/* UML */

Sesión conceptual 02

{desafío} latam_

- Objetivo de la sesión
- Activación de conceptos clave

- Conceptualización
- Ejercitación

- Reflexión

Inicio

{desafío}
latam_

/* Diagramas de frecuencia */

Objetivo

- Tener nociones de la vista de interacción.
- Conocer el concepto de los diagramas de secuencia.
- Conocer el papel de los roles y los mensajes.
- Construir diagrama de secuencia.
- Eficiencia en la construcción de los diagramas de secuencia.

Introducción

- Extraer los casos de uso de una aplicación por medio de los requerimientos funcionales de un usuario, parece ahora una tarea simple. Pero ¿qué pasa con su comportamiento?. Tenemos las interacciones que asumimos que tienen los actores con el sistema; pero es posible analizar esas interacciones con un poco más de detalle para poder interpretar si el comportamiento que se pensó o se piensa, sea el correcto. Para analizar este tipo de situaciones, usaremos un diagrama de secuencia.

Desarrollo

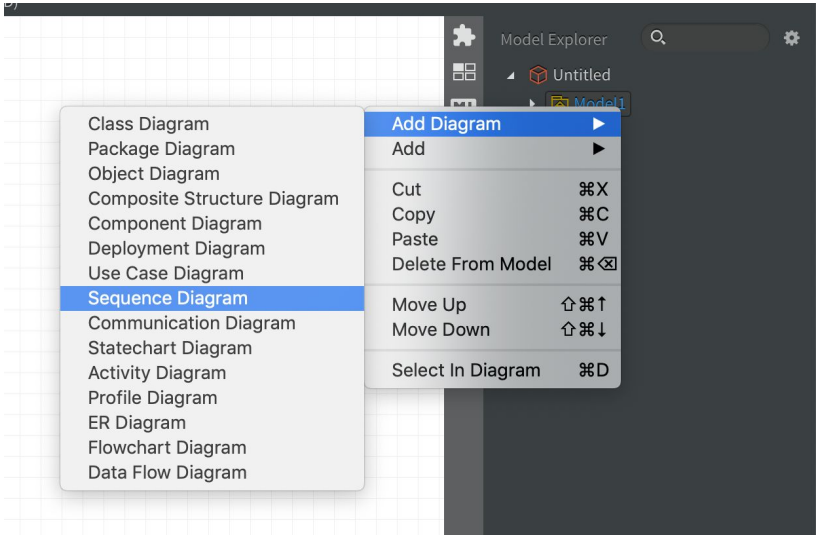
{desafío}
latam_

Diagrama de secuencia

- Vista de interacción: diagrama de colaboración y diagrama de secuencia.
- Un diagrama de secuencia, muestra un conjunto de mensajes, dispuestos en una secuencia temporal.
- El diagrama de de secuencia al mostrarnos interacciones entre los roles, está dentro de los diagramas dinámicos.
- Puede usarse un diagrama de secuencia, para mostrar las interacciones en un caso de uso o en un escenario de un sistema de software.

En StarUml

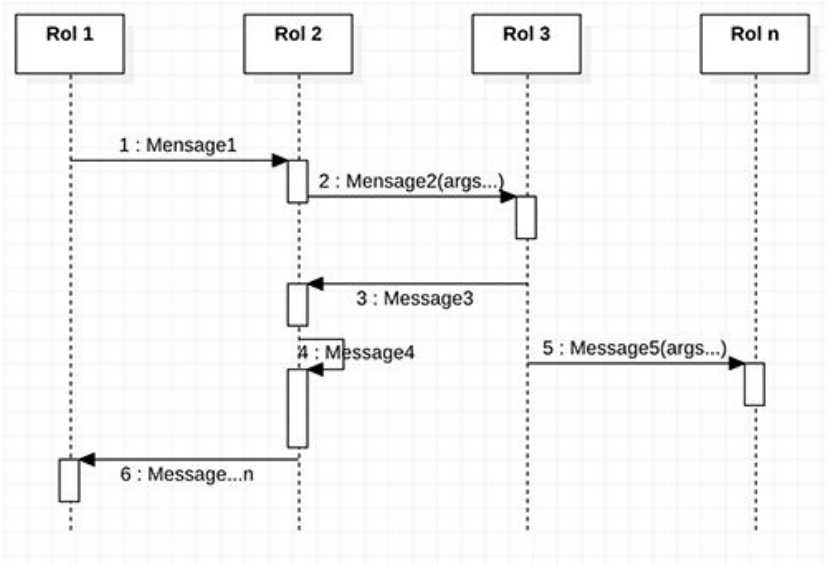
- model -> add diagram -> Sequence Diagram



