Apunte Algoritmos y Programación III - FIUBA

lcondoriz

Agosto 2023

Índice

1. Introducción

Apunte de la Materia Algoritmos y programación III.

2. Programación Orientada a Objetos

Definición 2.1. Paradigmas de Programación

- **Imperativos:** (énfasis en la ejecución de instrucciones)
  - Programación Procedimental (p. ej. Pascal).
  - Programación Orientada a Objetos (p. ej. Smalltalk, Java = multiparadigma).
- **Declarativos:** (énfasis en la evaluación de expresiones)
  - Programación Funcional (p. ej. Haskell).
  - Programación Lógica (p. ej. Prolog).

2.1. Sistemas Orientados a Objetos

Definición 2.2. (Entidad) Una entidad es un objeto del mundo real que tiene un identificador único, un estado y un comportamiento [2].

Tipos de entidades:

- **Entidades físicas:** (p. ej. un auto, una persona, un libro).
- **Entidades conceptuales:** (p. ej. un viaje, una reserva, un préstamo).
- **Entidades fuertes:** son entidades que pueden sobrevivir por sí solas.
- **Entidades debil:** no pueden existir sin una entidad fuerte y se representan con un cuadrado con doble línea

Definición 2.3. (Clase) Una clase es un conjunto de objetos que comparten características y comportamientos comunes, así como propiedades y atributos comunes. Es un prototipo o plano definido por el usuario a partir del cual se crean objetos

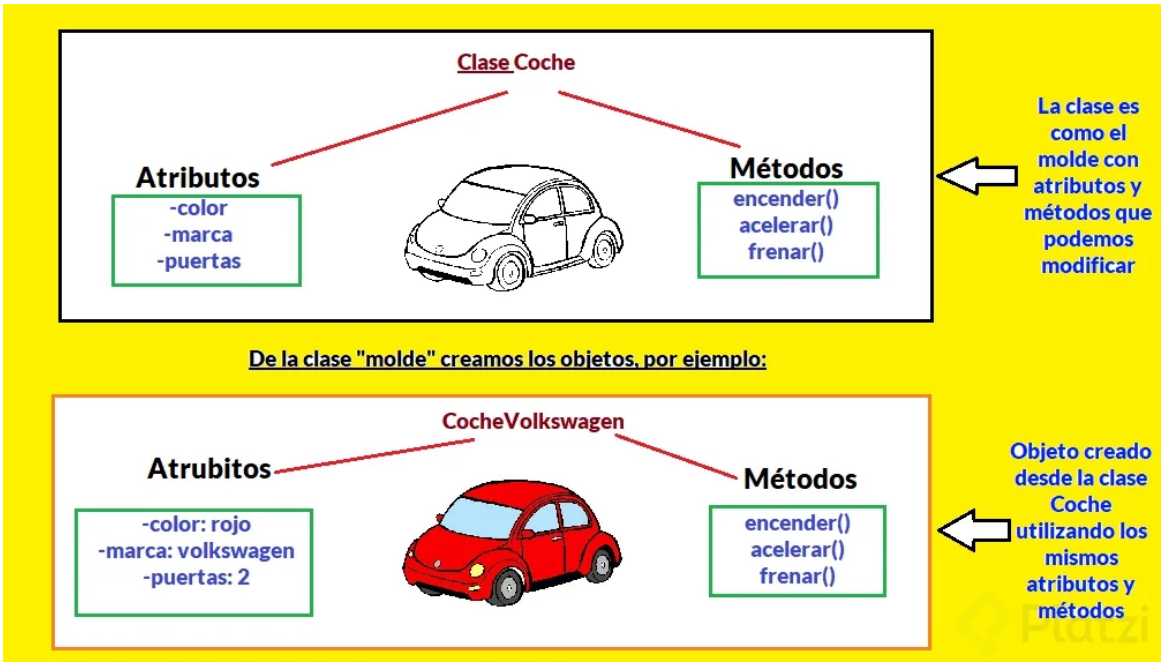

Figura 1: Clase

Definición 2.4. (Objeto) Un objeto es una entidad que puede recibir mensajes, responder a los mismos y enviar mensajes a otros objetos. Un objeto es una entidad que tiene comportamiento.

