Programación orientada a objetos

Fundamentos de programación

{desafío} latam_

Utilizar elementos de la programación orientada a objetos para la implementación de una pieza de software que da solución a un problema de baja complejidad

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Comprender conceptos de Herencia y Colaboración para realizar diagramas de clases de alto nivel.*
- *Aplicar conceptos de Herencia y Colaboración para realizar diagramas de clases de alto nivel.*

¿Qué entendemos por herencia y colaboración?

{desafío} latam_

/* Diagramas de secuencia */

Secuencia

Diagrama

- Vista de interacción: diagrama de colaboración y **diagrama de secuencia**.

- Un diagrama de secuencia:
  - muestra un conjunto de mensajes, dispuestos en una secuencia temporal.
  - al mostrarnos interacciones entre los roles, está dentro de los diagramas dinámicos.

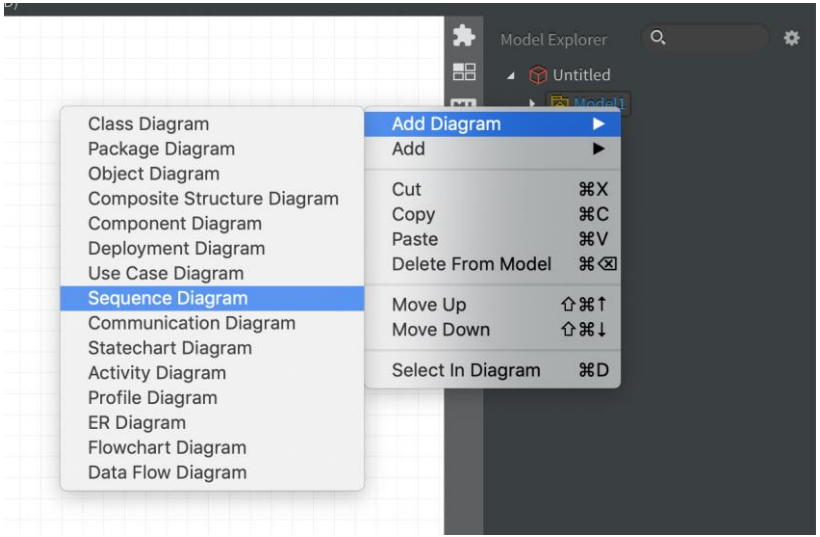
- Puede usarse para mostrar las interacciones en un caso de uso o en un escenario de un sistema de software.

Secuencia

Diagrama

En StarUML

model -> add diagram -> Sequence Diagram



{desafío} latam_

Secuencia

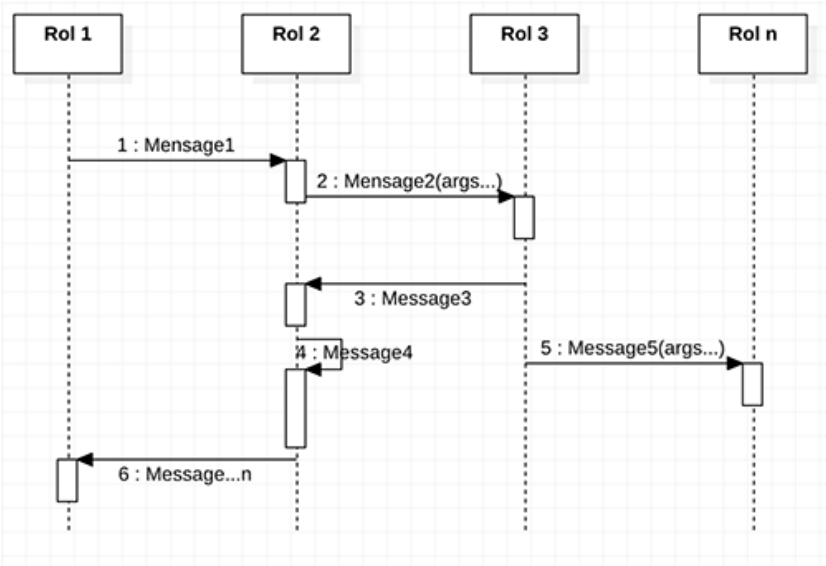
Diagrama

- Rol

Es la descripción de un objeto, que desempeña un determinado papel dentro de una interacción.

