TECNOLÓGICO DE ANTIOQUIA – INSTITUCIÓN UNIVERSITARIA

PROGRAMACIÓN ORIENTADA A OBJETOS

VICTOR MANUEL QUIROZ IBARRA

DOCENTE:

GILDARDO ANTONIO QUINTERO CORREA

FACULTADA DE INGENIERA

CONSTRUCCIÓN DE SOFTWARE

TECNOLOGÍA EN SISTEMAS

MEDELLÍN – ANTIOQUIA

2020

Introducción

Cuando hablamos de programación orientada a objetos nos referimos a combinación de datos y acciones asociadas en estructuras lógicas. De eso hablaremos en este trabajo, todo lo que tiene que ver con P.O.O, sus métodos, atributos, entre otros. Ampliará más su conocimiento sobre la programación y más que todo en programación orientada a objetos.

La programación orientada a objetos tiene como principio que todo en la vida es un objeto programable, entonces para empezar a programar con este paradigma tendríamos que empezar con desarrollar nuestro pensamiento basado en objetos.

Todo quedara un poco más claro en este artículo….

P.O.O.

Motivación

Durante años, los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que los esfuerzos de los programadores puedan ser reutilizados se creó la posibilidad de utilizar módulos. El primer módulo existente fue la función, que somos capaces de escribir una vez e invocar cualquier número de veces.

La programación orientada a objetos es un enfoque de programación que combina datos y acciones asociadas en estructuras lógicas. También es un paradigma de programación que usa objetos en sus interacciones, para diseñar aplicaciones y programas informáticos. Está basada en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

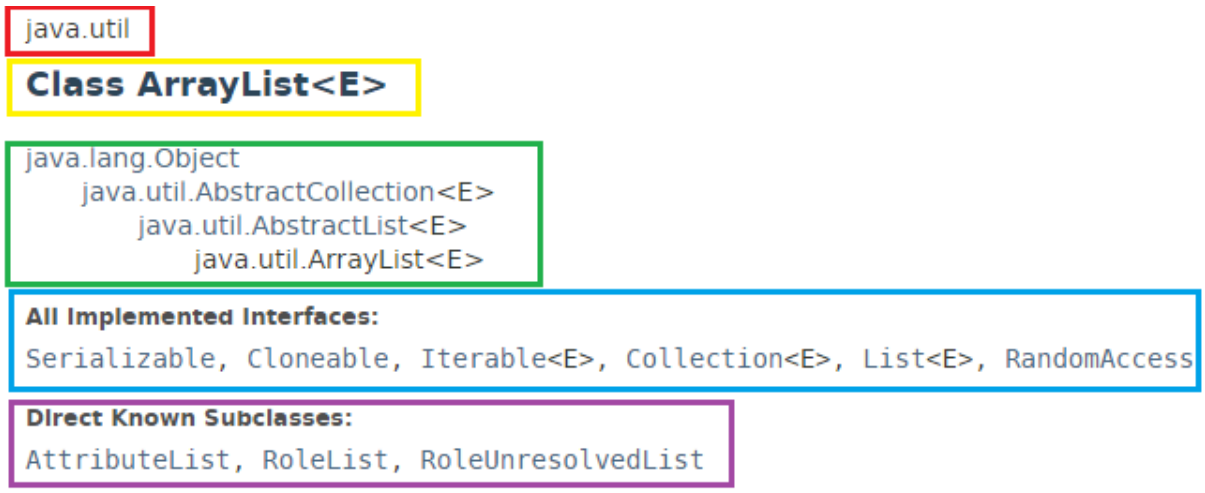
La POO es una forma especial de programar, este paradigma de programación es cercano a como expresamos las cosas en la vida real en nuestro día a día. Con la POO tenemos que aprender a pensar las cosas de una manera distinta para poder escribir nuestros programas en términos de objetos, propiedades y métodos. La POO tiene como principio que todo en la vida es un objeto programable, entonces para empezar a programar con este paradigma tendríamos que empezar con desarrollar nuestro pensamiento basado en objetos.