Todo objeto tiene tres características:

- **Identidad:** Es un identificador único que lo diferencia de los demás objetos. La identidad es lo que distingue a un objeto de otro.
- **Estado:** El estado es la situación en que un objeto se encuentra. Un objeto puede cambiar su estado a través del tiempo.
- **Comportamiento:** Es el conjunto de posibles respuestas de un objeto ante los mensajes que recibe. El comportamiento de un objeto está compuesto por las respuestas a los mensajes que recibe un objeto, que a su vez pueden provocar:
  - Un cambio de estado en el objeto receptor del mensaje.
  - La devolución del estado de un objeto, en su totalidad o parcialmente
  - El envío de un mensaje desde el objeto receptor a otro objeto (delegación)

Cuando un objeto es creado a partir de una clase, se dice que el objeto es una **instancia** de la clase, esto se puede ver en la Figura 1, creación del objeto CocheVolkswagen.

Definición 2.5. (Atributo (o variable de instancia)) En POO, llamamos atributo a una variable interna del objeto que sirve para almacenar parte del estado del mismo

Definición 2.6. (Método) Llamamos método a la implementación de la respuesta de un objeto a un mensaje. En términos de implementación, se asemeja a funciones o procedimientos de programación en otros paradigmas.

Definición 2.7. (Interfaz) Al conjunto de las firmas de los métodos se lo suele llamar interfaz o protocolo del objeto. La interfaz de un objeto es el conjunto de mensajes a los que puede responder.

Definición 2.8. (Mensaje, cliente y receptor) Un mensaje es la interacción entre un objeto que pide un servicio y otro que lo brinda.
El objeto que envía el mensaje se llama objeto cliente y quien recibe el mensaje se llama objeto receptor.

Definición 2.9. (Delegación) Cuando un objeto, para responder un mensaje, envía mensajes a otros objetos, decimos que delega ese comportamiento en otros objetos. También es la colaboración de otros objetos para poder responder un mensaje.

Definición 2.10. (Encapsulamiento) Cada objeto es responsable de responder a los mensajes que recibe, sin que quien le envía el mensaje tenga que saber cómo lo hace. Esto es lo que llamamos encapsulamiento.

"Tell, don't ask", implica que los objetos deben manejar su propio comportamiento, sin que nosotros manipulemos su estado desde afuera.

Definición 2.11. (Polimorfismo) El polimorfismo es la capacidad de respuesta que tienen distintos objetos de responder de maneras diferentes a un mismo mensaje.

2.1.1. Pasos para resolver un problema

1. **Encontrar objetos:** (entidades del dominio del problema) Comenzar con el diseño del modelo.
2. **Determinar los mensajes:** (cómo deben interactuar los objetos)
   a) Diagrama de secuencia UML, como colaboran los objetos.
3. **implementar el comportamiento de los objetos:**

2.1.2. Relaciones estáticas

Son los niveles de visibilidad que controlan el acceso a los atributos y métodos de una clase. min 46 Youtube

- **Público:** (public) se puede acceder desde cualquier clase.
- **Privado:** (private) solo se puede acceder desde la misma clase.
- **Protegido:** (protected) se puede acceder desde la misma clase y desde las subclases.

2.2. Diseño por contrato y un procedimiento constructivo

La idea primigenia del diseño por contrato es, entonces, que un objeto servidor brinda servicios a objetos clientes sobre la base de un contrato que ambos se comprometen a cumplir.

Definición 2.12. (Modelo) Un modelo es una representación simplificada de la realidad, que nos permite entenderla y manipularla.

Definición 2.13. (Abstracción) Es una simplificación que incluye solo aquellos detalles relevantes para determinado propósito y descarta los demás.

2.2.1. Precondiciones

Las precondiciones expresan en qué estado debe estar el medio ambiente antes de que un objeto cliente le envíe un mensaje a un receptor. En general, el medio ambiente está compuesto por el objeto receptor, el objeto cliente y los parámetros del mensaje, pero hay ocasiones en que hay que tener en cuenta el estado de otros objetos.

Ante el incuplimiento de una precondición se lanza una excepción.

Definición 2.14. (Excepción) Una excepción es un objeto que el receptor de un mensaje envía a su cliente como aviso de que el propio cliente no está cumpliendo con alguna precondición de ese mensaje.