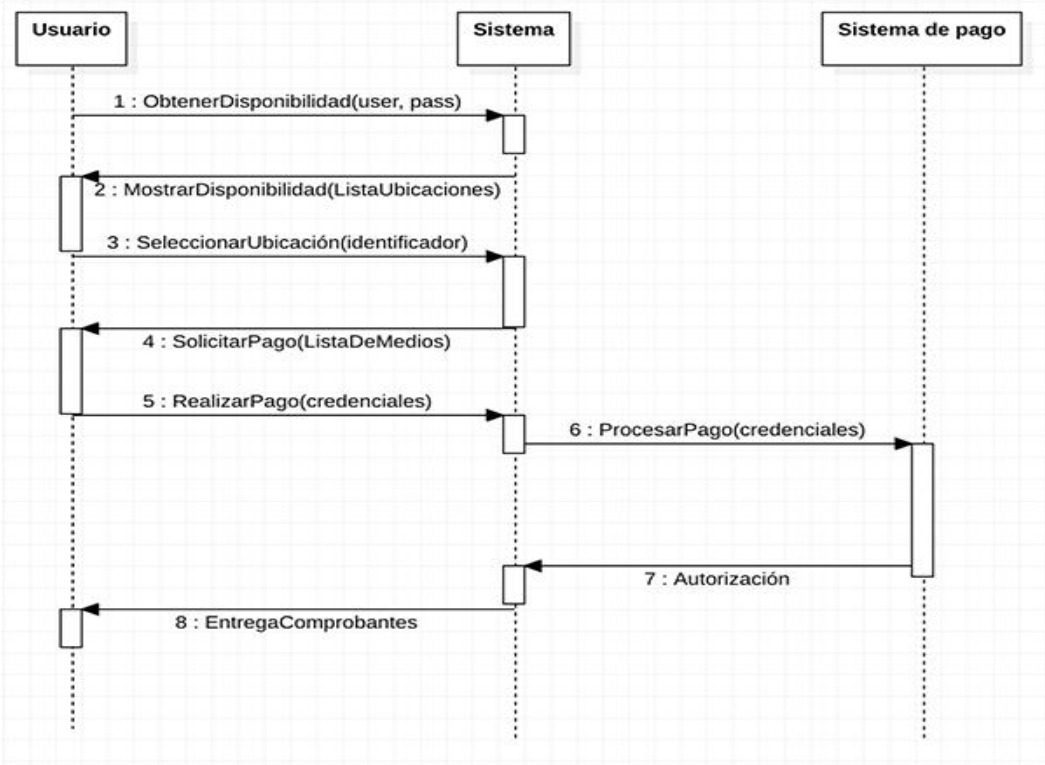
- Mensaje

Es la funcionalidad que permite la comunicación entre los roles. De acá, ya tendremos una idea de lo que serán los métodos y sus interacciones.



