TALENTO DIGITAL
INTELIGENCIA HUMANA

Talento Digital para Chile:

MÓDULO 1
PROGRAMACIÓN BÁSICA EN JAVA

UN PROYECTO DE:

Ministerio de Hacienda · Ministerio del Trabajo y Previsión Social · CORFO · sence · SOFOFA · OTIC SOFOFA Capital Humano · CPC · ACTI · BID · FCH Fundación Chile · kodea

MÓDULO 1 - PROGRAMACIÓN BÁSICA EN JAVA

1.3 EL PARADIGMA DE ORIENTACIÓN A OBJETOS

Semana 3 - Día 14

Objetivo de la jornada

- Comprender los conceptos de Modificadores de acceso y su utilidad en la Programación Orientada a Obj
- Entender y aplicar los conceptos de Accesores y Mutadores para dar solución a un problema según requ establecidos.
- Entender la colaboración entre objetos para luego utilizarlos para resolver problemas planteados.

Modificadores de acceso

Las clases tienen ciertos privilegios de acceso a los datos y a las funciones de otras clases dentro de un mism paquetes son una forma de organizar grupos de clases, contienen un conjunto de clases relacionadas. Los paqu el problema del conflicto entre los nombres de las clases. Al crecer el número de clases crece la probabilidad d el mismo nombre a dos clases diferentes. En Java, los paquetes se visualizan de la siguiente manera:

```
package nombrePaquete;


class Auto { }
```

Para importar clases de un paquete se usa el comando import. En Java se puede importar u individual al inicio de todo el código de la siguiente forma:

```
import java.util; package nombrePaquete;

class Auto { }
```

Los modificadores de acceso determinan desde qué clases se puede acceder a un determinado elemento.

En Java existen 4 tipos de modificadores de acceso: public, private, protected y el tipo por **default o por defecto**: Si no se especifica ningún modificador de acceso se utiliza el nivel de acceso por defecto, que consiste en que el elemento puede ser accedido sólo desde las clases que pertenezcan al mismo paquete. Para este modificador de acceso, no se le debe agregar ninguna palabra reservada como en los otros casos.

public: Permite acceder al elemento desde cualquier clase, independientemente de que esta pertenezca o no al paquete en que se encuentra el elemento.

private: Es el modificador más restrictivo y especifica que los elementos que lo utilizan sólo pueden ser accedidos desde la clase en la que se encuentran.

protected: indica que los elementos sólo pueden ser accedidos desde su mismo paquete (como el acceso por defecto) y desde cualquier clase que extienda la clase en que se encuentra, ndependientemente de si esta se encuentra en el mismo paquete o no.

Los distintos modificadores de acceso quedan resumidos en la siguiente tabla:

Modificadores de acceso				
public	La misma clase	Otra clase del mismo paquete	Subclase de otro paquete	Otra clase de otro paquete
default	X	X		
public	X	X	X	X
private	X			
protected	X	X	X	

Continuando con el ejemplo de la clase Auto, los modificadores de acceso de atributos y métodos quedarían de la siguiente forma:

```
class Auto {
    private String marca; private String color; private float
    velocidad;

    public void acelerar() {
     velocidad++; } public void frenar() { velocidad = 0; }
                            public Auto() {
    } public Auto(String marca, String color, float velocidad) {
this.marca = marca; this.color = color; this.velocidad = velocidad; } }
```

Los atributos poseen el modificador de acceso private ya que nos interesa salvaguardar los datos de nuestros objetos. Por otro lado, los métodos son públicos para que pueda existir la comunicación entre distintas clases.

Accesor y Mutadores

Si se observa el ejemplo anterior de la clase Auto, sus atributos son privados, por lo que otras clases no podrán acceder a los valores de sus atributos. Es por esto que se necesitan métodos específicos para cumplir con esta misión.

El papel de los accesores y mutadores es devolver y establecer los valores del estado de un objeto.

Quizás la conclusión más lógica sería cambiar los campos privados de la definición de la clase para que sean públicos y lograr los mismos resultados pero es importante recordar que lo que se desea es ocultar los datos del objeto tanto como sea posible.

Accesor

Se utiliza un método accesor para **devolver** el valor de un campo privado. Sigue un esquema de nombres que prefija la palabra "get" al principio del nombre del método.

Por ejemplo, se agregaran métodos accesores para los atributos marca, color y velocidad de la clase Auto:

```
class Auto {
    private String marca; private String color; private float
    velocidad;

    public void acelerar() {
    velocidad++; } public void frenar() { velocidad = 0; }
    public Auto() { } public Auto(String marca, String color, float velocidad) {
    this.marca = marca; this.color = color; this.velocidad =
    velocidad; } public String getMarca() {
    return this.marca; } public String getColor() {
    return this.color; } public float getVelocidad() {
    return this.velocidad; } }
```

Estos métodos siempre devuelven el mismo tipo de datos que su campo privado correspondiente y luego simplemente devuelven el valor de ese campo privado.