Clases

En informática, una clase es una plantilla para la creación de objetos de datos según un modelo predefinido. Las clases se utilizan para representar entidades o conceptos, como los sustantivos en el lenguaje. Cada clase es un modelo que define un conjunto de variables -el

estado, y métodos apropiados para operar con dichos datos. Cada objeto creado a partir de la clase se denomina instancia de la clase.

A continuación, voy a poner una imagen de la jerarquía de clases de la clase ArrayList según obtenemos de la propia documentación:



- En rojo aparece el paquete al que pertenece la clase
- En amarillo el nombre de la clase
- En verde la jerarquía de clases. ArrayList hereda de AbstractList que a su vez hereda de AbstractCollecion que a su vez hereda de Objetos. Hablaremos de la herencia más adelante.
- En azul vemos las interfaces que implementa la clase. Hablaremos de las interfaces más adelante.
- En morado vemos las clases que son descendiente de la clase ArrayList.

Los tipos de clases que existen son públicas, abstractas y finales.

Objetos

Los objetos son la clave para entender la programación orientada a objetos. Todo a nuestro alrededor puede ser considerado un objeto. Consideramos que los objetos de nuestro alrededor tienen 2 características comunes: el estado y el comportamiento. El estado hace referencia al estado actual de una característica del objeto, pensando en un coche su color, velocidad, marcha. Mientras que el comportamiento hace referencia a las acciones que el objeto puede llevar a cabo, continuando con el coche: acelerar, frenar, cambiar de marcha. En ocasiones un objeto puede estar compuesto por otros objetos. Los objetos guardan su estado en campos y exponen su comportamiento a través de métodos.

Al trabajar con objetos obtenemos una serie de beneficios como:

- Modularidad: Al escribir y mantener el código fuente de cada objeto por separado.
- Ocultación de información: Al hacer únicamente se interacciones con los métodos de un objeto conseguimos ocultar la implementación interna.
- Reusabilidad: Cuando trabajemos con la herencia veremos que si tenemos objetos funcionando podemos beneficiarnos de su implementación y evitarnos codificar lo ya hecho.
- Facilidad de uso: Si un determinado objeto nos ocasiona problemas será suficiente con eliminarlo de la aplicación y programar uno que lo reemplace.

Métodos

A los objetos podemos definirle una serie de métodos, que son instrucciones que pueden cambiar los valores que hemos asignado a esas propiedades. En el caso del Taller podrían ser métodos como realizar revisión periódica o arreglar avería, que podrían cambiar las fechas de la última revisión y de la última reparación respectivamente.

Pues bien, como quizás hayas pensado ya, los métodos no son más que funciones, como las que hemos visto en el apartado anterior. En ellos encapsulamos una serie de instrucciones que podemos luego llamar y aplicar sobre el objeto en el cuál se han definido, de forma parecida a como lo hemos hecho con las funciones, eso sí, especificando dicho objeto con la notación: objeto. método ().

Atributos

Anteriormente hemos indicado que una clase puede contener o no atributos (también llamados campos o variables miembro) y que estos atributos podían ser de tipo primitivo o bien otras clases.

Por ejemplo:

```
1  /**
2   *
3   * @author pablo
4   */
5  public class Clase {
6      //
7      int var1;
8      final int VAR_2 = 1;
9      //
10     Integer var3;
11     final Integer VAR_4 = 2;
12 }
```

En la clase Clase existe var1 (línea 7) que es un atributo variable de tipo int (tipo primitivo). Existe VAR_2 (línea 8) que es un atributo constante de tipo int (tipo primitivo). Existe var3 (línea 10) que es un atributo variable de la clase Integer y, por último, existe VAR_4 (línea 11) que es un atributo constante de la clase Integer. Aquí he utilizado la clase Integer perteneciente al API de Java, pero si tuviese creadas varias clases podría crear variables de la clase que me interesase.

Encapsulamiento

La encapsulación es un principio fundamental de la programación orientada a objetos y consiste en ocultar el estado interno del objeto y obligar a que toda interacción se realice a través de los métodos del objeto.