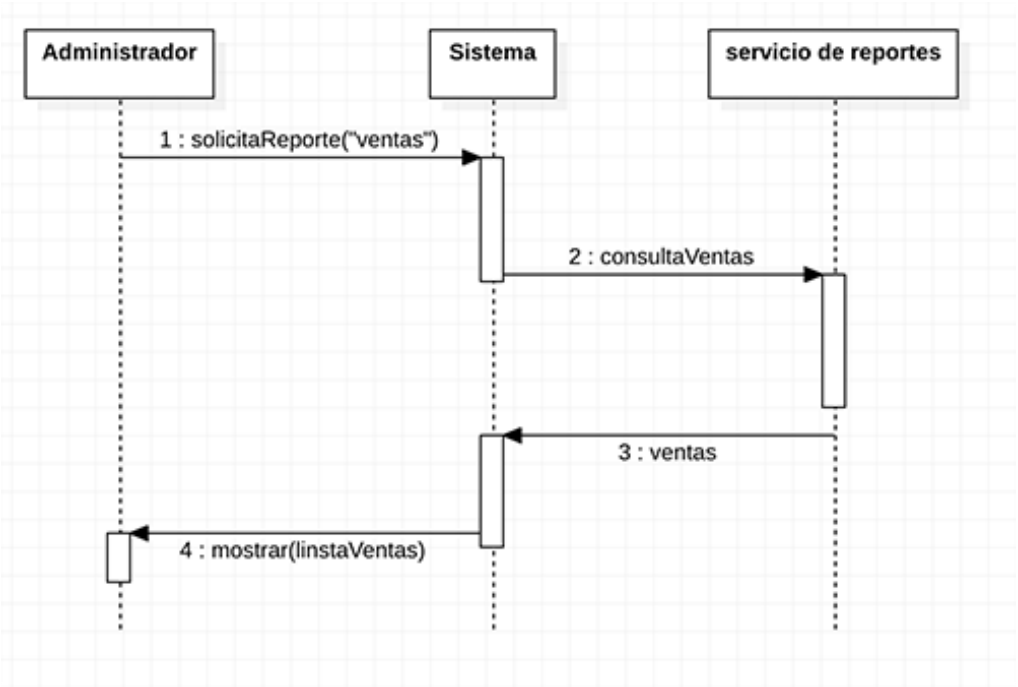
Compra de entradas online

Ejemplo 1



Administrador consultando las ventas

Ejemplo 2



/* Colaboración */

Colaboración

Permite unir uno o más objetos distintos entre sí para crear uno nuevo, esto nace por los conceptos técnicos como, por ejemplo:

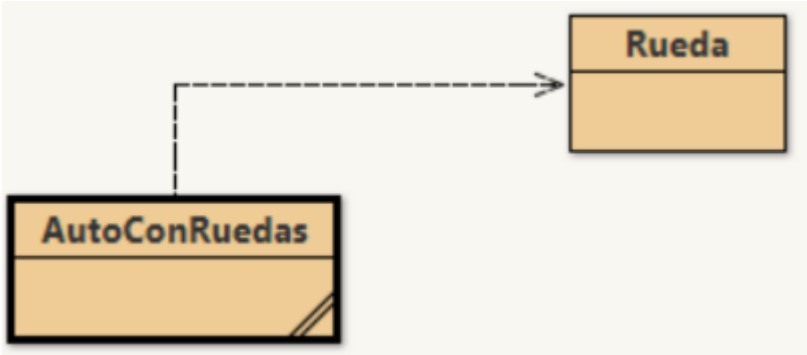
- “requiere de”
- “necesita a”

En las acciones de los mismos objetos.

Colaboración

Ejemplo: Describir una relación de colaboración de dos objetos distintos

- Se necesita crear un nuevo objeto, este objeto será llamado AutoConRuedas.
- Para crear el objeto AutoConRuedas se necesitan 4 objetos de tipo Rueda.
- El objeto Rueda ahora será una propiedad o atributo del objeto AutoConRuedas.



/* Herencia */

Herencia

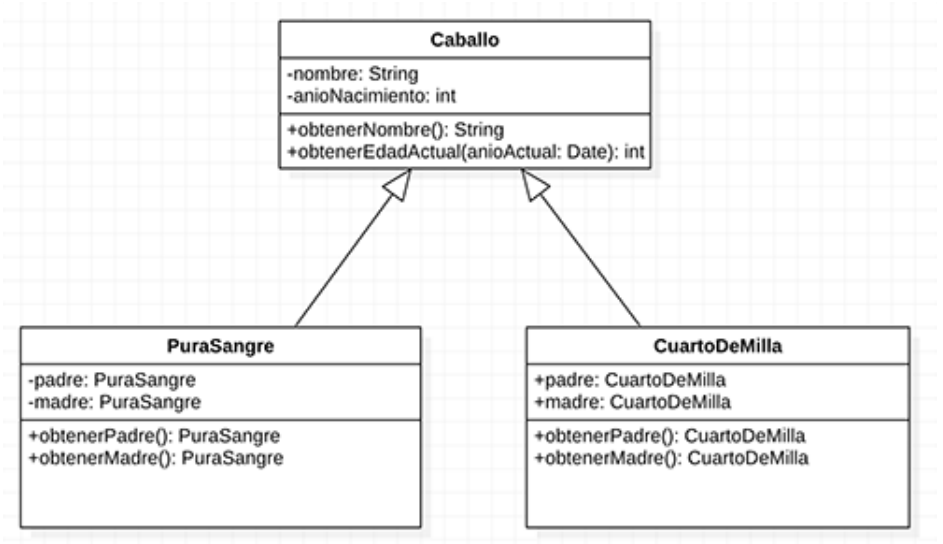
- Objetos con jerarquía relacionada entre ellas.
- *Es distinta a la colaboración porque en la colaboración no existe jerarquía.*
- Java utiliza este pilar para seguir con el concepto de no repetir código, propiedades u operaciones, esto es para utilizar lo menos posible las líneas de código y hacer sus programas más livianos y rápidos.

Con la herencia esto se viene a reafirmar, ya que nace el concepto de **superclase**, donde todas las propiedades y operaciones comunes entre sí, siempre que sean públicas, se heredarán a las clases hijas o de jerarquía menor.

