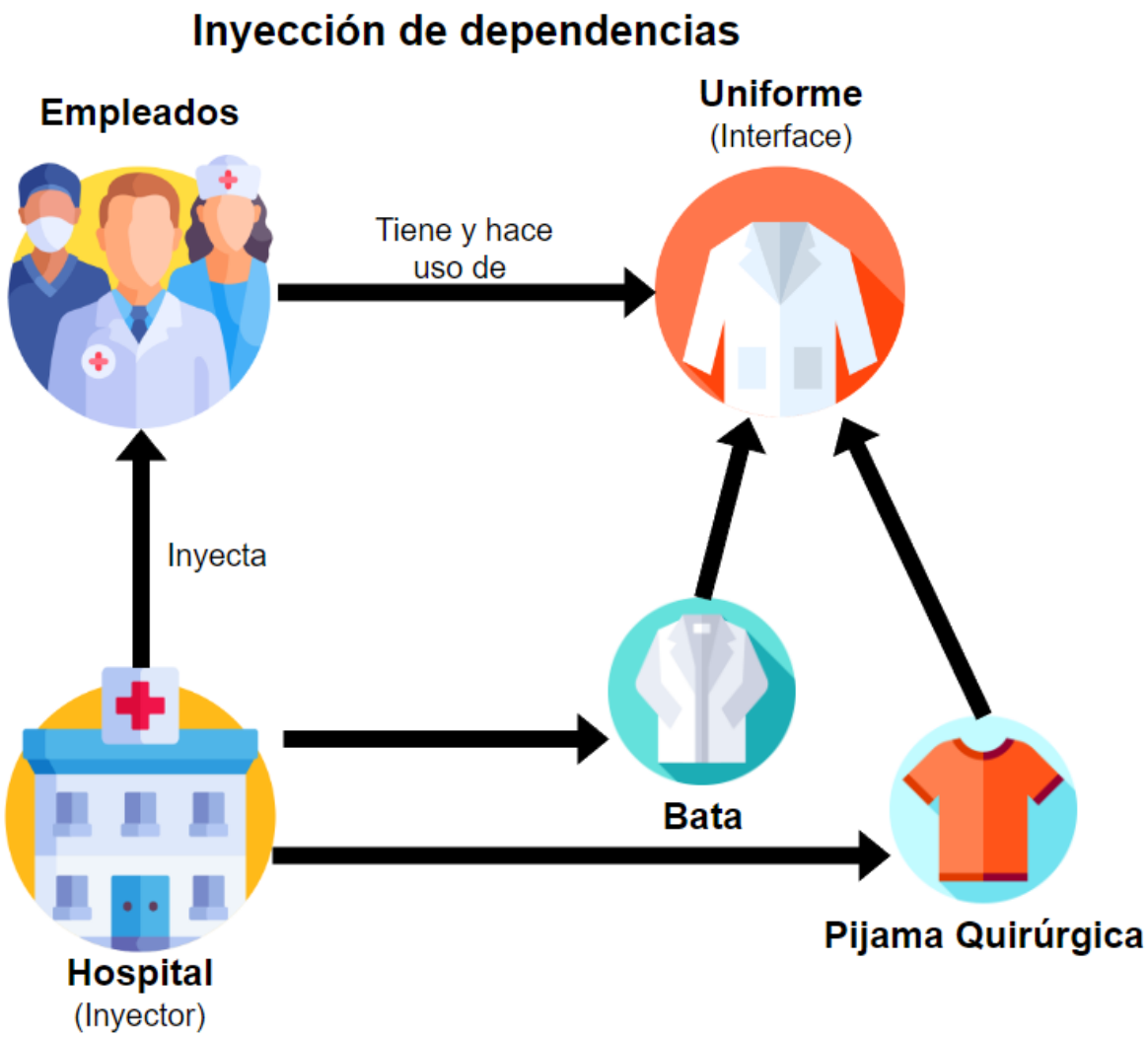
Examen Semana 2 - Fabiola Gómez Montiel

1. Explica qué es la inyección de dependencias con código y con diagrama

En este caso, cada empleado debe generar su propio uniforme para poder hacer uso de él. Codificar de esta forma es deficiente ya que cualquier cambio que deba hacerse en el uniforme, deberá verse reflejado en la clase Empleados y mantener el código a largo plazo será complejo y muy tardado.



