Herencia

La herencia es uno de los tres principios básicos de la programación orientada a objetos, ya que permite la creación de clasificaciones jerárquicas. Al usar la herencia, puedes crear una clase general que defina rasgos comunes a un conjunto de elementos relacionados. Esta clase puede ser heredada por otras clases más específicas, cada una agregando aquellas cosas que son exclusivas de ella.

En el lenguaje de Java, una clase de la que se hereda se llama superclase. La clase que hace la herencia se llama una subclase. Por lo tanto, una subclase es una versión especializada de una superclase. Hereda todas las variables y métodos definidos por la superclase y agrega sus propios elementos únicos.

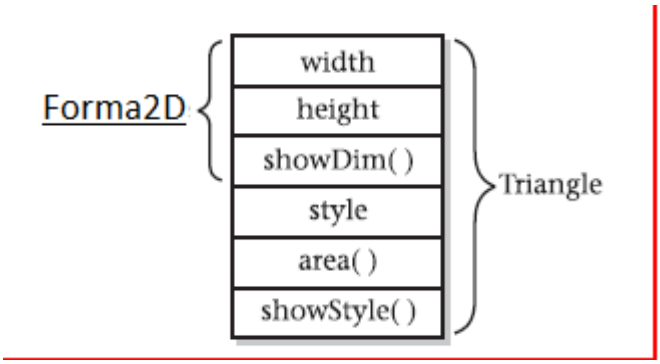
Inheritance Basics

Java admite la herencia al permitir que una clase incorpore otra clase en su declaración. Esto se hace usando la palabra clave extended. Por lo tanto, la subclase agrega a (extiende) la superclase.

```
public class Forma2D {

   double width;
   double height;

   void showDim() {
      System.out.println("Width and height are " + width + " and " + height);
   }
}

class Triangle extends Forma2D {

   String style;
   double area() {
      return width * height / 2;
   }
   void showStyle() {
      System.out.println("Triangle is " + style);
   }
}
```

Ejemplo de uso:

```
class Formas {

   public static void main(String args[]) {
      Triangle t1 = new Triangle();
      Triangle t2 = new Triangle();
      t1.width = 4.0;
      t1.height = 4.0;
      t1.style = "filled";
      t2.width = 8.0;
      t2.height = 12.0;
      t2.style = "outlined";
      System.out.println("Info for t1: ");
      t1.showStyle();
      t1.showDim();
      System.out.println("Area is " + t1.area());
      System.out.println();
      System.out.println("Info for t2: ");
      t2.showStyle();
      t2.showDim();
      System.out.println("Area is " + t2.area());
   }
}
```



Member Access and Inheritance

a menudo una variable de instancia de una clase se declarará privada para evitar su uso no autorizado. Heredar una clase no invalida la restricción de acceso privado. Por lo tanto, aunque una subclase incluye a todos los miembros de su superclase, no puede acceder a aquellos miembros de la superclase que han sido declarados privados.

```
public class AccesoForma2D {

   private double width; // these are
   private double height; // now private

   void showDim() {
      System.out.println("Width and height are "
            + width + " and " + height);
   }
}

class AccesoTriangle extends AccesoForma2D {

   String style;

   double area() {
      return width * height / 2; // Error! can't access
   }

   void showStyle() {
      System.out.println("Triangle is " + style);
   }
}
```

La solución estaría en proporcionar métodos para poder acceder a estas variables:

```
class AccesoForma2D {

   private double width; // these are
   private double height; // now private

   double getWidth() {
      return width;
```

```
  }

  double getHeight() {
    return height;
  }
  void setWidth(double w) {
    width = w;
  }

  void setHeight(double h) {
    height = h;
  }

  void showDim() {
    System.out.println("Width and height are " + width + " and " + height);
  }
}


class AccesoTriangle2 extends AccesoForma2D {

  String style;

  double area() {
    return getWidth() * getHeight() / 2;
  }

  void showStyle() {
    System.out.println("Triangle is " + style);
  }
}
```

Constructors and Inheritance

En una jerarquía, es posible que tanto las superclases como las subclases tengan sus propios constructores.

