Fundamentos de Java

Programación Orientada a Objetos en Java

Por Ing. Eudris Cabrera Rodríguez

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento. Su uso se popularizó a principios de la década de 1990.

Actualmente son muchos los lenguajes de programación que soportan la orientación a objetos.

Introducción
Los objetos son entidades que combinan estado, comportamiento e identidad:

- El **estado** está compuesto de datos, será uno o varios atributos a los que se habrán asignado unos valores concretos (datos).
- El **comportamiento** está definido por los **procedimientos** o **métodos** con que puede operar dicho objeto, es decir, qué operaciones se pueden realizar con él.
- La **identidad** es una propiedad de un objeto que lo diferencia del resto, dicho con otras palabras, es su identificador (concepto análogo al de identificador de una variable o una constante).

La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

De esta forma, un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos.

A su vez, los objetos disponen de mecanismos de interacción llamados métodos que favorecen la comunicación entre ellos. Esta comunicación favorece a su vez el cambio de estado en los propios objetos.

Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan ni deben separarse el estado y el comportamiento.

Conceptos fundamentales
La programación orientada a objetos introduce nuevos conceptos, que superan y amplían

conceptos antiguos ya conocidos.

Entre ellos destacan los siguientes:

Clase:

Definición de las propiedades y comportamiento de un tipo de objeto concreto.

La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.

Objeto:

Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos).

Se corresponde con los objetos reales del mundo que nos rodea, o a objetos internos del sistema (del programa). Es una instancia a una clase.

Método:

Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer.

Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

Evento:

Un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente.

También se puede definir como evento, a la reacción que puede desencadenar un objeto, es decir la acción que genera.

Mensaje:

Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

Propiedad o atributo:

Contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.

Estado interno:

Es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos).

No es visible al programador que maneja una instancia de la clase.

Componentes de un objeto:

Atributos, identidad, relaciones y métodos.

Representación de un objeto:

Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

En comparación con un lenguaje imperativo, una **"variable"**, no es más que un contenedor interno del atributo del objeto o de un estado interno, así como la **"función"** es un procedimiento interno del método del objeto.

Java soporta tres características clave de la programación orientada a objetos (POO): **Encapsulación**, **Herencia** y **polimorfismo.**

Clases y Objetos

Definición de clases en Java.

Sintaxis básica de una clase Java:

```
<control de acceso> class <nombre clase>
{
<atributos>
<constructores>
<métodos>
}
```

Dónde control de acceso puede ser: **private**, **public** o **protected**.
Los controles de acceso, son tal y como están escritos (en idioma inglés).

Atributos

Sintaxis básica de un atributo:

```
<control de acceso> * <tipo> <nombre> [= <valor_inicial>];
```

Los atributos son variables propias del objeto, son datos encapsulados dentro de él e indican un comportamiento del mismo, en varios lenguajes pueden ser accesadas anteponiendo la palabra reservada **this**.

El control de acceso define si un atributo será privado (**private**) o público (**public**). Un atributo privado en una clase implica que sólo los métodos de esa clase (de ninguna otra) pueden usar (consultar o modificar) dichos atributos. Esto se traduce en que sólo los objetos que se creen, de tal clase, podrán trabajar con tales atributos.

Un atributo público, por el contrario, puede ser usado (consultado o modificado) por los métodos de cualquier clase, esto se traduce en que cualquier objeto (sin importar la clase que los define) podrá usar tales atributos.

Métodos

Sintaxis básica de un método:

```
<control de acceso> * <tipo_retorno> <nombre> (<argumento> *) {
<Sentencia> *
}
```

Un método es un procedimiento propio de un objeto, un método cumple una función y retorna (o no).

Es como una función, un concepto muy propio de los lenguajes de procedimientos , pero un método es propio de un objeto.

Acceso a miembros de objetos

La notación de punto : **<object>.<miembro>.**
Se utiliza para acceder a miembros de objetos, incluyendo atributos y métodos.

Ocultación de la información

Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase.

Los objetos pueden comunicarse entre sí mismos o con el usuario por interfaces definidas por el programador.

Esto significa que se deben crear interfaces (métodos setter y getter) para acceder a los datos internos.

El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado.

Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas.

Encapsulamiento

Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema.

Un objeto puede cumplir con ciertas funciones sin que se conozca su estructura interna, un programador puede crear objeto que resuelva determinado problema y alguien más puede usarlo sólo ocupándose de su implementación.

Un objeto bien estructurado puede ser portado a otra parte del diseño del software o a otro proyecto sin preocuparse de adaptarlo para ese determinado proyecto (reutilización).

