Abraham Ramírez Moreno 1902627

Modificadores de acceso

De clases e interfaces

Modificador	Descripción
`public`	Cualquier clase puede tiene acceso a esa clase.
`default (Sin modificador)`	Solo es accesible a otras clases en el mismo paquete.

Miembros

Modificador	Descripción
`public`	Todas las clases pueden acceder al miembro.
`private`	Solo se puede acceder al miembro desde adentro de la clase.
`protected`	Las clases dentro del mismo paquete pueden acceder al miembro. También las subclases de esa clase en otros paquetes pueden acceder al miembro.
`default (Sin modificador)`	Las clases dentro del mismo paquete pueden acceder a los miembros.

Conceptos básicos

- ¿Qué es un objeto?

Un objeto es una instancia de una clase. Las clases son plantillas para definir un objeto, el objeto ya es algo construido usando esa plantilla.

- ¿Qué es una clase?

Una clase es una plantilla de objetos. Especifica las variables y métodos que tiene un objeto de esa clase.

- Métodos y atributos

Un método es un comportamiento que puede tener un objeto, por ejemplo, un objeto carro puede manejar y apagarse. Un atributo es una característica de un objeto, por ejemplo, un carro tiene color y peso.

- Método Constructor

Un constructor es un método que se llama siempre que se inicializan los objetos de una clase. Las clases se crean con un constructor predeterminado que sólo crea nuevas instancias sin modificarlas.

Un constructor personalizado se puede declarar en las clases para crear una inicialización personalizada de los objetos de esa clase. Los métodos constructores deben tener el mismo nombre que la clase, no tienen tipo de retorno y normalmente son públicos. Una vez que se implementa un constructor en una clase ya no va a tener constructor predeterminado.

Ejemplo:

Sea esta una clase:

```
public class ClaseC {
    private int prop;

    public int getprop() {
        return prop;
    }

    ClaseC(int propIni) {
        this.prop = propIni;
    }
}
```

Y dentro del main tengamos:

```
ClaseC miobj = new ClaseC(5);
miobj.getprop();
```

Entonces, ClaseC define al atributo prop, y al método getprop(). ClaseC(int propIni) es el método constructor. Dentro del main creamos un objeto del tipo ClaseC llamado miobj usando al constructor. Ese objeto tiene los métodos y atributos definidos por la clase. Por ejemplo, en el programa llamamos al método getprop() de ese objeto y nos retorna el valor del atributo prop de miobj.

- clase abstracta

Una clase abstracta es una clase "incompleta" que no está hecha para ser instanciada. En vez de eso, está hecha para servir como superclase o "plano" de otras clases. Las clases abstractas pueden tener métodos abstractos, es decir, métodos que no tienen implementación (cuerpo). El trabajo de las subclases de una clase abstracta es completar la clase implementando los métodos abstractos y si es el caso agregar atributos y métodos adicionales. Si las subclases de una clase abstracta no implementan los métodos abstractos de la clase abstracta, las subclases también serán abstractas.

Ejemplo

```
public abstract class Transporte {
    boolean encendido;
    public void encender() {
        encendido = true;
        System.out.println("Encendido");
    }
    public abstract void parar();
}
```

- interfaz

Una interfaz es un grupo que consiste únicamente de métodos abstractos y constantes. Todos los elementos de una interfaz deben ser públicos. Además, una clase puede implementar más de 1 interfaz.

Ejemplo:

```
public interface IConstruir {
    public void talarMadera();
    public void hacerEstructura();
    public void reparar();
}
```

- herencia

Es la capacidad de crear nuevas clases usando los miembros de otras clases. La nueva clase se llama subclase y la clase de la que heredas se llama superclase. La subclase también puede modificar los miembros de su superclase.

Las subclases a su vez pueden ser superclases de otras clases, creando una jerarquía de clases.

