación es un concepto fundamental en programación orientada a objetos

Se entiende por encapsulación la acción de encerrar a un elemento en una cápsula o contenedor, como al poner una carta en un sobre. En programación orientada a objetos, la encapsulación encierra o encapsula los trabajos internos de una instancia/objeto de Java.

Cómo funciona la encapsulación

- En programación orientada a objetos, la encapsulación encierra o encapsula los trabajos internos de una instancia/objeto de Java
- Las variables de datos o campos están ocultos del usuario del objeto
- Los métodos pueden brindar acceso a los datos privados o trabajar con los datos, pero los métodos ocultan la implementación
- Al encapsular sus datos evita las modificaciones por parte del usuario o por otras clases, de modo que los datos no sean dañados

Cómo se usa la encapsulación

- La encapsulación se puede usar para proteger datos sensibles, como la información personal, al evitar que los datos sufran cambios, excepto dentro del alcance de la clase propiamente dicha
- Los datos se protegen y la implementación se oculta al declarar modificadores de acceso en las variables y métodos
- Los modificadores de acceso (público, privado, protegido, “predeterminado”) son palabras clave que determinan si otras clases pueden tener acceso a los datos o a los métodos de la clase o no

Modificadores de acceso

- Los programadores pueden personalizar la visibilidad de sus datos y métodos con varios niveles de modificadores de acceso

Modificador de acceso	Acceso autorizado al:
public (público)	Cualquier clase en cualquier paquete
private (privado)	Solo para otros métodos dentro de la clase propiamente dicha
protected (protegido)	Todas las subclases y todas las clases en el mismo paquete
"default" ("predeterminado")	Cualquier clase en el paquete En realidad, cuando no se especifica ninguna palabra clave, la palabra predeterminado NO se utiliza

Declaración de modificadores de acceso

- La regla general para declarar modificadores de acceso es que cualquier dato que desea proteger de alteraciones provocadas por otras clases o los datos que son sensibles deberán ser declarados privados
- Esto incluye a las variables
- Los métodos se declaran generalmente como públicos de modo que otras clases los puedan usar
- Sin embargo, los métodos se pueden declarar privados cuando están destinados a ser utilizados únicamente por la clase propiamente dicha

Ejemplo de declaración de modificadores de acceso

- Si la clase Forma contenía datos para el color, los datos en esta clase serán privados

```
public class Shape {
   private String color;
}//end class Shape
```

Acceso de miembros

- Use la palabra clave privado para ocultar los datos que solamente la clase debería poder cambiar
- (Este es el modificador de acceso recomendado)
- Si es necesario acceder a los datos, se deberá escribir un método captador para lograrlo

```
public class Shape {
    private String color; //the color of the Shape

  //Method which returns the color
  public String getColor() {
    return color;
    }//end method getColor
}//end class Shape
```

Con la clase Forma, no deseamos que los objetos cambien el color de esta Forma Esto se logra al escribir privado en la declaración de la variable

El método getColor() devuelve el color Cualquier método destinado a acceder a los datos privados deberá ser público

Acceso de miembros

- Si las variables privadas necesitan ser (o se permite que sean) cambiadas, se deberá escribir un método establecedor

```
public class Shape {
   //the color of the Shape
   private String color;
   //Method which returns the color
   public String getColor() {
      return color;
   }//end method getColor
   //Method to change the color
   public void setColor(String c){
      color = c;
   }//end method setColor
}//end class Shape
```

Si el color necesita ser cambiado, se crea un método establecedor

¿Usa público o protegido para tener acceso a los datos?

- Si deseamos contar con la capacidad de alterar el color de la variable Forma desde afuera del código de clase Forma, podríamos establecer el color de la variable del String para que sea público o protegido
- Sin embargo, se recomienda que las variables de clases se declaren como privadas

```
public class Shape {
   protected String color;
}//end class Shape
```

```
public class Shape {
   public String color;
}//end class Shape
```

Cambio de color

- Si la variable se declara como pública, el código que extiende o crea un objeto Forma podría cambiar el color sin usar un método descriptor de acceso como setColor()

```
//example for extending the shape class, then changing the
color
super.color = "Blue";

//example for creating a Shape object and changing the
color
Shape s1 = new Shape();
s1.color = "Blue";
```

No recomendado

No recomendado

Acceso de miembros y herencia

- De qué manera estos modificadores de acceso afectan a la herencia?
- Con la encapsulación, ni siquiera las subclases pueden acceder a los métodos y variables privados
- Los modificadores públicos y protegidos brindan acceso a los métodos y variables de las superclases