Esto provee las siguientes ventajas:

1. Oculta los detalles de implementación de una clase.
2. Obliga al usuario a utilizar una interfaz para acceder a los datos.
3. Hace que el código sea más fácil de mantener.

Constructores

En Java, la creación de un objeto debe realizarse a través de constructores, éstos son un tipo especial de método que se deben definir como parte de la clase.

Un constructor posee las siguientes características:

- Es un método que lleva el mismo nombre de la clase.
- Se invoca una sola vez, cuando se crea un nuevo objeto.
- Puede usarse para inicializar los atributos (también llamadas variables de instancia) de un objeto. Sin embargo, puede no contener instrucción alguna, en este caso, su propósito es permitir la obtención de memoria para las variables de instancia.
- Puede incluir parámetros.
- Una clase puede incluir varios constructores con distintas definiciones de parámetros formales.

Sintaxis básica de un constructor:

```
public <nombre clase>([<lista de parámetros>])
{
[<sentencias>]
}
```

Un constructor debe ser definido como público, de manera que sea posible, para cualquier objeto, crear objetos de cualquiera otra clase (definida como pública).

Esto llevado al plano de las definiciones de clases, cualquier método que lo requiera podrá crear un objeto de la clase que sea, según las necesidades que se tengan en virtud del

problema a resolver.

Constructor por defecto

- Siempre hay al menos un constructor en cada clase.
- Si el programador no suministra ningún constructor, el constructor por defecto está presente de forma automática:
    - El constructor por defecto no tiene argumentos.
    - El cuerpo del constructor por defecto está vacía.
    - El valor predeterminado permite crear instancias de objetos con **new Xxx()** sin tener que definir un constructor.

Diferencia entre clases y Objetos

Una clase es un contenedor o envoltura de datos (atributos) y comportamientos (métodos), una estructura para crear objetos a partir de ellas, en la mayoría de los lenguajes una clase se define utilizando la palabra reservada ***class***.

Mientras que los objetos son instancias de una clase, son tipos de datos abstractos que cuentan con atributos y comportamientos propios, es la base de la programación orientada a objetos, y se puede pensar en ellos homologando a la vida cotidiana (o de eso se trata..), en la mayoría de los lenguajes para instanciar un objeto (crearlo, reservar memoria para el) se utiliza la palabra reservada ***new***

Estructura de un archivo de código fuente Java

Sintaxis básica de un archivo de código fuente de Java:

```
[<Declaración del paquete>]
<Declaración de importación>
<Definición de Clase>
```

Paquetes

Un paquete es un contenedor de clases que permite agrupar las distintas partes de un programa

cuya funcionalidad tienen elementos comunes.

Los paquetes ayudan a manejar grandes sistemas de software y pueden contener clases y sub-paquetes.

Definición y uso de paquete

Sintaxis básica de la sentencia de paquete es:

```
package <nombre paquete superiores>[.<paquetes hijos>]*;
```

Ejemplos:

```
package com.eudriscabrera.java.poo;
```

- Especificar la declaración del paquete al comienzo del archivo fuente.
- Sólo una declaración del paquete por archivo fuente.
- Si no se declara ningún paquete, entonces la clase se coloca en el paquete por defecto.
- Los nombres de paquetes deben ser jerárquicas y separados por puntos.

Declaración de importación

Sintaxis básica de la declaración de importación es:

```
import <nombre_paquete>[.<paquetes_hijos>.]*. <nombre_clase>;
```

O

```
import <nombre_paquete>[.<paquetes_hijos>]*.*;
```

Ejemplos:

```
import java.util.List;

import java.io.*;

import com.eudriscabrera.java.poo.*;
```

- La declaración de importación realiza lo siguiente:
  - Precede a todas las declaraciones de clases.
  - Le dice al compilador dónde encontrar clases.

Directorios y Paquetes

Los paquetes se almacenan en el árbol de directorios que contiene el nombre del paquete.

Principio de inicialización antes de uso

El compilador verificará que las variables locales han sido inicializadas antes de utilizarse.

Herencia

Un objeto puede extender las características de otro objeto. Una clase puede ser "hija" de otra clase y a la vez esa clase puede ser "padre" de otra, al derivar una clase de otra esta toma sus atributos y comportamientos (dependiendo del nivel de acceso).

Herencia simple de Java

Un objeto puede extender las características de otro objeto y de ningún otro, es decir, que solo puede heredar o tomar atributos de un solo padre o de una sola clase.

Lenguajes como Java, Ada y C# sólo permite herencia simple mientras que otros como C++, Python permite herencia múltiple. La cual consiste en que un objeto puede extender las características de uno o más objetos, es decir, puede tener varios padres.