Mutador

Se utiliza un método de mutador para **establecer** el valor de un campo privado. Sigue un esquema de nombres con el prefijo "set" al comienzo del nombre del método. Se agregarán los métodos mutadores para la marca, color y velocidad de la clase Auto:

```
class Auto {
    private String marca; private String color; private float
    velocidad;

    public void acelerar() {

      velocidad++; } public void frenar() { velocidad = 0;

    } public Auto() { } public Auto(String marca, String color, float velocidad) {
    this.marca = marca; this.color = color; this.velocidad =
    velocidad; } public String getMarca() {
    return this.marca; } public String getColor() {
    return this.color; } public float getVelocidad() {
    return this.velocidad; } public void setMarca(String marca) {
    this.marca = marca; } public void setColor(String color) {
    this.color = color; } public void setVelocidad(float velocidad) {
    return this.velocidad; } }
```

Estos métodos no tienen un tipo de retorno y aceptan un parámetro que es el mismo tipo de datos que su campo privado correspondiente. El parámetro se utiliza para establecer el valor de ese campo privado.

Colaboración entre objetos

Es común que para resolver un problema de mediana complejidad utilizando la POO se utilice más de una clase, las cuales deben interactuar y comunicarse a través de los métodos.

Dentro de una aplicación Java, los objetos pueden estar conectados dentro de un programa con distintos tipos de relaciones. Estas relaciones puede ser persistentes si establecen si la comunicación entre objetos se registra de algún modo y por tanto puede ser utilizada en cualquier momento. En el caso de relaciones no persistentes entonces el vínculo entre objetos desaparece tras ser empleado.

Esta colaboración se puede llevar a cabo mediante el establecimiento de relaciones entre clases o entre instancias (relación de asociación y relación todo-parte: agregación y composición).

Asociación

En una asociación, dos instancias A y B relacionadas entre sí existen de forma independiente. No hay una relación fuerte. La creación o desaparición de uno de ellos implica únicamente la creación o destrucción de la relación entre ellos y nunca la creación o destrucción del otro. Por ejemplo, un cliente puede tener varios pedidos de compra o ninguno.

La relación de asociación expresa una relación, de una dirección o bidireccional, entre las instancias a partir de las clases conectadas. El sentido en que se recorre la asociación se denomina navegabilidad de la asociación. Cada extremo de la asociación se caracteriza por el rol o papel que juega en dicha relación el objeto situado en cada extremo.
La cardinalidad o multiplicidad es el número mínimo y máximo de instancias que pueden relacionarse con la otra instancia del extremo opuesto de la relación. El formato en el que se escribe es mínima.. máxima, por ejemplo:

- **1**: Sólo uno (por defecto).
- **0..1**: Cero a uno.
- **N..M**: Desde N hasta M (enteros naturales).
- **0..***: Cero a muchos.
- **1..***: Uno a muchos.
- **1,5,9**: Uno o cinco o nueve.

Todo-Parte

En una relación todo-parte una instancia forma parte de otra. En la vida real se dice que A está compuesto de B o que A tiene B. La diferencia entre asociación y relación todo-parte radica en la asimetría presente en toda relación todo-parte. En teoría se distingue entre dos tipos de relación todo-parte:

Agregación

La agregación es una asociación binaria que representa una relación todo-parte (pertenece a tiene un, es parte de). Por ejemplo, un centro comercial tiene clientes.

La composición es una agregación fuerte en la que una instancia 'parte' está relacionada, como máximo, con una instancia 'todo' en un momento dado, de forma que cuando un objeto 'todo' es eliminado, también son eliminados sus objetos 'parte'.

Por ejemplo: un rectángulo tiene cuatro
vértices, un centro comercial está organizado mediante un conjunto de secciones de venta...

A nivel práctico se suele llamar agregación cuando la relación se plasma mediante referencias (lo que permite que un componente esté referenciado en más de un compuesto). Así, a nivel de implementación una agregación no se diferencia de una asociación binaria. Por ejemplo: un equipo y sus miembros. Por otro lado, se suele llamar composición cuando la relación se conforma en una inclusión por valor (lo que implica que un componente está como mucho en un compuesto, pero no

impide que haya objetos componentes no relacionados con ningún compuesto).
En este caso si se destruye el compuesto se destruyen sus componentes.

Por ejemplo: un ser humano y sus miembros. Algunas relaciones pueden ser consideradas agregaciones o composiciones, en función del contexto en que se utilicen.

Composición

Es una forma fuerte de composición donde la vida de la clase contenida debe coincidir con la vida de la clase contenedor. Los componentes constituyen una parte del objeto compuesto, de esta forma, los componentes no pueden ser compartidos por varios objetos compuestos.
La supresión del objeto compuesto conlleva la supresión de los componentes.

Un ejemplo de composición:

Una empresa contiene muchos empleados. Un objeto Empresa está a su vez compuesto por uno o varios objetos del tipo empleado. El tiempo de vida de los objetos Empleado depende del tiempo de vida de Empresa, ya que si no existe una Empresa no pueden existir sus empleados.

La composición es un tipo de relación dependiente en dónde un objeto más complejo es conformado por objetos más pequeños. En esta situación, la frase “Tiene un”, debe tener sentido: El auto tiene llantas o el notebook tiene un teclado.