# Academia Java

# Xideral

**Nombre: Josselyn Ileana Mosco García**

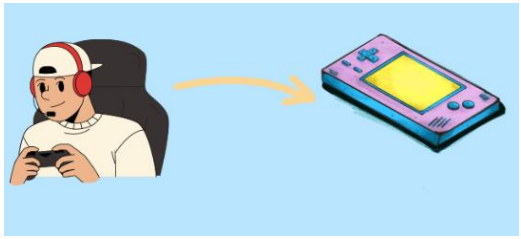
## EXAMEN SEMANA 2

**02 de diciembre de 2022**

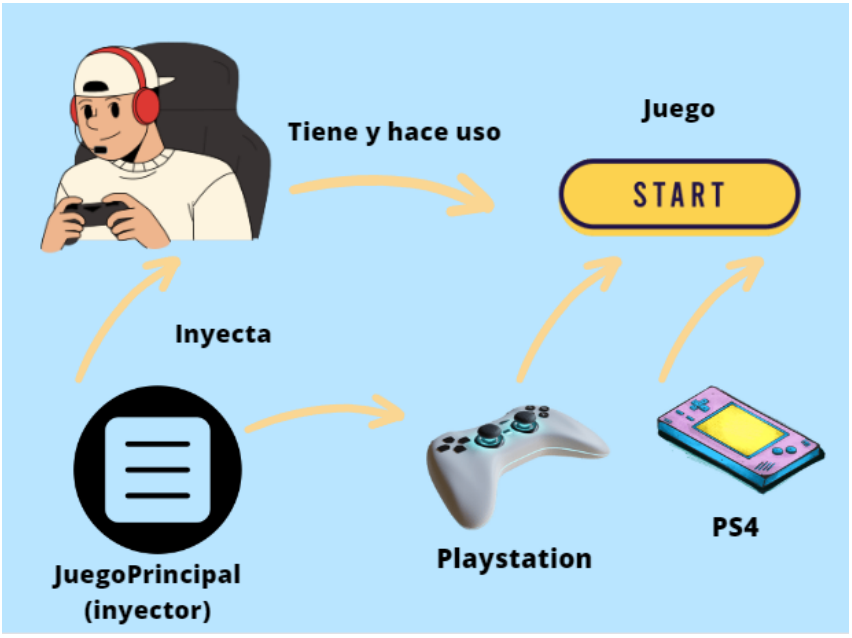
**Examen 2**

**1.- Explica que es Inyección de dependencias**

Con un ejemplo, supongamos que de una clase principal quiero implementar una clase jugador que juegue en ps4, e implementamos el método jugar, pero ahora mi jugador quiere cambiarse al play, entonces tendríamos que cambiar las propiedades del jugador porque ya no sería ps4. Para no estar cambiando a mi jugador, se implementa la inyección de dependencias.



La idea es que las clases no dependan directamente de la clase principal sino de una abstracción de la misma, para no estar modificando el jugador, en este caso. Entonces, le decimos al jugador que no dependa del tipo de juego como ps4 o playstation, sino que dependa de un objeto que implementa la interfaz juego así cualquier objeto que implemente esa interfaz podrá ser usado por esa clase.

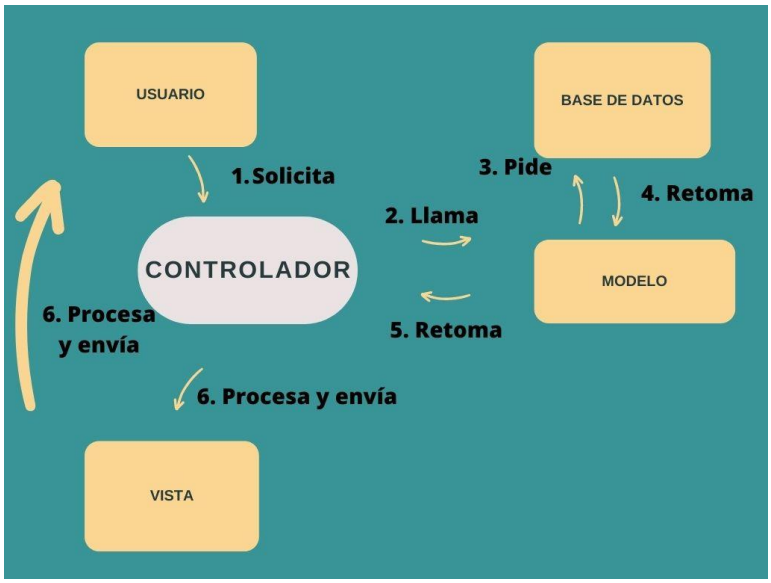


## 2.- Diagrama y explica una arquitectura web usando MVC

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

- El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- La **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
- El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.