2.2.2. Postcondiciones

El conjunto de postcondiciones expresa el estado en que debe quedar el medio como consecuencia de la ejecución de un método. En términos operativos, es la respuesta ante la recepción del mensaje.

El cumplimiento de las postcondiciones es responsabilidad del receptor. Si una postcondición no se cumple se debe a que el método está mal programado por quien deba implementar el objeto receptor.

Definición 2.15. (Prueba unitaria) Una prueba unitaria es aquélla prueba que comprueba la corrección de una única responsabilidad de un método.

Corolario: Deberíamos tener al menos una prueba unitaria por cada postcondición.

2.2.3. Invariantes

Los invariantes son condiciones que debe cumplir un objeto durante toda su existencia.

Los invariantes suelen estar presentes a través de precondiciones o postcondiciones.

2.3. Pruebas unitarias

2.3.1. Desarrollo empezando por las pruebas

2.4. colaboración entre objetos

Veremos delegación y programación por diferencia, cuestiones de comportamiento que se apoyan en los aspectos estructurales llamados asociación y herencia.

2.5. Pilares de la POO

1. **Abstracción:** Es la capacidad de representar un objeto del mundo real en un programa. Es el proceso de identificar las características esenciales de un objeto y eliminar las características no esenciales.
2. **Encapsulamiento:** Es la capacidad de ocultar los detalles de implementación de un objeto.
3. **Polimorfismo:** Es la capacidad de enviar mensajes sintácticamente iguales a objetos de distintas clases y que estos respondan de manera diferente.
4. **Herencia:** Es la capacidad de definir nuevas clases a partir de otras ya existentes.

2.6. Herencia

Definición 2.16. (Herencia) La herencia es un mecanismo que permite definir nuevas clases a partir de otras ya existentes. La herencia permite definir una clase a partir de otra, reutilizando sus atributos y comportamientos. La clase de la que se hereda se llama *superclase* y la clase que hereda se llama *subclase*.

La herencia es una relación entre clases, por la cual se define que una clase puede ser un caso particular de otra. A la clase más general la llamamos madre y a la más patricular hija.

Corolario: Cuando hay herencia, todas las instancias de la clase hija son también instancias de la clase madre.

Definición 2.17. (Programación por diferencia) La programación por diferencia es un procedimiento constructivo que consiste en definir una clase a partir de otra, agregando o modificando atributos y comportamientos.

Programamos por diferencia cuando indicamos que parte de la implementación de un objeto está definida en otro objeto, y por lo tanto sólo implementamos las diferencias específicas.

2.7. Clases

Tipos de clases:

1. **Clases abstractas:** Son clases que no pueden ser instanciadas, es decir, no pueden tener objetos. Son clases que sirven para definir un comportamiento común a varias clases.
2. **Clases concreta:** Son clases que pueden ser instanciadas, es decir, pueden tener objetos.