Cuando usamos la inyección de dependencias, el Hospital (que en este caso es nuestro inyector) se encargará de otorgar un Uniforme (ya sea Bata o Pijama) a cada Empleado, sin que el empleado tenga una relación directa con los mismos.

De esta forma, todas las modificaciones que se hagan a los Uniformes dejarán de afectar a la clase Empleados, pues ellos sólo deberán preocuparse de hacer uso de los Uniformes.

🔗 **Link al código**

2. Diagrama y explica una arquitectura web usando MVC.

MVC significa Modelo-Vista-Controlador y es un patrón de diseño de software cuya principal característica es dividirse en 3 componentes con tareas específicas.

Entre sus ventajas se encuentran:

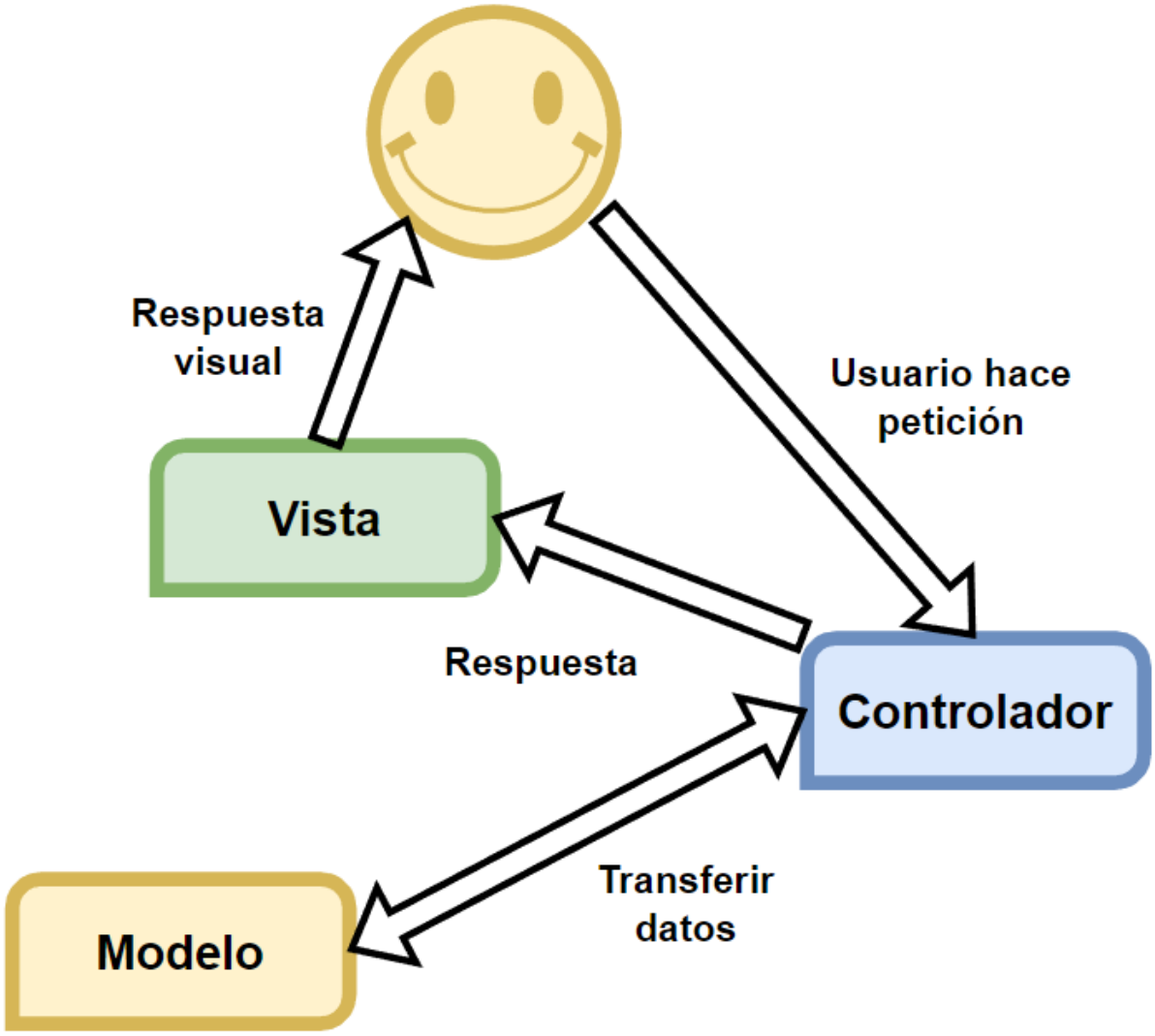
- Es fácilmente escalable.
- Permite el trabajo en equipo pues diferentes miembros pueden trabajar en componentes distintos.
- Debido a que solo tiene 3 componentes es sencillo organizarlo.

