INSTITUTO SAN ISIDRO

Curso de Java a distancia

Clase 30: Clases abstractas

El lenguaje Java permite definir clases que no se pueden instanciar o definir objetos de las mismas, este tipo de clases se las llama clases abstractas.

Una clase abstracta tiene por objetivo agrupar un conjunto de propiedades, métodos y firmas de métodos (métodos sin implementar, que deberán ser implementados por sus subclases)

Por si misma la clase abstracta no tiene sentido que se creen objetos, inclusive se genera un error sintáctico si se lo intenta.

Es común que se definan muchas subclases de una clase abstracta. Mediante un ejemplo veremos la sintaxis para crear una clase abstracta y como las subclases implementan los métodos abstractos de la misma.

Problema:

Confeccionar una clase abstracta que represente una Lista simplemente encadenada que defina dos métodos abstractos llamados: insertar y extraer un entero. Además definir un método que imprima la lista.

Luego plantear dos clases llamadas Pila y Cola que hereden de la clase abstracta Lista e implementen los dos métodos abstractos.

Clase: Lista

```
public abstract class Lista {
   public class Nodo {
      int info;
      Nodo sig;
   }

   protected Nodo raiz;

   public abstract void insertar(int x);

   public abstract int extraer(int x);

   public void imprimir() {
      Nodo reco = raiz;
      while (reco != null) {
         System.out.print(reco.info + " ");
         reco = reco.sig;
      }
      System.out.println();
```

```
    }
}
```

Una clase es abstracta si se le agrega la palabra clave 'abstract' previo a 'class', con esto ya no podemos crear objetos de la misma:

Lista lista1=new Lista(); // error de compilación

Una clase abstracta puede declarar clases internas:

```
   public class Nodo {
      int info;
      Nodo sig;
   }
```

Definir atributos:

```
   protected Nodo raiz;
```

Implementar métodos:

```
   public void imprimir() {
      Nodo reco = raiz;
      while (reco != null) {
         System.out.print(reco.info + " ");
         reco = reco.sig;
      }
      System.out.println();
   }
```

Podemos declarar métodos abstractos que obligan a las subclases a implementarlos:

```
   public abstract void insertar(int x);

   public abstract int extraer(int x);
```

Para que una clase abstracta tenga sentido deben declararse subclases de la misma, en nuestro ejemplo implementaremos las subclases 'Pila' y 'Cola'.

Clase: Pila

```
public class Pila extends Lista {

   public void insertar(int x) {
      Nodo nuevo = new Nodo();
      nuevo.info = x;
      nuevo.sig = raiz;
      raiz = nuevo;
   }

   public int extraer(int x) {
      if (raiz == null)
         return Integer.MAX_VALUE;
      else {
         int valor = raiz.info;
         raiz = raiz.sig;
         return valor;
      }
   }

   public static void main(String[] args) {
      Pila pila1 = new Pila();
```

```
        pila1.insertar(20);
        pila1.insertar(5);
        pila1.insertar(600);
        pila1.imprimir();
    }

}
```

Recordemos que mediante la palabra clave 'extends' indicamos que una clase hereda de otra:
public class Pila extends Lista {

Como heredamos de la clase Lista debemos obligatoriamente implementar los métodos insertar y extraer (en el caso que no lo hagamos se genera un error sintáctico):

```
    public void insertar(int x) {
        Nodo nuevo = new Nodo();
        nuevo.info = x;
        nuevo.sig = raiz;
        raiz = nuevo;
    }

    public int extraer(int x) {
        if (raiz == null)
            return Integer.MAX_VALUE;
        else {
            int valor = raiz.info;
            raiz = raiz.sig;
            return valor;
        }
    }
```

La clase Pila es una clase 'concreta' y no 'abstracta', luego podemos crear objetos de la misma:

```
    public static void main(String[] args) {
        Pila pila1 = new Pila();
        pila1.insertar(20);
        pila1.insertar(5);
        pila1.insertar(600);
        pila1.imprimir();
    }
```

De forma similar codificamos la clase 'Cola' e implementamos los algoritmos para insertar y extraer de una cola de enteros.