Listing 1: Clase abstracta

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, World!");
4     }
5 }
```

3. Los objetos colaboran

Apunte de la Materia Algoritmos y programación III.

4. Java

4.1. Tipos de clases en JAVA

- **Clases**: Son las clases comunes y corrientes, que se pueden instanciar.
- **Clases Abstractas**: Son clases que no se pueden instanciar, pero que sirven para heredar.
- **Interfaces**: Son clases que no se pueden instanciar, pero que sirven para heredar.
- **Clases Anónimas**: Son clases que no tienen nombre, y se usan para sobreescribir métodos.
- **Clases finales**: Son clases que no se pueden heredar.

4.2. Relaciones entre clases

Relaciones en UML: [1]

- **Asociación**
  - **Agregación**
  - **Composición**
- **Herencia/Generalización**
- **Dependencia**

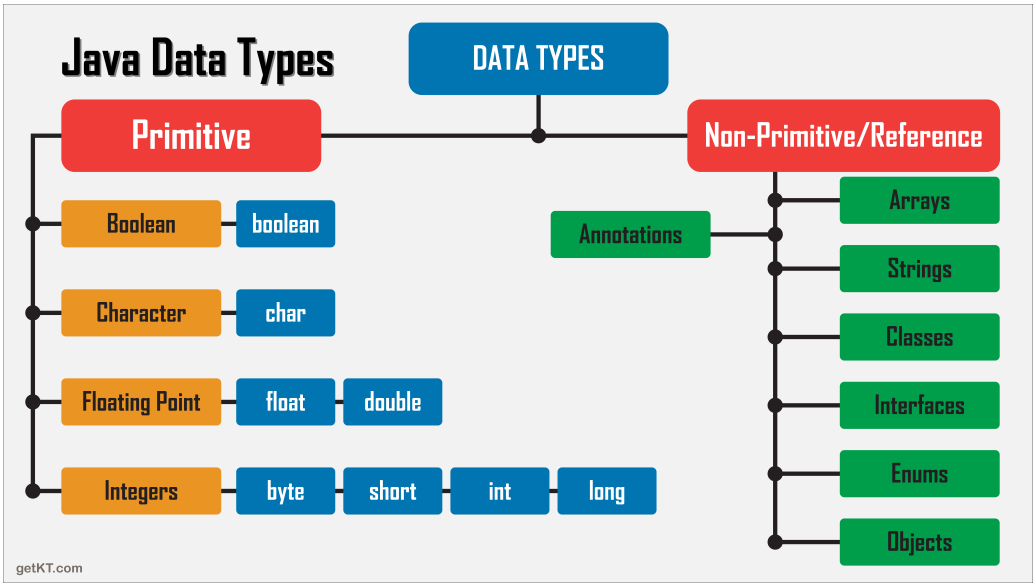

Figura 2: Tipos de datos en Java [4]

- **Realización/Implementación**

1. **Asociación:** Indica que una propiedad de una clase contiene una referencia a una instancia (o instancias) de otra clase.

   La asociación es la relación más utilizada entre una clase y otra clase, lo que significa que existe una conexión entre un tipo de objeto y otro tipo de objeto. Las combinaciones y agregaciones también pertenecen a las relaciones asociativas , pero las relaciones entre clases de afiliaciones son más débiles que las otras dos.

   Hay cuatro tipos de asociaciones : asociaciones bidireccionales , asociaciones unidireccionales , autoasociación y asociaciones de números múltiples.

   Por ejemplo: coches y conductores, un coche corresponde a un conductor en particular y un conductor puede conducir varios coches. El * en al fugura significa 0 o más.

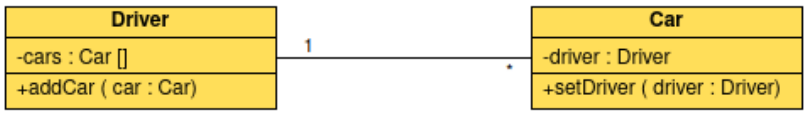

Figura 3: Asociación

   a) **Agregación:** La relación entre el todo y la parte, y el todo y la parte se pueden separar. Las relaciones agregadas también representan la relación entre el todo y una parte de la clase, los objetos miembros son parte del objeto general, pero el objeto miembro puede existir independientemente del objeto general.Tiempo de vida *independiente*.

   Por ejemplo, los conductores de autobús y la ropa y los sombreros de trabajo son parte de la relación general, pero se pueden separar. La ropa de trabajo y los sombreros se pueden usar en otros conductores. Los conductores de autobuses también pueden usar otra ropa de trabajo y sombreros.
   Otro ejemplo puede ser el de un auto y sus ruedas, el auto depende si o si de tener cuatro ruedas.

   b) **Composición:** La relación entre el todo y la parte, pero el todo y la parte no se pueden separar.
   La relación de combinación representa la relación entre el todo y la parte de la clase, y el total y la parte tienen una duración constante. Una vez que el objeto general no existe, algunos de los objetos no existirán y todos morirán en la misma vida. Tiempo de vida *dependiente*.

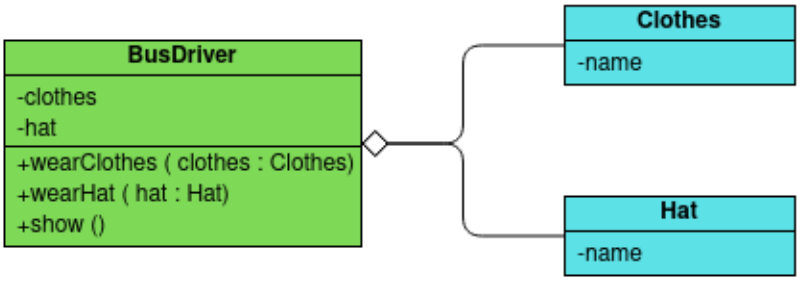

Figura 4: Agregación