Sintaxis de una clase Java es la siguiente:

```
<control de acceso> class <nombre> [extends <superclase>] {
<Declaración>
}
```

Las interfaces proporcionan los beneficios de la herencia múltiple en Java sin inconvenientes.

Control de Acceso

Cuando se crea una nueva clase en Java, se puede especificar el nivel de acceso que se quiere para las variables de instancia y los métodos definidos en la clase:

public

Cualquier clase desde cualquier lugar puede acceder a las variables y métodos de instancia públicos.

```
public void CualquieraPuedeAcceder(){}
```

protected

Sólo las subclases de la clase y nadie más puede acceder a las variables y métodos de instancia protegidos.

```
protected void SoloSubClases(){}
```

private

Las variables y métodos de instancia privados sólo pueden ser accedidos desde dentro de la clase. No son accesibles desde las subclases.

```
private String NumeroDelCarnetDeIdentidad;
```

```
friendly (sin declaración específica)
void MetodoDeMiPaquete(){}
```

Por defecto, si no se especifica el control de acceso, las variables y métodos de instancia se declaran ***friendly*** (amigas), lo que significa que son accesibles por todos los objetos dentro del mismo paquete, pero no por los externos al paquete. Es lo mismo que ***protected***.

Los métodos protegidos (***protected***) pueden ser vistos por las clases derivadas, como en C++, y también, en Java, por los paquetes (packages). Todas las clases de un paquete pueden ver los métodos protegidos de ese paquete.

Para evitarlo, se deben declarar como ***private protected*** , lo que hace que ya funcione como en C++ en donde sólo se puede acceder a las variables y métodos protegidos de las clases derivadas.

Sobreescritura de métodos

Una subclase puede modificar el comportamiento heredado de una clase padre cuando define un método con las mismas características (Nombre,Tipo de retorno, Lista de argumentos).

Las subclases emplean la sobreescritura de métodos la mayoría de las veces para agregar o modificar la funcionalidad del método heredado de la clase padre.

Invocación de métodos sobreescrito

Un método subclase puede invocar un método de una clase padre usando la palabra reservada **super**:

- La palabra reservada super se utiliza en una clase para hacer referencia a su superclase.
- La palabra reservada super se utiliza para referirse a los miembros de superclase, tanto los datos de atributos y métodos.
- Comportamiento invocado no tiene que estar en la superclase; puede ser más arriba en la jerarquía.

Polimorfismo

El polimorfismo se refiere a la posibilidad de definir múltiples clases con funcionalidad diferente, pero con métodos o propiedades denominados de forma idéntica, que pueden utilizarse de manera intercambiable mediante código cliente en tiempo de ejecución.

El operador instanceof

Utilice **instanceof** para comprobar el tipo de un objeto.

Ejemplo:

```
if (objeto instanceof type){ }
```

Sobrecarga de Métodos

La sobrecarga de métodos es la creación de varios métodos con el mismo nombre pero con diferentes firmas y definiciones. Java utiliza el número y tipo de argumentos para seleccionar cuál definición de método ejecutar.

Java diferencia los métodos sobrecargados con base en el número y tipo de argumentos que tiene el método y no por el tipo que devuelve.

También existe la sobrecarga de constructores: Cuando en una clase existen constructores múltiples, se dice que hay sobrecarga de constructores.

Utilice la sobrecarga de la siguiente manera:

```
public void println(int i)
public void println(float f)
public void println(String s)
```

Las listas de argumentos deben ser diferentes y los tipos de retorno puede ser diferente.

Métodos que reciben argumentos variables

Los métodos que utilizan argumentos variables permiten múltiples número de argumentos en los métodos.

Por ejemplo:

```
public class Statistics {
public float average(int... nums) {
        int sum = 0;
        for ( int x : nums ) {
```

```
            sum += x;
        }
        return ((float) sum) / nums.length;
    }
}
```

El parámetro **vararg** es tratado como un arreglo.

Por ejemplo:

```
float gradePointAverage = stats.average(4, 3, 4);
float averageAge = stats.average(24, 32, 27, 18);
```

Sobrecarga de constructores

Al igual que con los métodos, los constructores pueden ser sobrecargados.

Ejemplo:

```
public Empleado(String nombre, double salario, Date fechaNacimiento)
public Empleado(String nombre, double salario)
public Empleado(String nombre, Date fechaNacimiento)
```

- Las listas de argumentos deben ser diferentes.
- Puede utilizar la referencia ***this*** en la primera línea de un constructor para llamar a otro constructor.

Constructores y herencia

- Los constructores no se heredan.
- Una subclase hereda todos los métodos y variables de la super clase (clase padre).
- Una subclase no hereda el constructor de la superclase.
- Dos formas de incluir un constructor son:
- Utilice el constructor por defecto.
- Escribir un o varios constructores más explícitas.

