Ejercicios de repaso dados en clase - Pre Parcial

1. Refactorizar el siguiente fragmento de código

```
...SomeClass...{
...
bool wasInitialized(){
      return...
}

someFunction... {
if ((platform.toUpperCase().indexOf("MAC") > -1)
      && (platform.toUpperCase().indexOf("IE") > -1)
      && wasInitialized()
      && resize > 0) {
  someCode();
}
otherCode();
}

{}

}
```

Respuesta:

```
abstract class Plataforma{
boolean wasInitialized()
void someFunction(){
            someCode();
}
}

public class MAC extends Plataforma {}

public class IE extends Plataforma {}

public class Other extends Plataforma {

      @Override
      void someFunction{
            otherCode()
      }
}
```

2. Nombre las ceremonias de Scrum y detalle brevemente cada una.

- Sprint Planning
- Daily Scrum
- Sprint Review (demo al final del sprint para mostrar al cliente el resultado)
- Retro (equipo)
- Backlog Refinement

Ceremonia	Propósito	Asistentes	Consejos y Trucos
Planificación del Sprint	Identificar los objetivos del sprint y crear el backlog del sprint.	Todo el equipo Scrum	Fomentar que los miembros del equipo discutan y negocien elementos relacionados con el objetivo del sprint.
Scrum Diario	Facilitar que el equipo Scrum discuta el progreso y anuncie compromisos diarios.	Scrum Master y equipo de desarrollo	No permitir que la reunión exceda los 15 minutos.
Revisión del Sprint	Mostrar el trabajo completado durante el sprint.	Todo el equipo Scrum con Product Manager, stakeholders y clientes	Capturar retroalimentación accionable como elementos en el backlog.
Retrospectiva del Sprint	Permitir que el equipo Scrum inspeccione y planifique mejoras para el próximo sprint.	Scrum Master y equipo de desarrollo	Asegurarse de que se capturen, asignen y hagan seguimiento a las sugerencias accionables.

3. Explique las ventajas del uso del modelo de vistas de 4+1 para una arquitectura de software, haga un ejemplo de una posible vista de despliegue.

4. ¿Qué es refactoring? Como se asegura de lograrlo. De un ejemplo simple con código.

La refactorización es la modificación del comportamiento de caja negra del código sin modificar la caja blanca, es decir, modificar la forma en la que actúa internamente sin que devuelva algo diferente.

5. ¿Qué es un code smell? Nombre dos, y ejemplifíquelos con código

Un code smell es una convención, es una pieza de código que se sabe con solo verla que es errónea, porque se conoce con anterioridad. Sería similar a un antipatrón, pero en lo micro.

6. ¿Qué diferencia un token tradicional de un token JWT? Comente las ventajas y desventajas.

El JWT está firmado de una forma particular que solo el servidor sabe. Por lo que la forma que tiene de chequear si el token fue modificado sin permiso es verificar que la firma no haya sido alterada. Si alguien quiere hacer una modificación, va a poder, pero no va a poder replicar la firma en el token. Y es ahí cuando le salta la ficha.
El uso de JWT incrementa la eficiencia en las aplicaciones evitando hacer múltiples llamadas a la base de datos. No es necesario contrastarlo contra nada, solo verificar la firma

7. ¿Qué significa autenticar y que autorizar? De 3 ejemplos diferentes de cómo podría autenticarse a un usuario en un sistema.

Autenticación es ver ¿quién sos?
Autorización es ver ¿qué podes hacer?

- Autenticación: Verifica la identidad del usuario (¿Quién eres?). Es el paso en el que se confirma si alguien es quien dice ser.
- Autorización: Determina los permisos que tiene el usuario autenticado (¿Qué puedes hacer?).

Se puede autorizar sin autenticar. El ejemplo que se nos dio en clase es cuando en un edificio te dan la tarjeta para moverte libremente. El de la entrada te autentica y después cuando te ven la tarjeta, te autorizan sin necesidad de autenticarte.