El constructor para la superclase construye la porción de superclase del objeto, y el constructor para la subclase construye la parte de la subclase.

Ejemplo1 (La Superclase no tiene constructor)

```
class Forma2D {
   private double width; // these are
   private double height; // now private
   double getWidth() {
      return width;   }

   double getHeight() {
      return height;   }

   void setWidth(double w) {
      width = w;   }

   void setHeight(double h) {
      height = h;   }

   void showDim() {
      System.out.println("Width and height are " + width + " and " + height);   }

}

class Triangle extends Forma2D {
   private String style;
   Triangle(String s, double w, double h) { //Constructor
      setWidth(w);
      setHeight(h);
      style = s;
   }

   double area() {
```

```
        return getWidth() * getHeight() / 2;    }

    void showStyle() {
        System.out.println("Triangle is " + style);    }
}
```

Using super to Call Superclass Constructors

Una subclase puede llamar a un constructor definido por su superclase mediante el uso de la siguiente forma de super:

super (lista de parámetros);

Aquí, la lista de parámetros especifica los parámetros que necesita el constructor en la superclase.

super () siempre debe ser la primera instrucción ejecutada dentro de un constructor de subclase.

```
class Forma2D {
    private double width;
    private double height;

    Forma2D(double w, double h) {
        width = w;
        height = h;
    }
    double getWidth() {
        return width;    }
    double getHeight() {
        return height;    }
    void setWidth(double w) {
        width = w;    }
    void setHeight(double h) {
        height = h;    }
    void showDim() {
        System.out.println("Width and height are " + width + " and " + height);
    }
}

class Triangle extends Forma2D {

    private String style;
```

```
  Triangle(String s, double w, double h) {
      super(w, h); // call superclass constructor
      style = s;
    }
    double area() {
      return getWidth() * getHeight() / 2;    }

    void showStyle() {
      System.out.println("Triangle is " + style);    }
  }
```

Add more constructors to Forma2D

```
class Forma2D {

  private double width;
  private double height;

  Forma2D() {   //Primer constructor
       width = 0.0;
      height = 0.0;
  }

  Forma2D(double w, double h) {          //Segundo constructor
    width = w;
    height = h;
  }

  Forma2D(double x) {  //Tercer constructor
    width = x;
    height = x;
  }
…
}

class Triangle extends Forma2D {

  private String style;

  Triangle() {
    super();
    style = "none";
  }
```

```
  Triangle(String s, double w, double h) {
    super(w, h);          // call superclass constructor
    style = s;
  }

  Triangle(double x) {
    super(x);    // call superclass constructor
    style = "filled";
  }
…
}
```

Using super to Access Superclass Members

Hay una segunda forma de super que actúa de esta manera, excepto que siempre se refiere a la superclase de la subclase en la que se usa. Este uso tiene la siguiente forma general:

```
super.member
```

Here, *member* can be either a method or an instance variable.

```
          /* clase base vehicle */
class  Vehicle
{
int  maxSpeed = 120;
}
/* subclase Car extendiendo de vehicle */
class  Car extends  Vehicle
{
int  maxSpeed = 180;
void  display()
{
/* imprime maxSpeed de la clase base (vehicle) */
System.out.println("Velocidad máxima: "  + super.maxSpeed);
/* imprime maxSpeed de la subclase  (car) */
System.out.println("Velocidad máxima: " + maxSpeed);
}
}
/* Programa de controlador Test */
class  Test
{
public  static  void  main(String[]  args)
{
Car small = new  Car();
small.display();
}
}
```

Creating a Multilevel Hierarchy

Hasta este punto, hemos estado utilizando jerarquías de clases simples que consisten de solo una superclase y una subclase. Sin embargo, puede crear jerarquías que contengan tantas capas de herencia como desee. Como se mencionó, es perfectamente aceptable usar una subclase como una superclase de otra.

```
class ColorTriangle extends Triangle {
   private String color;

   ColorTriangle(String c, String s, double w, double h) {
      super(s, w, h);
      color = c;   }

   String getColor() {
      return color;   }

   void showColor() {
      System.out.println("Color is " + color);   }
}
```