Clase: Cola

```
public class Cola extends Lista {

    public void insertar(int x) {
        Nodo nuevo = new Nodo();
        nuevo.info = x;
        if (raiz == null) {
            nuevo.sig = raiz;
            raiz = nuevo;
        } else {
            Nodo reco = raiz;
            while (reco.sig != null)
                reco = reco.sig;
            reco.sig = nuevo;

        }
    }
```

```
    public int extraer(int x) {
        if (raiz == null)
            return Integer.MAX_VALUE;
        else {
            int valor = raiz.info;
            raiz = raiz.sig;
            return valor;
        }
    }

    public static void main(String[] args) {
        Cola cola1 = new Cola();
        cola1.insertar(20);
        cola1.insertar(5);
        cola1.insertar(600);
        cola1.imprimir();
    }
}
```

Una clase abstracta permite que un conjunto de clases que heredan de la misma tengan una serie de métodos homogéneos para administrarlas. En nuestro ejemplo siempre que tenemos que agregar o eliminar elementos de una pila o cola debemos llamar obligatoriamente a los métodos 'insertar' y 'extraer' que son las firmas definidas en la clase abstracta.

Problema propuesto

1. Plantear una clase abstracta llamada 'Operacion' que defina los atributos enteros:
   valor1
   valor2
   resultado

Define el método abstracto:

operar

Defina el método concreto imprimir que muestre el atributo resultado:

imprimir

Declarar cuatro clases que hereden de la clase Operacion llamadas: Suma, Resta, Multiplicacion y Division.

Crear un objeto de cada clase que hereda de Operacion.

Solución

```
public abstract class Operacion {

   protected int valor1, valor2, resultado;

   public Operacion(int valor1, int valor2) {
      this.valor1 = valor1;
      this.valor2 = valor2;
   }

   public abstract void operar();

   public void imprimir() {
      System.out.println(resultado);
   }
}
```

```
public class Suma extends Operacion {

   public Suma(int valor1, int valor2) {
      super(valor1, valor2);
   }

   public void operar() {
      resultado = valor1 + valor2;
   }

   public void imprimir() {
      System.out.print("La suma de " + valor1 + " y " + valor2 + " es ");
      super.imprimir();
   }

}
```

```
public class Resta extends Operacion {

   public Resta(int valor1, int valor2) {
      super(valor1, valor2);
   }

   public void operar() {
      resultado = valor1 - valor2;
   }

   public void imprimir() {
      System.out.print("La resta de " + valor1 + " y " + valor2 + " es ");
      super.imprimir();
   }

}
```

```
public class Multiplicacion extends Operacion {

    public Multiplicacion(int valor1, int valor2) {
        super(valor1, valor2);
    }

    public void operar() {
        resultado = valor1 * valor2;
    }

    public void imprimir() {
        System.out.print("El producto de " + valor1 + " y " + valor2 + " es ");
        super.imprimir();
    }

}
```

```
public class Division extends Operacion {

    public Division(int valor1, int valor2) {
        super(valor1, valor2);
    }

    public void operar() {
        resultado = valor1 / valor2;
    }

    public void imprimir() {
        System.out.print("La división de " + valor1 + " y " + valor2 + " es ");
        super.imprimir();
    }

}
```

```
public class PruebaOperaciones {

    public static void main(String[] args) {
        Suma suma1 = new Suma(10, 2);
        suma1.operar();
        suma1.imprimir();
        Resta resta1 = new Resta(10, 2);
        resta1.operar();
        resta1.imprimir();
        Multiplicacion multiplicacion1 = new Multiplicacion(10, 2);
        multiplicacion1.operar();
        multiplicacion1.imprimir();
        Division division1 = new Division(10, 2);
        division1.operar();
        division1.imprimir();
    }
}
```

Clases finales

En Java podemos sellar una clase para evitar que otras clases hereden de la misma mediante la palabra clave 'final' previo a la declaración de la clase:

```
public final class [Nombre de la clase] {
}
```

En algunas situaciones donde veamos que no tiene sentido aplicar herencia a la clase que estamos creando podemos declararla de tipo final. Si luego alguien quiere heredar de la misma se genera un error de compilación.