Por ejemplo, una persona está compuesta por una cabeza y un cuerpo. Los dos son inseparables y coexisten.

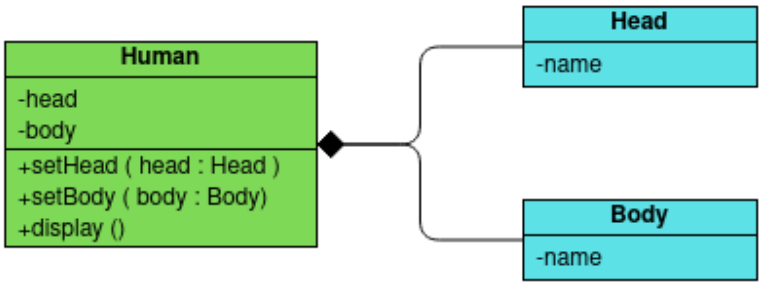

Figura 5: Composición

2. **Herencia/Generalización:** La herencia también se denomina generalización y se utiliza para describir la relación entre las clases padre e hijo. Una clase principal también se denomina clase base y una subclase también se denomina clase derivada.

   En la relación de herencia, la subclase hereda todas las funciones de la clase principal y la clase principal tiene todos los atributos, métodos y subclases. Las subclases contienen información adicional además de la misma información que la clase principal.
   a

   Por ejemplo: autobuses, taxis y automóviles son automóviles, todos tienen nombres y todos pueden estar en la carretera.

3. **Dependencia:** Suponga que un cambio en la clase A provoca un cambio en la clase B, luego diga que la clase B depende de la clase A. En la mayoría de los casos, las dependencias se reflejan en los métodos de una clase que utilizan el objeto de otra clase como parámetro . Una relación de dependencia es una relación de "uso". Un cambio en una cosa en particular puede afectar a otras cosas que la usan, y usar una dependencia cuando es necesario indicar que una cosa usa otra.

   Por ejemplo: El auto depende de la gasolina. Si no hay gasolina, el automóvil no podrá conducir.

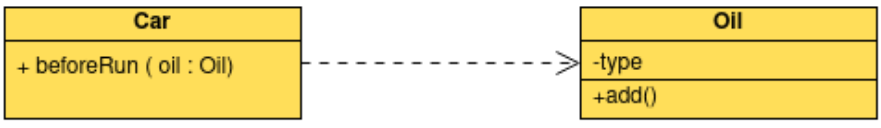

Figura 6: Dependencia

4. **Realización/Implementación:** La implementación (Implementación) se utiliza principalmente para especificar la relación entre las interfaces y las clases de implementación.

   Una interfaz (incluida una clase abstracta) es una colección de métodos. En una relación de implementación, una clase implementa una interfaz y los métodos de la clase implementan todos los métodos de la declaración de la interfaz.

   Por ejemplo: los automóviles y los barcos son vehículos, y el vehículo es solo un concepto abstracto de una herramienta móvil, y el barco y el vehículo realizan las funciones móviles específicas.

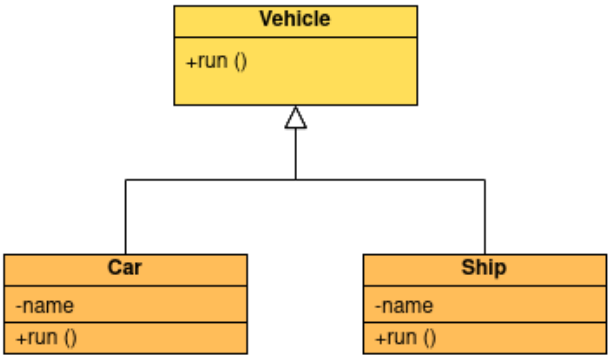

Figura 7: Realización/Implementación

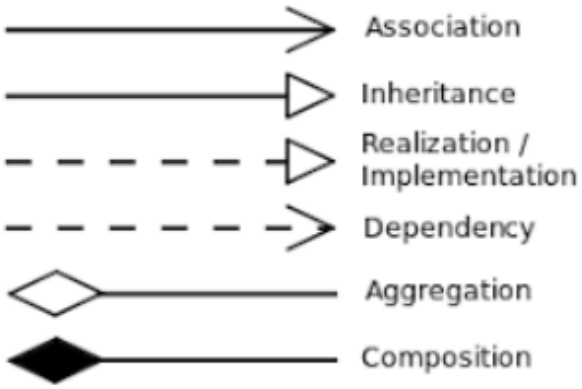

Figura 8: UML Relaciones entre clases

- **Asociación:** Es una relación de tipo **"tiene un"** entre clases, donde una clase tiene una referencia a otra clase. Por ejemplo, si tenemos una clase Persona y una clase Coche, podríamos decir que una persona tiene un coche, y representar esta relación con una asociación entre las clases Persona y Coche.
  - **Agregación:** Es un tipo especial de asociación que representa una relación de tipo **"parte de"** entre clases, donde una clase es parte de otra clase. A diferencia de la asociación, en la agregación las partes pueden existir independientemente del todo. Por ejemplo, si tenemos una clase Equipo y una clase Jugador, podríamos decir que un jugador es parte de un equipo, y representar esta relación con una agregación entre las clases Equipo y Jugador.