Ejemplo:

De la clase ClaseC hecha anteriormente podemos crear una subclase llamada ClaseD:

```
public class ClaseD extends ClaseC{
    private boolean condicion;

    public boolean getcondicion() {
        return condicion;
    }

    public void setCondicion(boolean cond) {
        this.condicion = cond;
    }

    public ClaseD(int propIni) {
        super(propIni);
    }
}
```

Si creamos un objeto claseD, ese objeto también va a tener los atributos y clases de claseC.

```
ClaseD otro = new ClaseD(5);
otro.getprop();
```

En este caso aunque el método getprop() y el atributo prop no están directamente definidos en claseD, un objeto claseD comoquiera tiene esos miembros porque ClaseD hereda de ClaseC.

- polimorfismo

Es la capacidad de objetos de pasar por otro tipo de objetos según sea el caso. Esto permite guardar diferentes tipos de objetos bajo un mismo tipo y después llamar una instrucción a

todos esos objetos. Aunque estén guardados en el mismo tipo, cada objeto va a actuar según su implementación lo diga.

Ejemplo:

Basándose en la interface IConstruir que se uso como ejemplo, creamos 2 clases que la implementan: Castor y Albañil

```
public class Castor implements IConstruir{

    public void talarMadera() {
        System.out.println("Mordiendo troncox");
    }

    @Override
    public void hacerEstructura() {
        System.out.println("Haciendo Presa");
    }

    @Override
    public void reparar() {
        System.out.println("ReparandoHoyos");
    }
  }
public class Albanil implements IConstruir{

    @Override
    public void talarMadera() {
        System.out.println("Cortando con sierra");
    }

    @Override
    public void hacerEstructura() {
        System.out.println("Haciendo casa");
    }

    @Override
    public void reparar() {
        System.out.println("Arreglando Vigas");
    }

}
```

En el main podemos crear un arreglo de objetos tipo IConstruir, que guarda a 1 albañil y a 1 castor.

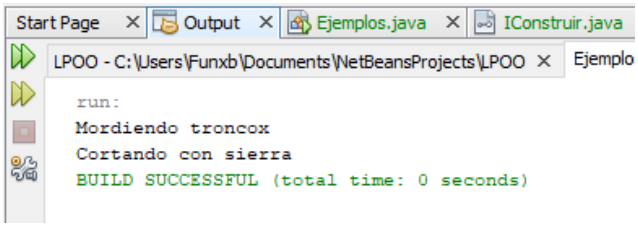
```
public static void main(String[] args) {

        IConstruir[] constructores = new IConstruir[2];
        constructores[0] = new Castor();
        constructores[1] = new Albanil();

        for(IConstruir constructor : constructores) {
            constructor.talarMadera();
        }

    }
```

Aunque son clases diferentes, el programa va a poder tomarlas como objetos del mismo tipo, y ejecutará el método talar madera de cada uno. La salida es la siguiente:



- encapsulamiento

Los objetos pueden intercambiar mucha información entre ellos, sin embargo, no es buena práctica hacer que un objeto pueda acceder y modificar a los atributos y métodos de otros. En vez de esto, los objetos deben tener un conjunto de operaciones que muestran al público y los detalles los manejan ellos mismos y nadie más. Esta ocultación de información se llama encapsulamiento.

Ejemplo:

En el ejemplo de esta clase:

```
public class ClaseC {
    private int prop;

    public int getprop() {
        return prop;
    }

    ClaseC(int propIni) {
        this.prop = propIni;
    }
}
```

La variable prop está encapsulada, pues solo puede ssada directamente por nadie más.sada directamente por nadie más.er accedida mediante los métodos definidos por la clase y no puede ser accesada directamente por nadie más.

- abstracción

Es la existencia de una separación entre la implementación de un módulo y su interfaz. Es decir, el usuario sin conocer los detalles de como funciona el programa debe poder interactuar con él sin dificultades.

Referencias

Deitel, P. Deitel, H. (2014) Java How to Program: Late Objects Version. Pearson.