Problema:

Desarrollar una clase que represente un punto en el plano y tenga los siguientes métodos: constructor, imprimir en que cuadrante se encuentra dicho punto (concepto matemático, primer cuadrante si x e y son positivas, si x<0 e y>0 segundo cuadrante, etc.)

Sellar la clase para evitar que se pueda aplicar herencia en la misma.

Programa:

```
public final class Punto {
   private int x, y;

   public Punto(int x, int y) {
      this.x = x;
      this.y = y;
   }

   void imprimirCuadrante() {
      System.out.print("[" + x + "," + y + "] ");
      if (x > 0 && y > 0)
         System.out.println("Se encuentra en el primer cuadrante.");
      else if (x < 0 && y > 0)
         System.out.println("Se encuentra en el segundo cuadrante.");
      else if (x < 0 && y < 0)
         System.out.println("Se encuentra en el tercer cuadrante.");
      else if (x > 0 && y < 0)
         System.out.println("Se encuentra en el cuarto cuadrante.");
      else
         System.out.println("El punto no está en un cuadrante.");
   }

   public static void main(String[] ar) {
      Punto punto1;
      punto1 = new Punto(4, 5);
      punto1.imprimirCuadrante();
      Punto punto2;
      punto2 = new Punto(-4, 5);
      punto2.imprimirCuadrante();

   }
}
```

Hemos declarado la clase 'Punto' de tipo final para evitar que se puede heredar de ella:

```
public final class Punto {
```

Luego si se intenta heredar de la misma debe generar un error sintáctico:

Clases finales en el API de Java.

Podemos nombrar que el API de Java tiene una gran cantidad de clases definidas como final:

System
Scanner
String
Math
Byte
Short
Integer
Long
Float
Double
Boolean
Character

Como podemos ver hay numerosas clases en el API de Java que hemos utilizado a lo largo del curso que están pensadas para que no las heredemos.

Problema propuesto

Desarrollar una clase que represente un Cuadrado. Definir dos métodos que retornen la superficie y el perímetro.
Crear la clase como 'final'.

Solución

```
public final class Cuadrado {

   private int lado;

   public Cuadrado(int lado) {
      this.lado = lado;
   }

   public int retornarPerimetro() {
      return lado * 4;
   }

   public int retornarSuperficie() {
      return lado * lado;
   }

   public static void main(String[] ar) {
      Cuadrado cuadrado1 = new Cuadrado(20);
      System.out.println("El perímetro de un cuadrado de lado 20 es " +
                  cuadrado1.retornarPerimetro());
      System.out.println("La superficie de un cuadrado de lado 20 es " +
                  cuadrado1.retornarSuperficie());
   }
}
```

Métodos finales

En Java podemos sellar un método para que las subclases no puedan sobreescribirlo, para ello debemos agregar la palabra clave final previo al tipo de dato que devuelve:

```
public final void imprimir()
```

El método imprimir no podrá ser implementado en una sublclase.

Un método que se lo declara como final es más eficiente, ya que el compilador puede vincular con el código definido en dicho método ya que las subclases no lo pueden implementar.

Problema:

Confeccionar una clase Persona que tenga como atributos el nombre y la edad. Retornar si la persona es mayor de edad o no (definir dicho método como final)
Plantear una segunda clase Empleado que herede de la clase Persona. Añadir un atributo sueldo y los métodos de cargar el sueldo e imprimir su sueldo.

Definir un objeto de la clase Persona y llamar a sus métodos. También crear un objeto de la clase Empleado y llamar a sus métodos.