1.- El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

2.- El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

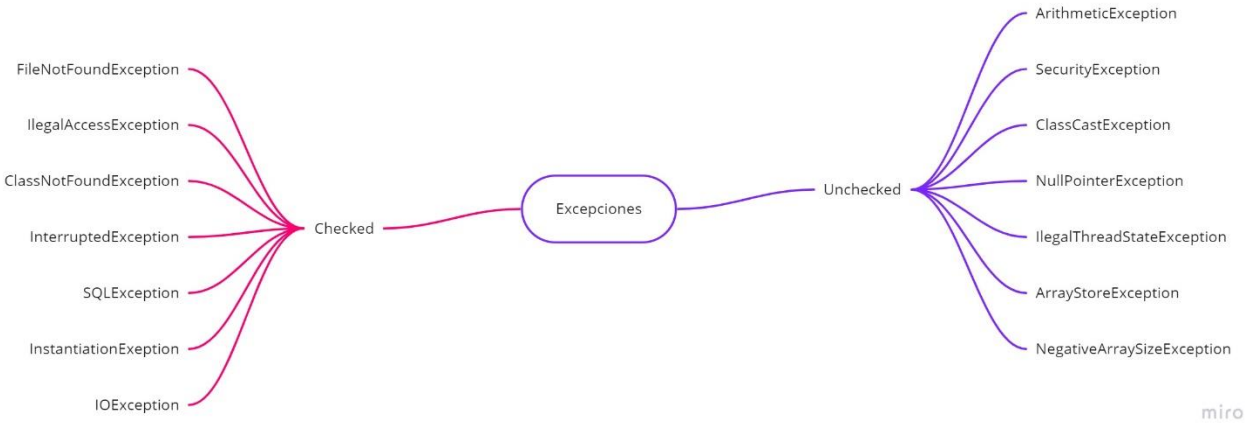
3-4.- El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). De ser necesario consultará la base de datos y obtendrá información de respuesta.

5.- El controlador retoma a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista.

6.-La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

### 3.- Explica los diferentes tipos de excepciones

Las excepciones son eventos que se producen cuando se ejecuta el programa de forma que interrumpen el flujo normal de instrucciones. Estas pueden manejarse o controlarse para evitar que el programa se detenga. Existen excepciones de tipo checked que son un objeto de la clase Exception y se da cuando existe algún error que puede ser corregido; es necesario usar bloques try/catch o la palabra reservada throws para poder manejarlas. Por otro lado existen las excepciones de tipo unchecked y estas se heredan de la clase RuntimeException; no es necesario darle tratamiento pero sí se puede si así lo deseamos. De manera adicional existen los errores que son problemas que no se pueden controlar como fallas en la red o algún error en la base de datos.



1. **ArithmeticException**

   Se lanza cuando se ha producido una condición excepcional en una operación aritmética.

2. **ArrayIndexOutOfBoundsException**

   Se lanza para indicar que se ha accedido a una matriz con un índice ilegal. El índice es negativo o mayor o igual que el tamaño de la matriz.

3. **ClassNotFoundException**

   Esta excepción se genera cuando intentamos acceder a una clase cuya definición no se encuentra

4. **FileNotFoundException**

   Esta excepción se genera cuando un archivo no es accesible o no se abre.

5. **IOException**

   Se lanza cuando una operación de entrada-salida falla o se interrumpe

6. **InterruptedException**

   Se lanza cuando un hilo está esperando, durmiendo o realizando algún procesamiento y se interrumpe.

7. **NoSuchFieldException**

   Se lanza cuando una clase no contiene el campo (o variable) especificado

8. **NoSuchMethodException**

   Se lanza al acceder a un método que no se encuentra.

9. **NullPointerException**

   Esta excepción se genera cuando se hace referencia a los miembros de un objeto nulo. Nulo no representa nada

10. **NumberFormatException**

    Esta excepción se genera cuando un método no puede convertir una string en un formato numérico.

11. **RuntimeException**

    Representa cualquier excepción que se produzca durante el tiempo de ejecución.

12. **StringIndexOutOfBoundsException**

    Lo lanzan los métodos de la clase String para indicar que un índice es negativo o mayor que el tamaño de la string

## 4.- Explica los 4 propósitos del *final* con código

*Final variable:*

Una vez declara como final la variable, el valor de la variable no se pudo cambiar. Es como una constante.

*Final Atributo*

Es similar al anterior. Permite convertir un atributo a una constante, por esta razón no es posible generar un setter para dicho atributo.

*Final method:*

Una palabra clave final en un método que no se puede anular. Si un método se marca como final, la subclase no puede anularlo.