Modificador de acceso	Acceso autorizado al:
public (público)	Todas las clases
private (privado)	Solo la clase propiamente dicha
protected (protegido)	Todas las subclases y todas las clases en el mismo paquete
"default“ ("predeterminado“)	Si no se especifica una palabra clave, se puede acceder a las variables de los miembros mediante cualquier clase en el paquete

Extensión de la clase Forma

- Dado que la Forma no es una clase específica, podemos extenderla al crear clases más específicas, como, Rectángulo y Cuadrado
- Comenzaremos por crear una clase Rectángulo que extiende a la clase Forma

Extensión de la clase Forma

- Usted puede:
  - Escribir su propio constructor o constructores
  - Usar el constructor predeterminado
    - Si no declara un constructor, se le proporciona un constructor predeterminado sin argumento
    - Si declara su propio constructor, ya no se le proporciona el constructor predeterminado

Constructores de herencia

- Aunque una subclase hereda todos los métodos y campos de una clase padre, no hereda los constructores
- Usted puede:
  - Escribir su propio constructor o constructores
  - Usar el constructor predeterminado
    - Si no declara un constructor, se le proporciona un constructor predeterminado sin argumento
    - Si declara su propio constructor, ya no se le proporciona el constructor predeterminado

Uso de la palabra clave super en un constructor

- Al crear el objeto Rectángulo, necesitará establecer el color del Rectángulo
- Si el color de la variable es privado en la superclase Forma, ¿cómo lo establece?
- Para construir una instancia de una subclase, con frecuencia es más sencillo invocar al constructor de la clase padre

Uso de la palabra clave super en un constructor

- La palabra clave super se usa para invocar al constructor del padre
- Debe ser la primera sentencia del constructor
- Si no aparece, se insertará implícitamente una invocación predeterminada a super()
- La palabra clave super también se puede usar para invocar el método del padre o acceder al campo del padre (no privado)

Por ello, proporcionar siempre un constructor por defecto es una buena práctica al sobrecargar constructores. Si no se proporciona uno en la superclase, es posible que falle la instanciación de la subclase si se utiliza la llamada a super() por defecto.

Ejemplo del uso de la palabra super

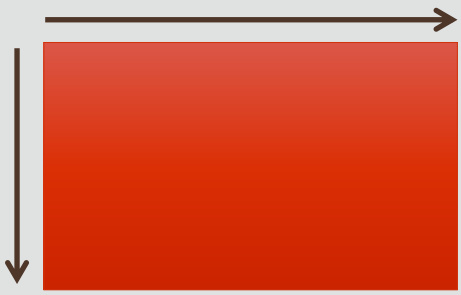
```
public class Rectangle extends Shape
{
    private int length;
    private int width;

    //Constructor
    public Rectangle(String color, int length, int width)
    {
        super(color);
        this.length = length;
        this.width = width;
    }//end constructor
}//end class Rectangle
```

Invoca al constructor para Forma, que inicializa el color de la variable

Suma de métodos para la clase Rectángulo

- Métodos Rectángulo que se recomienda incluir:

```
public int getWidth()
public int setWidth(int width)
public int getHeight()
public int setHeight(int height)
public int getArea()
```



Métodos de clase Rectángulo

- Dado que el Cuadrado es un tipo de Rectángulo o se extiende a la clase Rectángulo, heredará todos los métodos de la superclase Rectángulo:

```
public int getWidth()
public int setWidth(int width)
public int getHeight()
public int setHeight(int height)
public int getArea()
```

Configuración de la clase

- Este código configura la clase
- Use la palabra clave extends para heredar los métodos del Rectángulo

```
public class Square extends Rectangle {

}//end class Square
```

Escritura del constructor

- Para escribir el constructor, considere qué valores deben ser inicializados
- Si usamos el super constructor Rectángulo, le debemos pasar los valores: color del String, longitud de int (entero) y ancho de int (entero)
- Nuestro constructor Cuadrado requiere estos valores como parámetros si deseamos invocar al constructor super

```
public class Square extends Rectangle {
    public Square(String color, int length, int width) {

    }//end consructor
}//end class Square
```

¿La longitud y el ancho no son iguales para un cuadrado?