Clase: Persona

```
public class Persona {
   protected String nombre;
   protected int edad;

   public Persona(String nombre, int edad) {
      this.nombre = nombre;
```

```
        this.edad = edad;
    }

    public void imprimirDatosPersonales() {
        System.out.println("Nombre:" + nombre);
        System.out.println("Edad:" + edad);
    }

    public final boolean esMayor() {
        if (edad >= 18)
            return true;
        else
            return false;
    }
}
```

Clase: Empleado

```
public class Empleado extends Persona {
    protected int sueldo;

    public Empleado(String nombre, int edad, int sueldo) {
        super(nombre, edad);
        this.sueldo = sueldo;
    }

    public void imprimirSueldo() {
        System.out.println("El sueldo es:" + sueldo);
    }
}
```

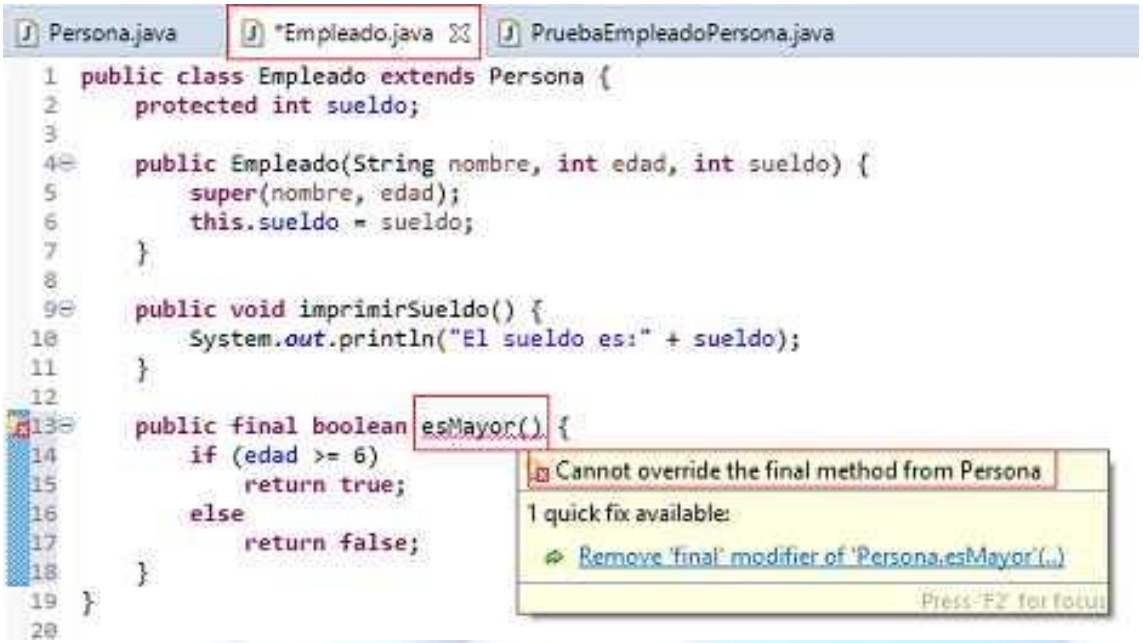
Clase: PruebaEmpleadoPersona

```
public class PruebaEmpleadoPersona {
    public static void main(String[] args) {
        Persona persona1 = new Persona("Juan", 10);
        persona1.imprimirDatosPersonales();
        if (persona1.esMayor())
            System.out.println("Es mayor de edad");
        else
            System.out.println("No es mayor de edad");
        Empleado empleado1 = new Empleado("Carlos", 22, 10000);
        empleado1.imprimirDatosPersonales();
        empleado1.imprimirSueldo();
        if (empleado1.esMayor())
            System.out.println("Es mayor de edad");
        else
            System.out.println("No es mayor de edad");
    }
}
```

Hemos definido el método 'esMayor' de tipo final:

```
    public final boolean esMayor() {
        if (edad >= 18)
            return true;
        else
            return false;
    }
```

Luego no se puede cambiar su significado en una subclase (se genera un error):



No podemos reescribir el método 'esMayor' para la clase Empleado indicando que 6 años es mayor de edad:

```
public final boolean esMayor() {
   if (edad >= 6)
      return true;
   else
      return false;
}
```

Muchas veces por seguridad evitamos que las subclases puedan sobreescribir métodos definiendo los mismos como 'final'.

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar info@institutosanisidro.com.ar