3 ejemplos:
- Algo que sos: huella digital / face id
- Algo que tenés: código mfa virtual & hardware
- Algo que sabés: el nombre de tu primer mascota, el club del que sos hincha.

8. ¿Cuáles son los síntomas de un mal diseño? Detalle brevemente cada uno.

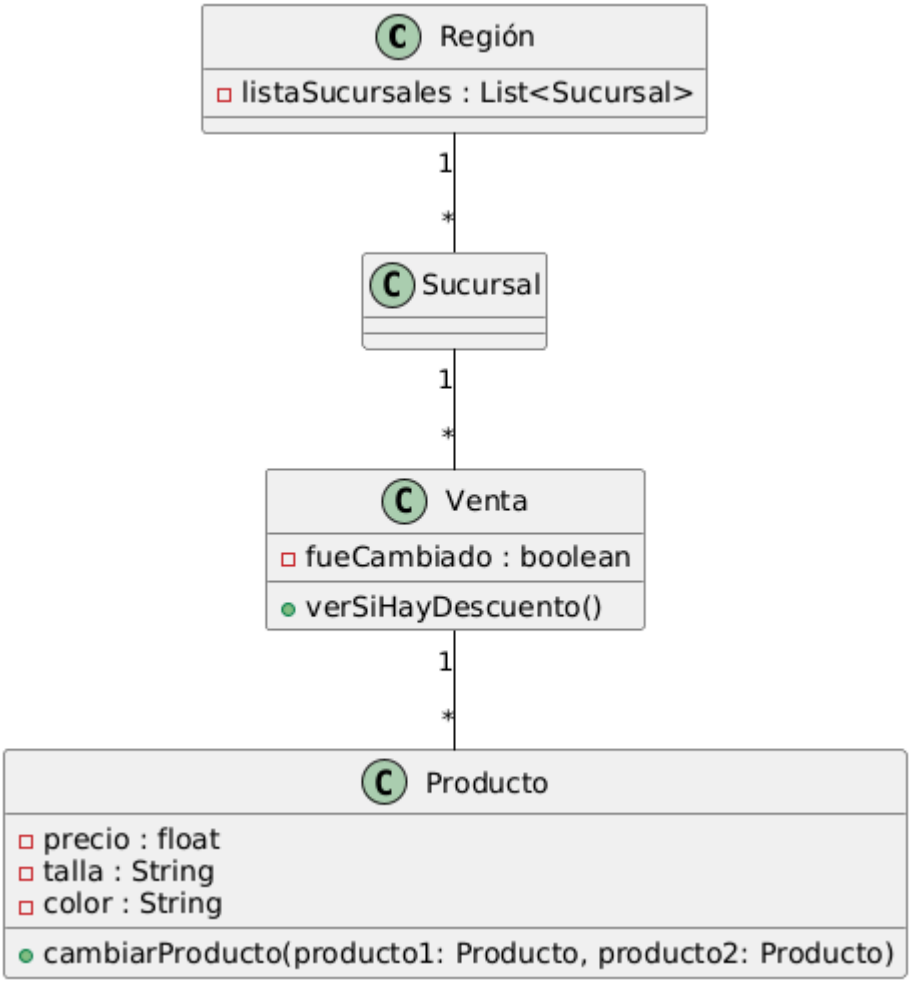
- Rigidez: quiero modificar algo y tengo que cambiar cosas en 35 lugares
- Fragilidad: toco algo de algun lado y se rompe algo en otro
- Inmovilidad: quiero mover ciertas funcionalidades de un lado a otro y no puedo. Está todo demasiado acoplado

9. ¿Qué es la inversión de dependencia? Explique y dé un ejemplo de código.

Es un principio SOLID. Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones. Las abstracciones no deberían depender de detalles (implementaciones concretas). Los detalles deberían depender de abstracciones.