Herencia

Generalización

- Los diagramas de clase, también pueden mostrar relaciones entre las clases.
- Una clase que sea una subclase de otra clase se conecta con ella mediante una flecha con una línea sólida y con una punta triangular hueca.
- La flecha apunta de la subclase a la superclase.



{desafío} latam_

Podemos relacionar esto con la relación de herencia en la POO (Programación orientada a objetos).

Clase caballo

Ejemplo

```java
package cl.desafiolatam.uml.diagramaclase;

import java.util.Date;

public class Caballo {
    private String nombre;
    private int anioNacimiento;

    public Caballo(String nombre, int anioNacimiento) {
      this.nombre = nombre;
      this.anioNacimiento = anioNacimiento;
    }

    public String obtenerNombre() {
        // TODO implementar aquí.
        return "";
    }

    public int obtenerEdadActual(Date anioActual) {
        // TODO implementar aquí.
        return 0;
    }

}
```

{desafío} latam_

Clase PuraSangre

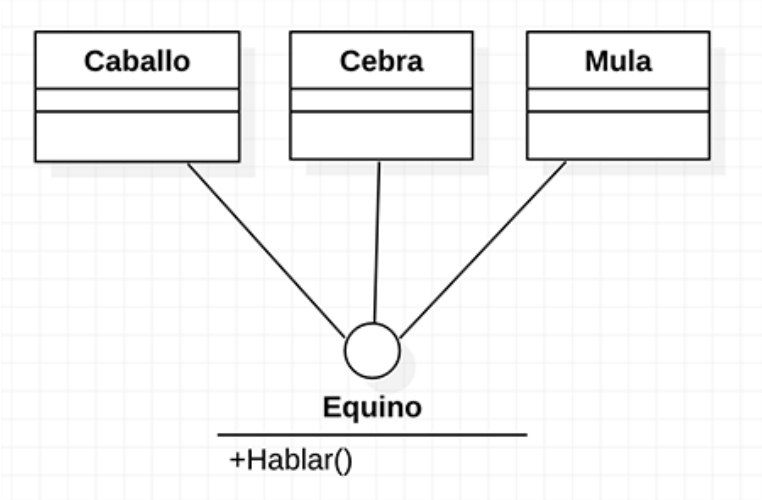
```java
package cl.desafiolatam.uml.diagramaclase;

import java.util.Date;

public class PuraSangre extends Caballo {
    private PuraSangre padre;
    private PuraSangre madre;

    public PuraSangre(PuraSangre padre, PuraSangre madre, String nombre, int anioNacimiento) {
      super(String nombre, int anioNacimiento);
      this.padre = padre;
      this.madre = madre;
    }

   public PuraSangre obtenerPadre() {
      // TODO implementar acá
      return null;
   }

    public PuraSangre obtenerMadre() {
      // TODO implementar acá
      return null;
    }

}
```

Clase CuartoDeMilla

```java
package cl.desafiolatam.uml.diagramaclase;

import java.util.Date;

public class CuartoDeMilla extends Caballo {
    public CuartoDeMilla padre;
    public CuartoDeMilla madre;

    public CuartoDeMilla(PuraSangre padre, PuraSangre madre, String nombre, int anioNacimiento) {
      super(String nombre, int anioNacimiento);
      this.padre = padre;
      this.madre = madre;
    }

    public CuartoDeMilla obtenerPadre() {
        // TODO implementar aquí.
        return null;
    }

    public CuartoDeMilla obtenerMadre() {
        // TODO implementar aquí.
        return null;
    }

}
```

Implementación de Interfaces en UML

Nos ayuda a poder diseñar un bosquejo de lo que se pretende construir, de esta forma, podemos hacer un mapa completo de la estructura que tendrán nuestras clases e interfaces, además de poder generar el código desde la misma herramienta.



Clase Equino y Clase Caballo

```java
package cl.desafiolatam.uml.interfaces

public interface Equino {

    public abstract void Hablar();

}
```

```java
package cl.desafiolatam.uml.interfaces

public class Caballo implements Equino {

    public void Hablar() {
        // TODO implementar aquí.
    }

}
```

Clase Cebra y Clase Mula

```java
package cl.desafiolatam.uml.interfaces


public class Cebra implements Equino {

    public Cebra() {}
    @Override
    public void Hablar() {
        // TODO implementar aquí.
    }

}
```

```java
package cl.desafiolatam.uml.interfaces


public class Mula implements Equino {

    @Override
    public void Hablar() {
        // TODO implementar aquí.
    }

}
```