Rol y Mensaje

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

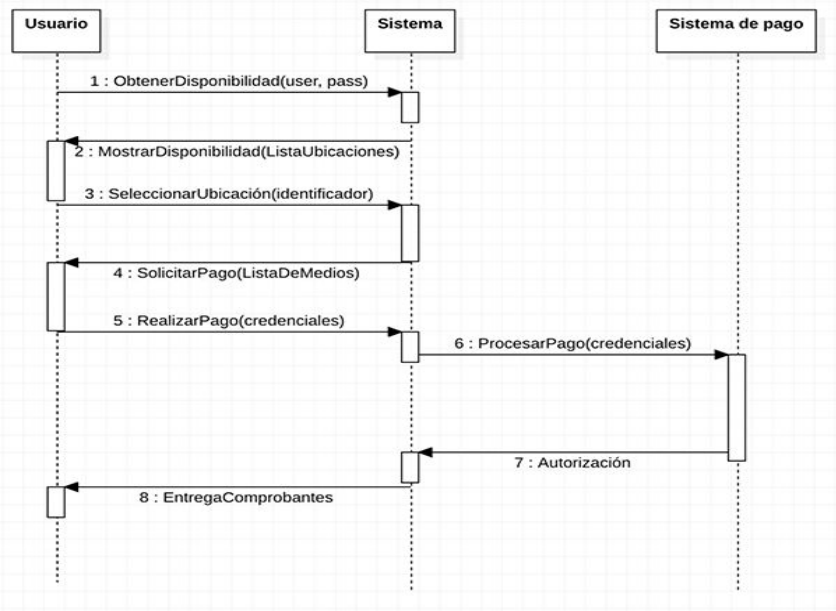
- **Mensaje**
  - Es la funcionalidad que permite la comunicación entre los roles. De acá, ya tendremos una idea de lo que serán los métodos y sus interacciones.



{desafío} latam_

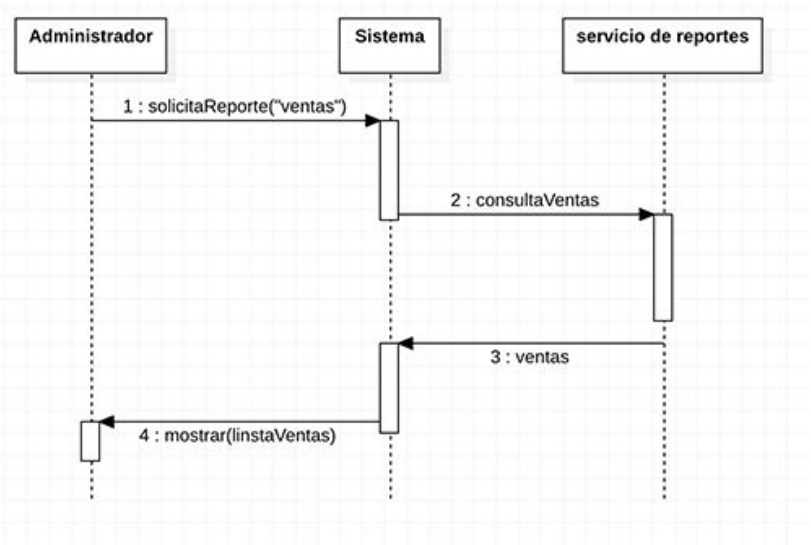
Ejemplo 1

- Flujo normal: compra de entradas online.