10. Una tienda de múltiples sucursales vende ropa, manteniendo estadísticas de venta por región y producto.
La tienda permite cambios de productos, el cambio de un producto se realiza personalmente en la tienda donde se compró originalmente.
Cada producto puede cambiarse una única vez, ya sea por otros productos diferentes cuyo precio total sea de mayor valor, o por el mismo en otro color o talla.
Cualquier compra realizada de más de 10 productos de un mismo tipo, aplica un 10% de descuento.

a) Realice un modelo de dominio



Región tiene lista de sucursales
Sucursal tiene lista de ventas
Venta(fueCambiado) -> Sucursal y Lista[Producto]
-> calcularPrecioFinal()
Producto(precio, talla, color) -> cambiarProducto(producto)

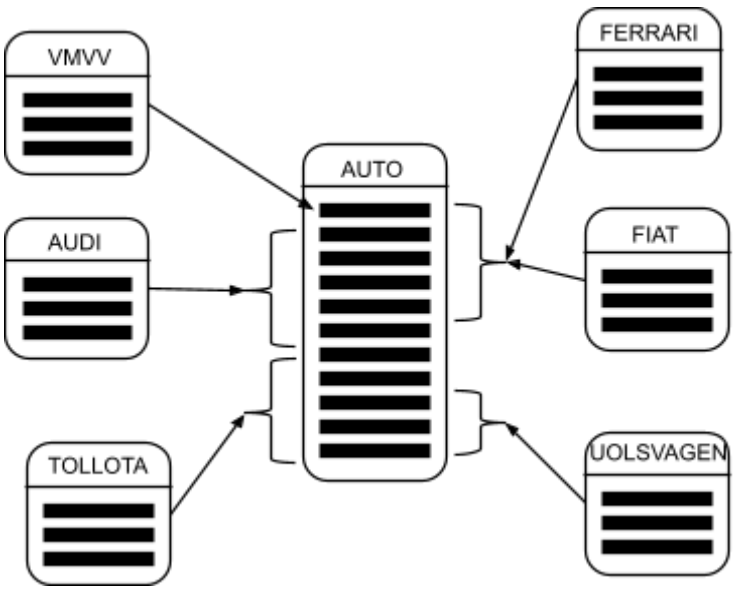
11. ¿Se viola algún/os principio SOLID? En caso afirmativo indicar cuál/es, y cómo lo resolvería.

a)
```
public class MileageCalculator {
        private List<Car> cars;
        public MileageCalculator(List<Car> cars) {
            this.cars = cars;
        }
        public void CalculateMileage() {
            for (Car car : cars) {
                if (car.name.equals("Audi")) {
```

```
                    System.out.println("Mileage of the car " +
    car.name + " is 10M");
                } else if (car.name.equals("Mercedes")) {
                    System.out.println("Mileage of the car" +
    car.name + "is 20M");
                }
            }
        }
    }
```

- Single Responsibility Principle: para imprimir el resultado (el print por pantalla). debería encargarse solamente de devolver el valor de su kilometraje.
- Open/Closed Principle: Si llega a haber una marca más de auto no va a funcionar el código actual. Hay que hacerlo en base a abstracciones. Usar polimorfismo con interfaces.
- Liskov Substitution Principle: No hay herencia de clases, por lo que no aplica.
- Interface Segregation Principle: No hay interfaces 😀
- Dependency Inversion Principle: Car está arriba del MileageCalculator.

b)



