1. **¿En UML cuales son los diagramas dinámicos y estáticos?**

**Dinámicos:** Diagrama de secuencia, colaboración, estado, actividad y casos de uso.

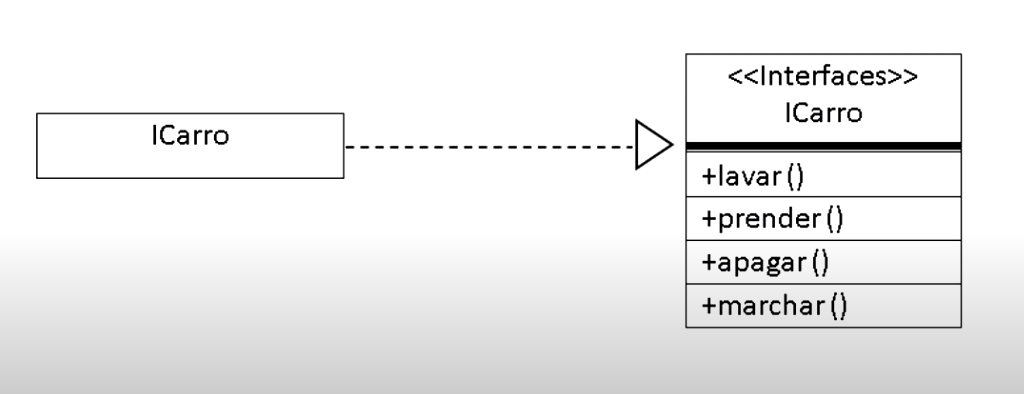
**Estáticos:** Diagrama de clases, objetos, componentes e implementación.

1. **Tipos de relaciones entre clases**

* Generalización.
* Asociación.
* Composición y agregación (casos particulares de asociación).
* Realización.

1. **Diferencia entre Generalización y Realización. Ejemplo en java.**

La generalización es una relación entre una clase padre y una clase hija, donde esta hereda los atributos y métodos definidos en la clase padre a su vez que puede puede definir e implementar métodos propios (operaciones polimórficas), en cambio, la realización en una relación semántica entre clasificadores donde uno va a implementar un contrato que otro garantiza que cumplirá, esta relación se usa con frecuenta entre interfaces y clases.



Ejemplos en java:

* Generalización:

public class Persona {

protected String nombre;

protected Integer dni;

public Persona(nombre, dni){

this.nombr = nombre;

this.edad = edad;

}

}

public class Alumnos **extends** Persona(){

private Integer codigoEstudiante;

private float notaFinal;

public Estudiante(String nombre, Integer dni, Integer codigoEstudiante, Float notaFinal){

super(nombre,dni);

this.codigoEstudiante = codigoEstudiante;

this.notaFinal = notaFinal;

}

}

public class Principal{

public static void main (String [] args){

Estudiante estudiante = new Estudiante(“Nahuel”,”43525348”,”620”);

}

}

* Realización

public inteface IBaseControlador <T, ID>{

public abstract void crear(T objeto);

public abstract void modificar(T objeto);

public abstract void eliminar (T objeto);

public abstract T extraer (ID id);

public ArrayList<T> extraerTodos();

}

public class VendedorControlador **implements** IBaseControlador<Vendedor, Integer>{

@Override (Decorador que indica sobreescrutra)

Public void crear (Vendedor vendedor){ /\*\* Desarrollo de cuerpo\*\*/}

}

1. **Diferencias entre sobrecarga y sobre-escritura. Ejemplo.**

La sobrecarga de métodos es la posibilidad de tener 2 o más métodos con el mismo nombre pero con parámetros y funcionalidades distintas a diferencia de la sobre-escritura que modifica la funcionalidad de un método heredado (se coloca el decorador @Override para indicar la sobreescritura).

public class Persona{

String nombre;

Int edad;

Int documento;

public void correr(){

System.out.println(“Me llamo ”+nombre+ “, tengo ”+edad+”y estoy compitinedo en una maraton”);

}

public void correr(Int km){

System.out.println(“Corrí ”+km+”kilometros”)