*Final Class:*

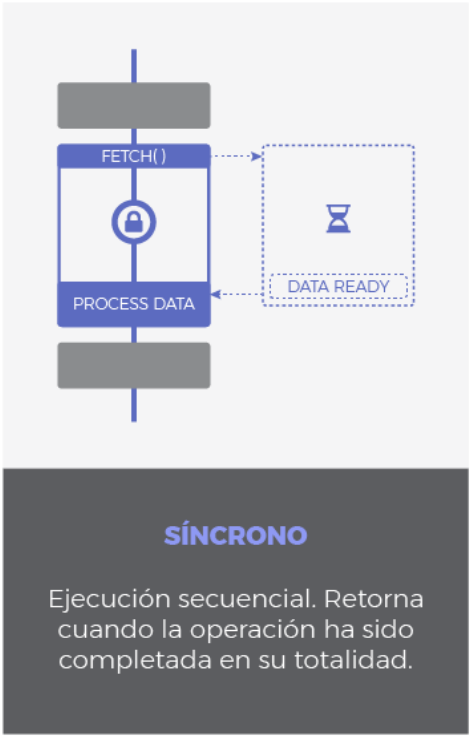
Si una clase se declara como final, ya no puede ser subclasificada. Ya que ninguna clase puede extender la clase final.

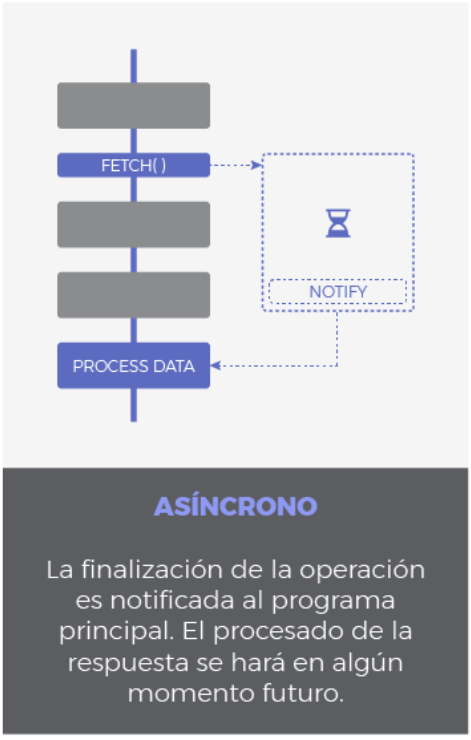
5.- Diferencia de un proceso síncrono y asíncrono

Se refiere a ¿cuándo tendrá lugar la respuesta?:

Síncrono: La respuesta sucede en el presente, una operación síncrona esperará el resultado.

Asíncrono: La respuesta sucede a futuro, una operación asíncrona no esperará el resultado.



6.- Exponer un código con multicacth, try with resources

Ejemplo1.java

```java
package com.jimg.multi;

import java.util.InputMismatchException;
import java.util.Scanner;

public class Ejemplo1 {

    public static void main(String[] args) {
        int x = 10, y, z;
        System.out.print("Ingrese y: ");
        try {
            y = new Scanner(System.in).nextInt();
            z = x / y;
            System.out.println("Resultado: " + z);
        }
        catch (InputMismatchException ime) {
            System.out.println("No es un entero... ¬¬");
        }
        catch (ArithmeticException ae) {
            System.out.println("No se puede dividir por cero");
        }
        catch (Exception ex) {
            System.out.println("Error: " + ex.getMessage());
            ex.printStackTrace(System.out);
        }
        System.out.println("Adiós");
    }

}
```

Se utilizan muchos catch para cubrir todos los posibles errores en orden, porque si se pone antes, bota el programa, para esto existe la jerarquía; de las más particulares a las generales.

Ejemplo2.java

```java
package com.jimg.tryres;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class Ejemplo2 {

    public static void main(String[] args) throws IOException {
        //FileWriter archivo = null;
        PrintWriter printer = null;

        try(FileWriter archivo = new FileWriter("C:\\Users\\HP\\Desktop\\Examen_2\\Archivo2.txt", true)){
            printer = new PrintWriter(archivo);

            printer.println("Hola, soy Josselyn");
        } catch (Exception e) {
            //Exception
        }
    }

}
```

Try with resources. Permite acceder a recursos y que sean gestionados de manera interna.

7.- Realizar un código usando lambdas

```java
package com.jimg.lambda;

@FunctionalInterface
public interface ICalcu {
    //Definimos el método con un parámetro
    void operacion(double a, double b);

}

```

```java
package com.jimg.lambda;

public class Ejecutor {
    public static void main(String[] args) {

        //creamos una referencia hacía la interfaz
        //usamos expresión lambda
        ICalcu calculadora = (n1, n2) -> {
            double resultado = n1+n2;
            System.out.println("La suma es: "+resultado);
        };
        ICalcu calculadora1 = (n1, n2) -> {
            double resultado = n1-n2;
            System.out.println("La suma es: "+resultado);
        };

        //para imprimir hacemos referencia a la variable que apunta a la clase
        calculadora.operacion(5, 8);
        calculadora1.operacion(15, 8);
```