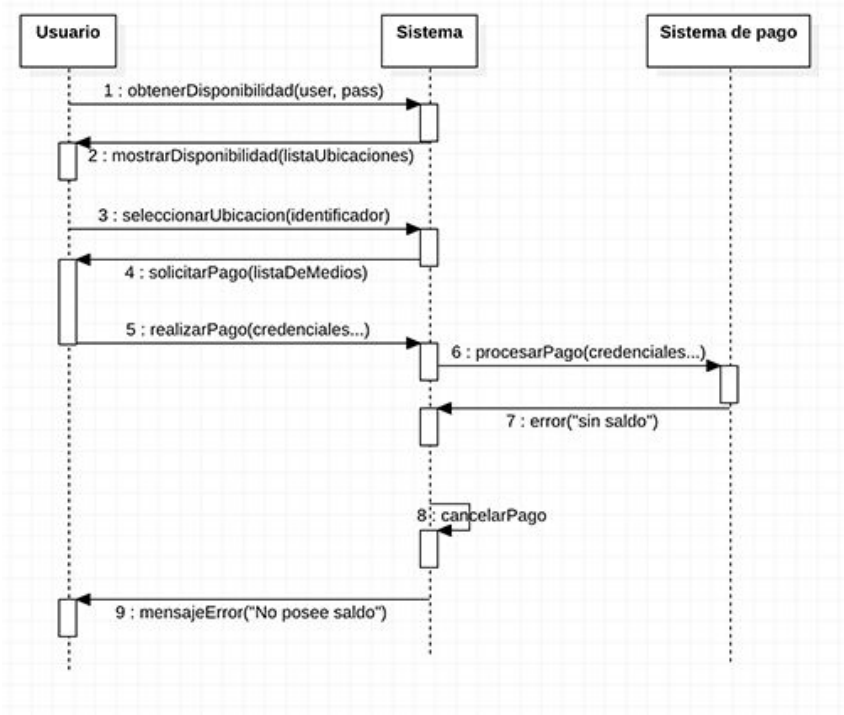
Ejemplo 2

- Flujo normal del administrador consultando las ventas.



Ejemplo 3

- Un flujo alternativo, es aquél que no forma parte del problema principal, pero amerita que sea expuesto, por ejemplo, si es que pasa algún error en el flujo y saber cómo abordarlo.
- Flujo alternativo, cuenta no posee saldo.



/* Diagrama de clases */

Objetivo

- Relacionar el diagrama con la POO.
- Reconocer las notaciones como cajas y generalizaciones.
- Construir diagrama de clases.
- Llevar los diagramas de clase a código en Java.

Introducción

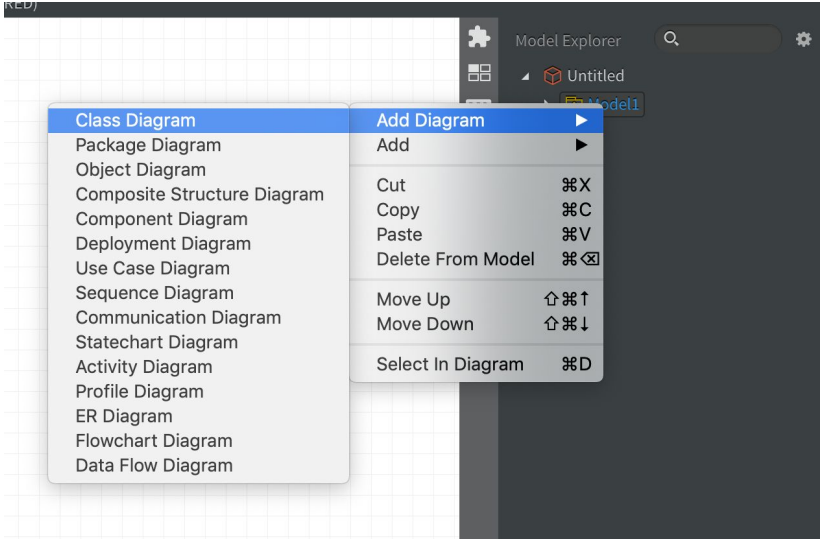
- Para modelar las clases e interfaces, incluidos sus atributos, operaciones, relaciones y asociaciones con otras clases, el UML proporciona el diagrama de clases, que aporta una visión **estática** o de estructura de un sistema, sin mostrar la naturaleza dinámica de las comunicaciones entre los objetos de las clases.
- El diagrama de clase, además de ser de uso extendido, también está sujeto a la **más amplia gama de conceptos de modelado.** Aunque los elementos básicos son necesarios para todos, los conceptos avanzados se usan con mucha menor frecuencia. Es por eso que **se toman los temas más importantes y suficiente para lograr los objetivos propuestos.**