}

}

public class Main{

public static void main(String []args){

Persona p = new Persona();

Persona p2 = new Persona();

p.correr();

p2.correr(321);

}

}

1. **¿Para qué sirve el modificador Static? En las clases, métodos y atributos.**

* **En clases:** Para definir clases las cuales queremos llamar sin crear instancias.
* **En atributos:** Para volverlos atributos de clase, si un objeto cambia el valor, se cambiará el valor para todas las instancias de la clase.
* **En métodos:** Para volverlos métodos de clase y poder invocarlos sin necesidad de instanciar la clase.

1. **¿Para qué sirve el modificador Abstract? En clases, métodos y atributos**

* **En clases:** para definir clases generales de las cuales no se desean crear instancias sino heredarlas.
* **En métodos:** para definir la existencia de un método pero no su implementación (método vacio), es decir, las clases hijas que lo hereden serán las encargadas de implementar el método.
* **En atributos:**

1. **¿Para qué sirve el modificador Final? En clases, métodos y atributos.**

* **En clases:** Para definir clases de las cuales no se pretenden crear derivadas de dicha clase.
* **En métodos:** Para definir métodos que no podrán ser redefinidos.
* **En atributos:** Para definir atributos con valor constante.

1. **Nombre algunos de los componentes que podemos tener en un Jframe.**

* JMenuBar.
* JLabel.
* JButton.
* JRadioButton.
* JTextField.
* JTextArea.
* JTable.

1. **Defina que es una transacción. Mencione y describa ACID.**

Es conjunto de operaciones sobre una base de datos que se deben ejecutar como una unidad.

ACID.

* Atomic: Una transacción debe ser atómica y por lo tanto indivisible.
* Consistent: La información que se ejecuta dentro de una transacción debe ser consistente.
* Isolated: Una transacción está aislada del resto y se ejecuta de forma independiente.
* Durable: Los cambios persisten en el tiempo.

1. **¿Qué es un Procedimiento Almacenado (Store Procedure) ventajas y desventajas.**

Es un procedimiento almacenado físicamente en una base de datos. Su implementación de cada gestor de bd. La ventaja es que al ser ejecutado, en respuesta a una petición de usuario, es ejecutado directamente en el motor de bases de datos, el cual usualmente suele estar en un servidor separado. Desventaja: difícil de mantener y desarrollar.

1. **¿Qué son los desencadenadores (Triggers)? En que casos se aplican.**

Un trigger o disparador es un script que se usa en SQL. Consiste en una serie de reglas predefinidas que se asocian a una tabla. El trigger desencadena determinadas acciones de forma automática en las tablas de la base de datos cuando se realiza un INSERT, UPDATE O DELETE.

1. **¿Cuáles son las reglas que debo respetar para realizar un sobrecarga de métodos?**

* Cambiar la lista de parámetros.
* Mantener el nombre del método.

1. **Diferencia entre asociación simple y composición.**

La asociación simple es una relación general entre clases, es decir con menor contenido semántico mientras que la composición es una relación más fuerte en donde la parte no puede existir sin el todo, si se destruye el objeto contenedor el objeto contenido también.

1. **Qué componente es necesario para mostrar un JInternalFrame**

Es necesario un JDesktopPane.

1. **En UML que representa el diagrama de clases**

Representa las clases y las relaciones entre estas que conforman el modelo de un determinado sistema.

1. **Clases abstactas e interfaces.**

* **Abstractas**: Una clase abstracta es aquella de la que no se pueden declarar instancias, dicho de otra manera, no se pueden declarar objetos de una clase abstracta. La finalidad de una clase abstracta es servir como clase base para otras clases a las que generalmente se conoce como clases "concretas".
* **Interfaces**: Una interfaz es como Java Class, pero solo tiene constantes estáticas y método abstracto. Java usa la interfaz para implementar herencia múltiple.

Una interfaz en Java es una colección de métodos abstractos y propiedades constantes. En las interfaces se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas interfaces las que describen la lógica del comportamiento de los métodos.