Invocación de constructores de una clase padre

- Para llamar a un constructor principal, debe realizar una llamada a ***super*** en la primera línea del constructor.

- Puede llamar a un constructor específico del padre por parte del argumentos que se utilizan en la llamada a ***super***.
- Si no se utiliza esta llamada o super en un constructor, a continuación, el compilador añade una llamada implícita a ***super()*** que llama el constructor del padre sin ningún argumento (que podría ser el constructor por defecto).
- Si la clase padre define constructores, pero no proporciona un constructor sin argumentos, a continuación, el compilador dará error.

Variables y métodos estáticos

La palabra reservada static se usa como un modificador de variables, métodos y clases anidadas.

En un momento determinado se puede querer crear una clase en la que el valor de una variable de instancia sea el mismo (y de hecho sea la misma variable) para todos los objetos instanciados a partir de esa clase.

Los miembros estáticos son a menudo llamados miembros de la clase, tales como atributos de clase o métodos de clase.

Al igual que las variables, los métodos se pueden declarar como ***static*** y poseen una caracteristica especial, que se pueden llamar sin ninguna instancia de la la clase a la que pertenece.

Inicializadores estáticos

- Una clase puede contener código en un bloque estático que no lo hace existir dentro de un cuerpo de método.
- Código de bloque estático se ejecuta sólo una vez, cuando la clase es cargado.
- Por lo general, un bloque estático se utiliza para inicializar estática (clase) atributos.

Uso de final

- No se puede crear una subclase de una clase final.
- No se puede reemplazar un método final.
- Una variable final es una constante.
- Puede asignar valores a una variable final sólo una vez, pero la asignación puede ocurrir independientemente de la declaración;esto se llama una variable final en blanco.
    - Un atributo final en blanco debe tener una asignación en cada constructor.
    - Una variable final en blanco de un método debe fijarse en el cuerpo del método antes de ser utilizados.

Uso de this y super

Al acceder a variables de instancia de una clase, la palabra reservada ***this*** hace referencia a

los miembros de la propia clase.

Si se necesita llamar al método padre dentro de una clase que ha reemplazado ese método, se puede hacer referencia al método padre con la palabra clave ***super***.

Clases Abstractas

Una clase abstracta es una clase de la cual no se pueden instanciar objetos, de estas clases sólo se pueden heredar otras, en desarrollo se utilizan simplemente para derivar a otras clases más especializadas.

En los lenguajes que implementan este tipo de clases se utiliza la palabra reservada ***abstract*** para especificarlas.

Una clase abstracta debe tener uno o más métodos abstractos, y estos métodos deben ser implementados en las clases derivadas.

```
public abstract class Instrumento{

public Instrumento(){}

public abstract void tocar();

public abstract void tocar(String nota);

public abstract void afinar();

}
```

Interfaces

Una interfaz es una clase únicamente para implementación en otras, en una interfaz se indican los prototipos de los métodos que debería tener un objeto pero el cuerpo del mismo debe ser escrito en las clases que la implementan, en los lenguajes que lo implementan se utiliza la palabra reservada **interface** para indicar una interfaz e ***implements*** para su implementación.

Sintaxis de una clase Java es la siguiente:

```
<control de acceso> class <nombre> [extends <clase padre>]
[implements <interface> [,<interface>]* ] {
<Declaración>*
}
```

Usos de Interfaces

Los usos son los siguientes:

- Declaración de métodos que una o más clases esperan implementar.
- Determinación de la interfaz de programación de un objeto sin revelar el cuerpo real de la clase.
- Captura de similitudes entre clases no relacionadas sin forzar una relación de clase.
- La simulación de la herencia múltiple por la que se declara una clase que implementa varias interfaces.

Diferencia entre Clase Abstracta e Interface

Clases Abstractas	**Interfaces**
Contiene tanto métodos ejecutables como métodos abstractos.	No tiene código de implementación. Todos sus métodos son abstractos.
Una clase solo puede extender de una única clase abstracta.	Una clase puede implementar n número de interfaces.
Puede tener variables de instancia, constructores y cualquiera de los tipos de visibilidad: public, private, protected, ninguno (aka package).	No puede tener variables de instancia o constructores y solo puede tener métodos públicos o package.

Referencias:

http://www.javapassion.com/
http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java
http://es.wikipedia.org/wiki/Orientado_a_objetos
https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html

Eudris Cabrera Rodríguez

Ingeniero Telemático

Consultor / Desarrollador Informático

LinkedIn: http://www.linkedin.com/in/eudriscabrera

Revisado Noviembre 2018, Santiago de los Caballeros, R. D.