Superclass References and Subclass Objects

Java es un lenguaje fuertemente tipado. Aparte de las conversiones estándar y las promociones automáticas que se aplican a sus tipos primitivos, la compatibilidad de tipos se aplica estrictamente. Por lo tanto, una variable de referencia para un tipo de clase normalmente no puede referirse a un objeto de otro tipo de clase

```
class X {
   int a;
   X(int i) {
      a = i;   }
}
class Y {
   int a;
   Y(int i) {
      a = i;   }
}
class IncompatibleRef {
   public static void main(String args[]) {
```

```
      X x = new X(10);
      X x2;
      Y y = new Y(5);
      x2 = x; // OK, both of same type
      x2 = y; // Error, not of same type
   }
}
```

Compatibilidad entre objetos derivados de una clase y una subclase

```
class X {
   int a;

   X(int i) {
      a = i;   }
}

class Y extends X {
   int b;

   Y(int i, int j) {
      super(j);
      b = i;   }
}

class SupSubRef {

   public static void main(String args[]) {
      X x = new X(10);
      X x2;
      Y y = new Y(5, 6);

      x2 = x; // OK, both of same type
      System.out.println("x2.a: " + x2.a);

      x2 = y; // still Ok because Y is derived from X
      System.out.println("x2.a: " + x2.a);

// X references know only about X members
      x2.a = 19; // OK
   // x2.b = 27; // Error, X doesn't have a b member
   }
}
```

Usando this()

La otra posibilidad a super() es el uso de this() en un constructor. Esto lo que hace es invocar a otro constructor que este en la misma clase y que soporte el conjunto de parámetros que le pasamos.

```
public class Persona {
 private String nombre;
public Persona(String nombre) {
this.nombre = nombre;
}
public String getNombre() {
return nombre;
}
public void setNombre(String nombre) {
this.nombre = nombre;
}
}
```

```
public class Deportista extends Persona{
private String deporte;
public Deportista(String nombre) {
super(nombre);
}
 public Deportista(String nombre, String deporte) {
this(nombre);
this.deporte = deporte;
}
public String getDeporte() {
return deporte;
}

public void setDeporte(String deporte) {
this.deporte = deporte;
}
}
```

El uso de this() y de super() es excluyente o usamos uno u otro.

Ejercicio 1:

Crea una **subclase** llamada *Programador,* que herede de la clase ***Empleado*** *del capítulo 9.* Añade el siguiente atributo privado:

• *String[ ] lenguajes.* Almacenará los lenguajes de programación que conoce.

• El constructor tendrá el siguiente aspecto:

Programador (String nombre, String telefono, String … lenguajes)

• Implementa un método público para recuperar los lenguajes de programación:

String[ ] getLenguajes()

• Crea una clase aparte con el método *main* que rellene un array de empleados, pudiéndose cargar programadores.

Ejercicio 2:

Crea una **subclase** llamada *Comercial,* que herede de la clase *Empleado,* creada en el ejercicio anterior.

• Tendrá los siguientes **atributos**:

◦ *float totalVentas.* Guardará el total de ingresos por ventas acumulados.

◦ *String ciudad.* Ciudad en la que opera el comercial.

◦ *String productosVendidos.* Separados por una coma, almacena los productos que va vendiendo.

• Y los siguientes **métodos**:

◦ *String getCiudad ( )*

◦ *void sumarVenta (float importe)* Suma *importe* a *totalVentas.*

◦ *float getTotalVentas( )*

◦ *void vendeProducto (String producto)* Añade el producto al atributo *productosVendidos*, separado por una coma del resto.

◦ *String[ ] getProductosVendidos ( )* Devuelve un array con los nombres de los productos vendidos.

• En el **constructor** hay que pasarle la *ciudad* en la que opera el comercial. *totalVentas* y *productosVendidos* son inicializados a 0 y “” respectivamente.

• Crea una clase aparte con el método *main* para probar todas las funcionalidades de la subclase *Comercial.*