Para ello, el acceso a los atributos se establece como privado y se crean 2 métodos por cada atributo, un getter y un setter. El getter de un atributo se llamará getNombreAtributo mientras que el setter de un atributo se llamará setNombreAtributo. Vamos a ver un ejemplo para entender esto mejor:

```java
package clases;
/**
 * @author Pablo Ruiz Soria
 */
public class Mesa {
    private String color;

    /**
     * @return the color
     */
    public String getColor() {
        return color;
    }

    /**
     * @param color the color to set
     */
    public void setColor(String color) {
        this.color = color;
    }

}
```

```java
package lanzadores;

import clases.Mesa;

/**
 * @author Pablo Ruiz Soria
 */
public class Principal {
    public static void main(String[] args) {
        Mesa mesa = new Mesa();
        //mesa.color = "Blanco";
        mesa.setColor("Blanco");

        //System.out.println(mesa.color);
        System.out.println(mesa.getColor());
    }
}
```

```
Output - Modulo3Encapsulamiento (run)
run:
Blanco
BUILD SUCCESSFUL (total time: 1 second)
```

Si nos fijamos en Mesa.java veremos que es una clase con 1 atributo privado llamado color y 2 métodos que se llaman getColor y setColor. Los métodos anteriores serían, respectivamente, el getter y setter de del atributo color. Ahora que ya conocemos el control de acceso que sobre clases, atributos, constructores y métodos puede realizarse sabemos que no puede accederse directamente al atributo ni para obtener su valor ni para establecer su valor, siempre que quieran realizarse esas acciones habrá que pasar respectivamente por el getter y el setter y en ellos podremos programar aquello que nos interese. Si nos fijamos en Principal.java veremos que si ahora queremos acceder directamente al atributo para modificar u obtener su valor no podemos (he dejado comentado el código que falla) y que para hacerlo estamos obligados a utilizar los getter y setters que hemos creado antes.

Abstracción

La abstracción forma parte de los elementos fundamentales en el modelo de objetos. las características específicas de un objeto, aquellas que lo distinguen de los demás tipos de objetos y que logran definir límites conceptuales respecto a quien está haciendo dicha abstracción del objeto.

Una abstracción se enfoca en la visión externa de un objeto, separa el comportamiento específico de un objeto, a esta división que realiza se le conoce como la barrera de abstracción, la cual se consigue aplicando el principio de mínimo compromiso.

Hay una alta gama de abstracciones que existen desde los objetos que modelan muy cerca de entidades, a objetos que no tienen razón para existir, vamos a hacer una rápida mención de ello.

- Abstracción de Entidades: Es un objeto que representa un modelo útil de una entidad que se desea.
- Abstracción de Acciones: Un objeto que representa un conjunto de operaciones y todas ellas desempeñan funciones del mismo tipo.
- Abstracción de Máquinas virtuales: Un objeto que agrupa operaciones, todas ellas virtuales, utilizadas por algún nivel superior de control u operaciones (entre ellos podríamos hablar de un circuito).
- Abstracción de coincidencia: Un objeto que almacena un conjunto de operaciones que no tienen relación entre sí.

Herencia

Una de las características de un lenguaje de programación orientado a objetos es la herencia. En algún capítulo anterior a este ya hemos visto alguna funcionalidad de la herencia, pero de aquí en adelante vamos a profundizar en ella. En POO, la herencia es un mecanismo que nos permite extender las funcionalidades de una clase ya existente. De este modo vamos a favorecer la reutilización de nuestro código.