Componentes:

✨ **Modelo:** Es la capa que se encarga de manejar datos, analiza la información que le regresa el controlador y enviar información que el controlador necesita para trabajar adecuadamente.

🔷 **Controlador:** Recibe la información del modelo y la manda a la vista, aunque también se comunica con el modelo para pedirle datos que manda a la vista. Actúa como un intermediario.

💚 **Vista:** Es la representación visual o interfaz gráfica a través de la cual se muestran los datos recibidos y con la que puede interactuar el usuario.



3. Explica los diferentes tipos de excepciones.

Las excepciones son errores que se presentan al momento de correr el código y que pueden interrumpir o detener definitivamente el flujo del programa. Las excepciones pueden manejarse o controlarse para evitar que el programa se detenga

Existen dos tipos principales:

- **Checked:** Son un objeto de la clase Exception y se dan
- **Unchecked:** Estas excepciones heredan de la clase

cuando existe algún error que puede ser corregido. Con este tipo de excepciones es necesario utilizar un bloque try/catch o la palabra reservada throws para poder controlarla.

Ejemplos:

- ClassNotFoundException
- InterruptedException
- InstantiationException
- IOException
- SQLException
- IllegalAccessException
- FileNotFoundException

RuntimeException.
No es necesario que le demos tratamiento a estas excepciones, sin embargo es posible hacerlo si así lo deseamos.

Ejemplos:

- ArithmeticException
- ClassCastException
- NullPointerException
- NegativeArraySizeException
- ArrayStoreException
- IllegalThreadStateException
- SecurityException

❌ Adicionalmente existen los **errors**, que son problemas que no se pueden controlar de ninguna manera, por ejemplo: conflictos con la red, falta de archivos o fallos en bases de datos.

---

4. Explica los 4 propósitos del final con código.

1. **Final para variables locales:** En este contexto, el uso de final sirve para convertir una variable local en una constante local, es decir que su valor ya no podrá ser cambiado.
2. **Final para atributos:** Es muy parecido al caso anterior ya que nos permitirá convertir un atributo a una constante, debido a esto no será posible generar un setter para dicho atributo.

3. **Final para métodos:** El método al que se le aplique final no podrá ser sobreescrito por otras clases que lo hereden.
4. **Final para clases:** Nos sirve para determinar que una clase ya no puede tener subclases, es decir que otras clases no podrán heredar nunca de la clase final.

🔗 **Link al código**

---

5. Exponer un código con multicatch, try with resources.

🔗 **Link al código**

---

6. Explica la diferencia entre los procesos síncronos y asíncronos.

🛑 **Procesos Síncronos**

El programa realiza una acción y no puede empezar otra hasta que termina con la primera, es decir, no puede llevar varios procesos de forma simultánea.

✅ **Procesos Asíncronos**

De forma asíncrona sí se pueden llevar a cabo varios procesos a la vez, lo cual vuelve mucho más rápida la ejecución del código (especialmente en programas complejos).

Ejemplo:

Imagina que estás preparando un pastel y debes hornear la masa por una hora:

Si tu proceso fuera **síncrono** deberías esperar hasta que el pastel haya terminado de hornearse para poder continuar con el proceso de elaboración,

> Si tu proceso fuera **asíncrono** podrías aprovechar el tiempo de horneado para preparar el

a pesar de que los otros pasos no dependan del horneado.

> relleno y el betún, de forma que puedas comenzar a decorar en cuanto el pastel haya salido del horno.

---

7. Realiza un código utilizando Lambdas

🔗 **Link al código - Ejemplo con Lambdas**

🔗 **Link al código - Ejemplo con Functions**