  - **Composición:** Es otro tipo especial de asociación que también representa una relación de tipo **"parte de"** entre clases, pero en este caso las partes no pueden existir independientemente del todo. Por ejemplo, si tenemos una clase Casa y una clase Habitacion, podríamos decir que una habitación es parte de una casa, y representar esta relación con una composición entre las clases Casa y Habitacion.
- **Herencia:** Es una relación de tipo **"es un"** entre clases, donde una subclase hereda las propiedades y comportamientos de su superclase y puede agregar o modificar propiedades y comportamientos propios. Por ejemplo, si tenemos una clase Vehiculo con propiedades como marca, modelo y color, y comportamientos como acelerar y frenar, podríamos crear una subclase Coche que herede estas propiedades y comportamientos de la clase Vehiculo, y agregar propiedades específicas como numeroDePuertas y comportamientos específicos como abrirTechoSolar.
- **Dependencia:** Es una relación entre clases donde una clase depende de otra clase para su funcionamiento. Por ejemplo, si tenemos una clase Calculadora y una clase Operacion, podríamos decir que la calculadora depende de la operación para realizar cálculos, y representar esta relación con una dependencia entre las clases Calculadora y Operacion.
- **Realización/Implementación:** Es una relación de tipo **"se comporta como"** entre clases, donde una clase implementa una interfaz y debe proporcionar implementaciones para todos los métodos definidos en la interfaz. Por ejemplo, si tenemos una interfaz Volador con métodos como despegar y aterrizar, podríamos tener clases como Avion y Pajaro que implementen esta interfaz y proporcionen implementaciones para los métodos despegar y aterrizar.

4.3. Interface

Interfaces: [3].

- Son una **colección de métodos abstractos** con propiedades (atributos) **constantes**.
- Una interfaz **solamente puede extender o implemtar otras interfaces** (la cabtidad que quiera).
- Da a conocer qué se debe hacer (métodos) **pero sin mostrar su implemtación** (solo puede tener métodos abstractos).
- Solo puede tener **métodos** con **métodos públicos** (no pueden ser protected o private).
- Solo puede tener "variables"public static final (o sea constantes).
- La palabra tener **abstract** en la definición de métodos no es obligatoria.
- Generalmente las interfaces indican el **PUEDE HACER** de un objeto.

Referencias

[1] relación de clases. En: 1 (). URL: https://blog.visual-paradigm.com/es/what-are-the-six-types-of-relationships-in-uml-class-diagrams/#:~:text=Hay%20seis%20tipos%20principales%20de,%2C%20agregaci%C3%B3n%2C%20asociaci%C3%B3n%20y%20dependencia..

[2] Definicion de Entidad. En: 1 (). URL: https://platzi.com/clases/1566-bd/20197-entidades-y-atributos/.

[3] Interfacez. En: 1 (). URL: https://www.youtube.com/watch?v=hfwtzjOhvKk.

[4] Tipos de datos primitivos JAVA. En: 1 (). URL: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html.