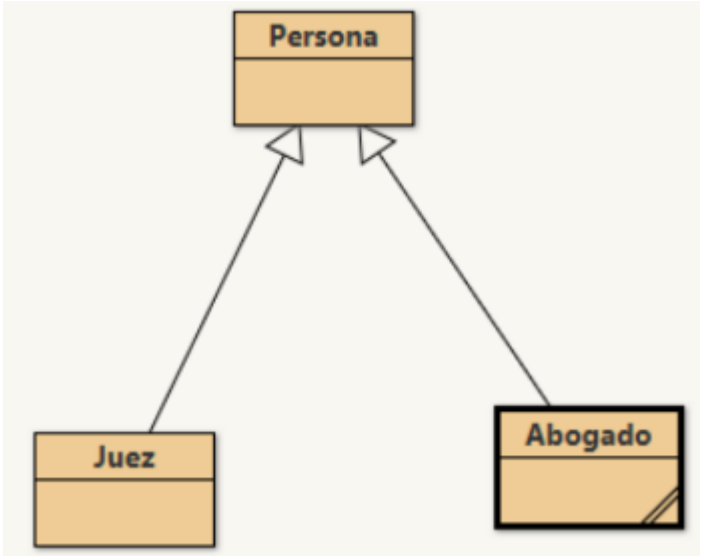
{desafío} latam_

Otro ejemplo

- Tenemos el objeto **Abogado** con sus atributos nombre, rut y título, también tenemos el objeto llamado **Juez** con sus atributos nombre, rut y dirección.
- Estos objetos tienen atributos similares y relación entre sí, y para no crear dos objetos donde se repiten las mismas operaciones y atributos, se crea una superclase donde se identifica la relación directa entre **Juez** y **Abogado**, esta relación es que ambos son personas.
- Con esta información podemos crear el objeto Persona con los atributos rut y nombre, y bajo la definición de lo que es herencia, se dirá que la clase Persona le heredará todos sus atributos y operaciones públicas a las clases **Juez** y **Abogado**.

Otro ejemplo

- La clase **Juez** solo tendrá el atributo dirección y **Abogado** solo el atributo título, sin embargo, Java interpretará que la clase Persona y **Abogado** es solo una, así como Persona y **Juez**, ya que al heredar y tener jerarquía Java lo interpreta como una sola clase, para la Java Virtual Machine (JVM) solo existen dos clases **Juez** y **Abogados** con atributos y operaciones de la clase Persona.



Ejercicio guiado

Identificar las clases y sus relaciones

En un zoológico hay un zorro, un león y una ciudad.
El zorro tiene:

- Edad
- Tipo de zorro
- Origen

El león tiene:

- Edad
- Origen
- Sexo

Identificar y describir las clases y sus relaciones.

Identificar las clases y sus relaciones

Solución

Se debe identificar lo siguiente:

1. Si hay relación entre un león y un zorro.
2. Dónde están ubicados el león y el zorro.
3. Características de cada objeto.
4. Identificar la relación entre zoológico, león y zorro.

Identificar las clases y sus relaciones

Luego de identificar las clases y sus relaciones, quedaría de la siguiente manera:

Animal:

- edad: int
- origen: String

Zorro es un Animal:

- tipoDeZorro: String

León es un Animal:

- sexo: String

Zoológico:

- zorro: Zorro
- león: León
- ciudad: String

¿En qué aspectos te enfocarías para realizar una herencia?

¿Para qué utilizamos la colaboración?

Los principios del modelado

- El equipo de software, tiene como objetivo principal, elaborar software y no modelos.
- Viajar ligero, no crear más modelos de los necesarios.
- Tratar de producir el modelo más sencillo que describa al problema o al software.
- Construir modelos susceptibles al cambio.
- Ser capaz de enunciar un propósito explícito para cada modelo que se cree.
- Adaptar los modelos que se desarrollan al sistema en cuestión.
- Tratar de construir modelos útiles, pero olvidarse de construir modelos perfectos.
- No ser dogmáticos respecto a la sintaxis del modelo. Si se tiene éxito para comunicar contenido, la representación es secundaria.
- Si su instinto dice que un modelo no es correcto a pesar de que se vea bien en el papel, hay razones para estar preocupados.
- Obtener retroalimentación tan pronto como sea posible.

Próxima sesión...

- *Comprender el comportamiento de las relaciones entre los conceptos de herencia y colaboración.*
- *Crear diagramas de clases de alto nivel para tener una mejor lectura del código que vamos a desarrollar.*

{desafío}
latam_

Academia de
talentos digitales