Vamos a ver un ejemplo a continuación:

```java
package modulo3clases;
/**
 * @author pablo
 */
public class Abuelo {
    String varAbuelo = "Abuelo";
    void metodoAbuelo(){
        System.out.println("varAbuelo: " + varAbuelo);
    }
}
```

```java
package modulo3clases;
/**
 * @author pablo
 */
public class Padre extends Abuelo{
    String varPadre = "Padre";
    void metodoPadre(){
        System.out.println("varAbuelo: " + varAbuelo);
        System.out.println("varPadre: " + varPadre);
    }
}
```

```java
package modulo3clases;
/**
 * @author pablo
 */
public class Hijo extends Padre {
    String varHijo = "Hijo";
    void metodoHijo(){
        System.out.println("varAbuelo: " + varAbuelo);
        metodoPadre();
    }
}
```

```java
package modulo3clases;
/**
 * @author Pablo Ruiz Soria
 */
public class Modulo3Clases {
    public static void main(String[] args) {
        //Creo variables
        Abuelo abuelo = new Abuelo();
        Padre padre = new Padre();
        Hijo hijo = new Hijo();
        // Padre puede llamar a sus métodos y los de su padre (abuelo)
        padre.metodoAbuelo();
        padre.metodoPadre();
        // Abuelo no hereda a nadie por lo que hace uso de sus propios métodos
        abuelo.metodoAbuelo();
        //hijo puede usar sus métodos o los de su padre o los del
        //padre de su padre (abuelo)
        hijo.metodoHijo();
        hijo.metodoPadre();
        hijo.metodoAbuelo();
    }
}
```

Polimorfismo

El polimorfismo es la capacidad que nos proporciona un lenguaje de programación orientado a objetos para tratar un objeto como si fuera un objeto de otra clase. Existen lenguajes de programación donde una variable puede contener prácticamente cualquier tipo de dato, es el caso de los lenguajes PHP, Python y Javascript (recuerda, Javascript no es Java, son lenguajes distintos), mientras que existen otros lenguajes de programación en los que una variable definida de un modo solo pude contener variables de dicho tipo, es el caso de Java.

Por ejemplo:

```java
 * @author Pablo Ruiz Soria
 */
public class Lanzador {
    public static void main(String[] args) {
        Perro sira = new Perro("Sira");
        Perro nala = new Perro();
        Pato donald = new Pato("Blanco");
        Gato isidoro = new Gato("Siamés");

        ArrayList<Mamifero> animales = new ArrayList();
        animales.add(sira);
        animales.add(nala);
        animales.add(donald);
        animales.add(isidoro);

        for(Mamifero animal:animales){
            if(animal instanceof Perro){
                ((Perro) animal).saludar();
            }else if(animal instanceof Pato){
                ((Pato) animal).croar();
            }else if(animal instanceof Gato){
                ((Gato) animal).maullar();
            }else{
                System.out.println("No conozco este animal");
            }
        }
    }
```

```
Output ×
Debugger Console × | Modulo3HerenciaPolimorfismo (run) ×
run:
Me llamo Sira y tengo 0 patas.
Me llamo null y tengo 0 patas.
Cuac! Cuac!
Miau!
```

Conclusión

La programación orientada a objetos es una programación con un amplio manejo para el diseño de aplicaciones y programas informáticos que usa objetos para sus interacciones. Está basada en varias técnicas, incluyendo herencia, abstracción, polimorfismo, atributos y encapsulamiento.

El objetivo principal del anterior trabajo es analizar y dejar claro lo que es la programación orientada a objetos, todo lo que lo rodea, sus grandes funciones. Después de la gran definición de los que es P.O.O. vendríamos definiendo todo lo que es sus atributos, clases, métodos, herencias, su polimorfismo, encapsulamiento, abstracción y también objetos.

Lo aprendido en la consulta

En la consulta anterior me dejo algo muy claro que los objetos se crean siempre y cuando tenga unas especificaciones o normas que defina como va a ser ese objeto, para esto P.O.O. se conoce como una clase.

La creación de una aplicación en P.O.O. se realiza mediante estructuras de objetos, que contienen unos procedimientos e información destinadas a cumplir sus tareas asignadas en las líneas códigos. La creación de un objeto a partir de una clase se conoce como instanciación de un objeto.

Bibliografía

http://aularagon.catedu.es/materialesaularagon2013/POO-Tecnologia/M3/objetos.html

https://www.lainter.edu.mx/blog/2018/03/18/programacion-orientada-a-objetos/

https://www.digitallearning.es/intro-programacion-js/objetos-propiedades-metodos.html

https://styde.net/abstraccion-programacion-orientada-a-objetos/