Parámetro de tamaño

- Aunque los Cuadrados son un tipo de Rectángulo, tienen una propiedad única, de modo que la longitud = ancho
- Para acomodarlos únicamente se requiere un parámetro de tamaño que establece los valores del ancho y de la longitud

```
public class Square extends Rectangle {

    public Square(String color, int size) {
       super(color, size, size);
   }//end constructor
}//end class Square
```

En lugar de pasar dos parámetros diferentes de longitud y ancho, podemos pasar un parámetro de tamaño dos veces, que establecerá los valores de longitud y ancho (ubicados en la clase Rectángulo) para que sean iguales

Variables únicas para la subclase

- ¿Qué sucede con las variables únicas que se aplican solamente a los Cuadrados, pero no a los Rectángulos?
- Por ejemplo, una función que nos indique si debemos completar un Cuadrado o no
- Agregue un valor booleano en la lista de parámetros para agregar esta variable única para la clase Cuadrado:

```
public class Square extends Rectangle {
   private boolean isFilled;
   public Square(String color, int size, boolean isFilled){
        super(color, size, size);
        this.isFilled = isFilled;
   }//end constructor
}//end class Square
```

la variable isFilled es única con relación a la clase Cuadrado y es un ejemplo de la manera en que las subclases pueden contener más métodos o campos que sus superclases

Métodos a medida

- Dado que Cuadrado tiene los mismos valores para alto y ancho, deseamos personalizar los métodos setWidth(int width) y setHeight(int height), de modo que ambos se actualicen cuando se invoque el método
- Use la palabra clave super para invocar los métodos de las superclases setLength() y setWidth() y establézcalos en el valor del parámetro pasado al método

```
 public int setWidth(int width){
     super.setLength(width);
     super.setWidth(width);
}//end method setWidth
```

Subclase Cuadrado

- El producto final se verá de la siguiente manera:

```
public class Square extends Rectangle {
  private boolean isFilled;
  public Square(String color, int size, boolean isFilled){
     super(color, size, size);
     this.isFilled = isFilled;
  }//end constructor

  public void setLength(int length) {
     super.setLength(length);
     super.setWidth(length);
  }//end method setLength
  public void setWidth(int width) {
     super.setWidth(width);
     super.setLength(width);
  }//end method setWidth
  public boolean getIsFilled() {
     return isFilled;
  }//end method getIsFilled
}//end class Square
```

Herencia y Applets

- Las Applets de Java representan otro ejemplo del uso de la herencia
- Una Applet de Java es un programa Java basado en la web que se puede incorporar a un navegador
- El Applet de clase se puede extender para crear Applets especiales usando alguno de los métodos principales en la clase del Applet

Documentación de Java para la clase del Applet

- Visite la documentación de Java sobre la clase del Applet, para obtener más información
- Para ver toda la documentación, visite:
  - http://docs.oracle.com/javase/8/docs/api/
- Para ver solo la documentación de la clase del Applet, visite:
  - http://docs.oracle.com/javase/8/docs/api/java/applet/Applet.html

Creación de Applets

- Para crear un Applet, puede pedir todos los métodos principales de la clase del Applet y personalizar estos métodos para que se adapten a las necesidades particulares de su Applet

Creación de Applets

- Por ejemplo, para hacer un Applet que dibuje Formas, comience por configurar la herencia con extends:

```
public class DrawShapes extends Applet {
   ...
}//end class DrawShapes
```

- Ahora la clase de nuestro Applet DrawShapes heredará los métodos del Applet que podemos personalizar para desarrollar el Applet

Ejemplo de Applet

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Graphis2D;
import java.awt.Rectangle;

public class RectangleApplet extends Applet{

   public void paint(Graphics g){
      Graphics2D g2 = (Graphics2D)g;
      Rectangle testRectangle = new Rectangle(5,10,20,30);
      g2.draw(testRectangle);
   }//end method paint

}//end class RectangleApplet
```

Terminología

- Los términos clave usados en esta lección son los siguientes:
  - Modificadores de acceso
  - Clase hijo
  - Predeterminado
  - Encapsulación
  - Extends (extiende)
  - Jerarquía
  - Herencia
  - Relación “Is-a”

Terminología

- Los términos clave usados en esta lección son los siguientes:
  - Clase padre
  - Privado
  - Protegido
  - Público
  - Subclase
  - Super (súper)
  - Superclase
  - Unified Modeling Language (UML)

Descripción general

- En esta lección, habrá aprendido a:
  - Demostrar y explicar los diagramas de clase UML (Unified Modeling Language)
  - Usar la palabra clave extends para heredar una clase
  - Comparar y contrastar superclases y subclases
  - Describir de qué manera afecta la herencia el acceso a los miembros
  - Usar super para invocar al constructor de una superclase
  - Usar super para acceder a los miembros de una superclase

Descripción general

- En esta lección, habrá aprendido a:
  - Crear una jerarquía de clases de niveles múltiples
  - Reconocer cuándo se invocan los constructores en la jerarquía de una clase
  - Demostrar la comprensión de la herencia mediante el uso de Applets
  - Reconocer los cambios de parámetros correctos en una Applet existente


ORACLE
Academy