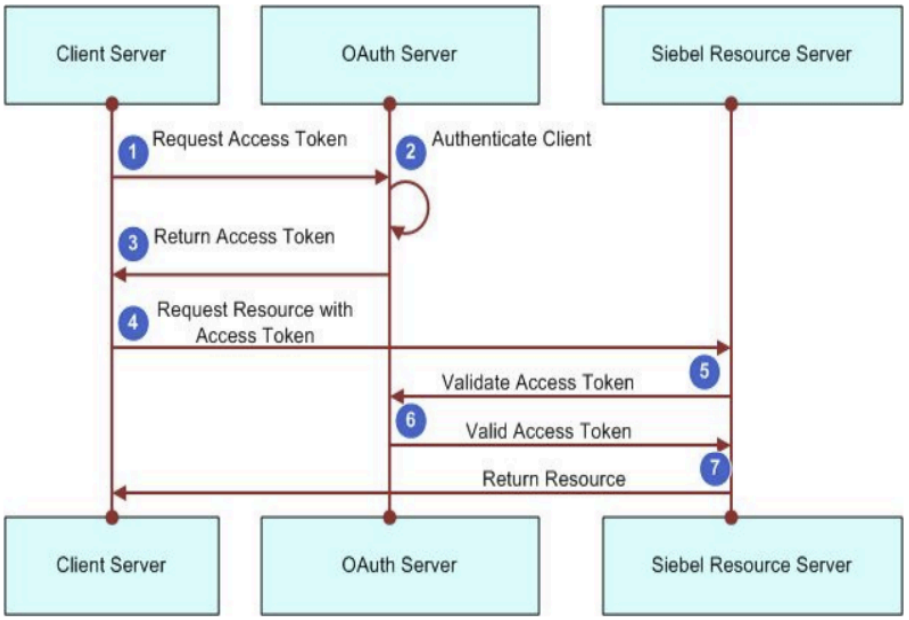
12. Explique el mecanismo de Autenticación Basic Authentication, y como lo utiliza en una API Rest

```
header {
      Authorization: {Base64: “user:password”}
}

Header= Basic aeqwtdrfhsdefrwetgdr
Base64Decoder.(HTTPRequest.getHeader(“Authorization”).getValue;
user:password
```
Basic auth encodea el user y la pass en base64. Es fácilmente decodificable y por eso se debe usar siempre en conjunto con https/sll

13. Haga un diagrama de secuencia explicando cómo se realiza un login que retorna un token JWT y como luego se accede a un servicio que requiere autenticación.

```
User -> autenticarUsuario(username, password)
Aplicación -> verificarUsuario(username, password)
```



14. A qué nos referimos por Calidad Semántica de las historias de usuario.

La Calidad Semántica de las historias de usuario se refiere a la claridad y el significado detrás de las historias que se utilizan para definir requisitos en el desarrollo de software. Implica asegurar que las historias de usuario no solo sean comprensibles,

sino que también capturen correctamente las necesidades y expectativas de los usuarios finales.

Calidad **Sintáctica**	Bien formadas	Atómicas
	Mínimas	
Calidad **Semántica**	Conceptualmente Acertadas	Orientadas al Problema
	Sin Ambigüedades	Sin Conflictos
Calidad **Pragmática**	Usan Oraciones Completas	Estimables
	Únicas	Uniformes
	Independientes	Completas

15. ¿Qué son los criterios de aceptación? ¿Qué características se desea que tengan? Escriba una historia de usuario y defina los criterios de aceptación para la misma

Son condiciones específicas que deben cumplirse para aceptar trabajo realizado
Lista de condiciones y escenarios: Dado que <contexto>, cuando suceda <evento> entonces <consecuencia>.
Estos deben ser:
- Claros
- Concisos
- Verificables
- Independientes
- Orientados al Problema
- Alcanzables

Ejemplo de Historia de Usuario:

Historia de Usuario:
Como usuario registrado, quiero poder calificar las películas que he visto, para compartir mis opiniones y ayudar a otros a decidir qué ver.

Criterios de Aceptación:
1. El usuario puede calificar una película con 1 a 5 estrellas.
2. Las calificaciones son visibles en la página de la película.
3. El usuario puede editar su calificación.
4. Un usuario solo puede calificar cada película una vez.

5. Las calificaciones se almacenan y muestran correctamente en la interfaz.