Desarrollo

{desafío}
latam_

En StarUml

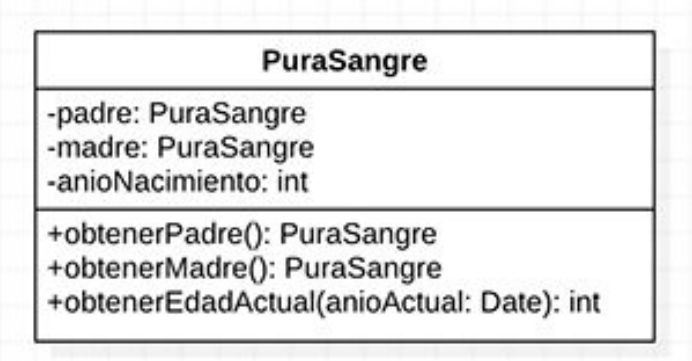
- model -> add diagram -> Class Diagram.



Componentes de un diagrama de clases

- Atributos de una clase.
- Operaciones.
- Cajas.

Puede haber otros atributos que no se muestren en el diagrama.



Visibilidad de los atributos y operaciones

- **-** Visibilidad privada.
- **#** Visibilidad protegida.
- **~** Paquete.
- **+** Visibilidad privada.
- También es posible, especificar si un atributo es del tipo static subrayando.

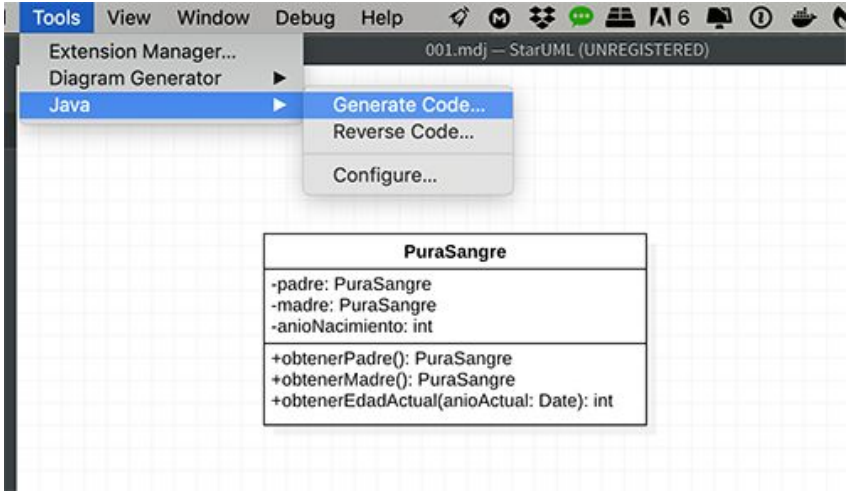
- Para los atributos especificamos el nombre y el tipo de retorno de la siguiente forma:
  - nombre:tipoDeRetorno
- En el caso de las operaciones, tenemos estas más opciones:
  - nombreOperacion(nombreParametro:tipo, ...)
  - nombreOperacion():tipoRetorno
  - nombreOperacion(nombreParametro:tipo, ...):tipoRetorno

La implementación de la caja de ejemplo en Java

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

import java.util.Date;

public class PuraSangre {
  private PuraSangre padre;
  private PuraSangre madre;
  private int anioNacimiento;

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

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

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

}
```

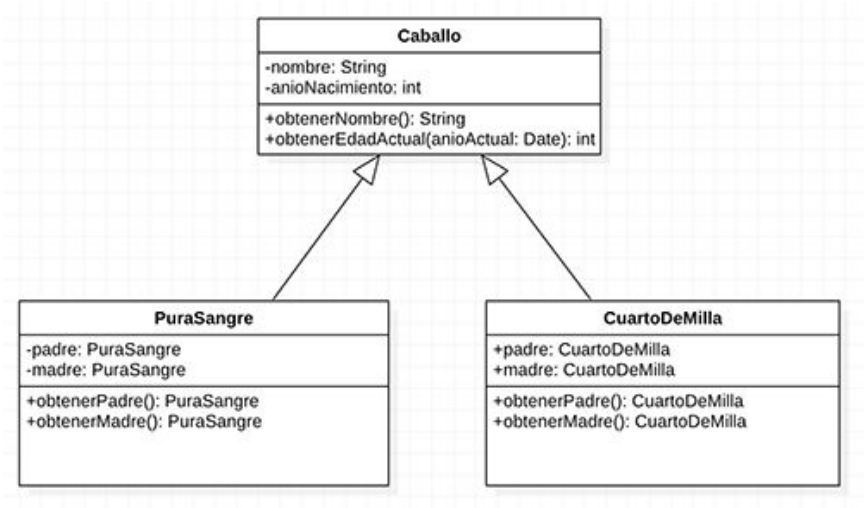
Exportar desde StarUml

- tools -> Java -> generate code.
- Seleccionamos la el modelo y la ubicación en donde lo queremos dejar. Esto genera una carpeta con el nombre del modelo y si revisamos su interior, encontraremos los archivos generados, en este caso es solo uno.



Generalización (Herencia)

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. Podemos relacionar esto con la relación de herencia en la POO.



Clase Caballo

```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;
    }

}
```

Clase PuraSangre

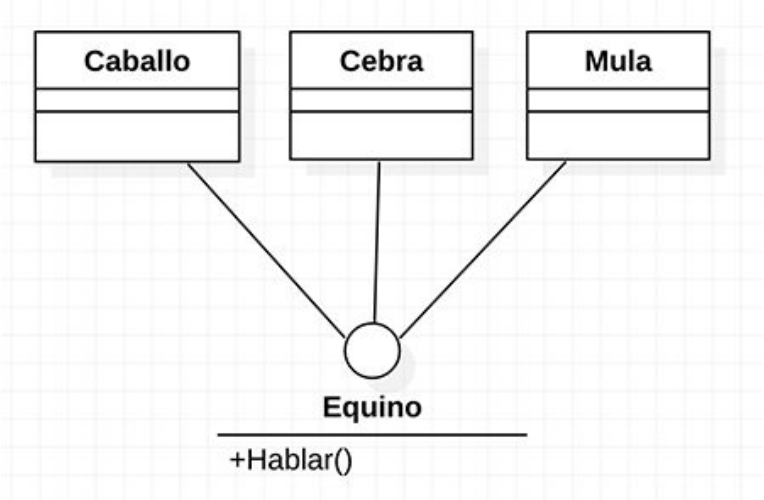
```java
package cl.desafiolatam.uml.diagramaclase;

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

Podemos expresar además, la relación de implementación de interfaces. Esto 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

```
package cl.desafiolatam.uml.interfaces

public interface Equino {

    public abstract void Hablar();

}
```

Clase: Caballo

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

public class Caballo implements Equino {

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

}
```

Clase: Cebra

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


public class Cebra implements Equino {

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

}
```

Clase: Mula

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


public class Mula implements Equino {

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

}
```

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.

Quiz

{desafío}
latam_

Cierre

{desafío}
latam_

¿Existe algún concepto que no hayas comprendido?

Volvamos a revisar los conceptos que más te hayan costado antes de seguir adelante

Reflexionemos

{desafío} latam_

{desafío} latam_

Academia de talentos digitales