Ejercicio 3

Considere el siguiente extracto de código de un framework para construir clientes de correo electrónico, en particular el módulo de definición de filtros de correo. Este módulo del framework permite definir un filtro para correos electrónicos que siempre verifica si debe agregarse o no el correo a un inbox dado, y en caso positivo lo agrega.
Por ejemplo, podría definirse un filtro que verifique mediante el método isEmailAllowed si un correo incluye en el asunto el string "Objetos 2" y, en tal caso, lo agrega mediante el método addEmail a la bandeja de entrada marcándolo como importante.

```
1   abstract class EmailFilter{
2       private EmailInbox emailInbox;
3
4       public EmailFilter(EmailInbox emailInbox){
5           this.emailInbox = emailInbox;
6       }
7
8       public filterEmail(Email anEmail){
9           if (this.isEmailAllowed(anEmail)){
10              this.addEmail(anEmail);
11          }
12      }
13
14      public abstract Boolean isEmailAllowed(Email anEmail);
15      public abstract void addEmail(Email anEmail);
16  }
```

Responda las siguientes preguntas, basándose en el subconjunto de clases y métodos que conoce del framework:

1. *¿El comportamiento variable del framework (hotspots), está implementado mediante herencia o composición? Justifique su respuesta.*
2. *Indique la/s línea/s donde encuentra inversión de control. Justifique su respuesta.*
3. *El método isEmailAllowed, ¿es un hot spot? ¿Por qué?*

If 8% > mayo y 10%.

else 11% > Objeto y 15%.

Parcial de OO2 - Curso 2022 - 16/jul/2022

Ejercicio 1 - Patrones

Sea una estación meteorológica hogareña que permite conocer información de varios aspectos del clima. Esta estación está implementada con la clase HomeWeatherStation que interactúa con varios sensores para conocer fenómenos físicos. La misma implementa los siguientes métodos:

```
//retorna la temperatura en grados Fahrenheit
public double getTemperaturaFahrenheit()

//retorna la presión atmosférica en hPa
public double getPresion()

//retorna la radiación solar
public double getRadiacionSolar()

//retorna una lista con todas las temperaturas sensadas hasta el momento, en
grados Fahrenheit
public List<Double> getTemperaturasFahrenheit()
```

Esta clase se encuentra implementada por terceros y **no se puede modificar.**

Nos piden construir una aplicación que además de lo anteriormente descripto pueda obtener:

- La temperatura en grados Celsius (°C = (°F-32) ÷ 1.8).
- El promedio de las temperaturas históricas en grados Fahrenheit.

Además, la aplicación debe permitir al usuario configurar qué datos mostrar y en qué orden. Esto significa que podría querer ver la información de muchas maneras, por ejemplo:

- *Ejemplo 1: "Presión atmosférica: 1008"*
- *Ejemplo 2: "Presión atmosférica: 1008 Radiación solar: 500"*
- *Ejemplo 3: "Radiación solar: 500 Temperatura C: 28 Promedio de temperaturas C: 25"*

Para ello, usted debe proveer en algún punto de su solución, la implementación del mensaje public String displayData() que devuelva los datos elegidos en el orden configurado (dado que la app aun no cuenta con interface de usuario).

Haga uso de la clase HomeWeatherStation sin modificarla.

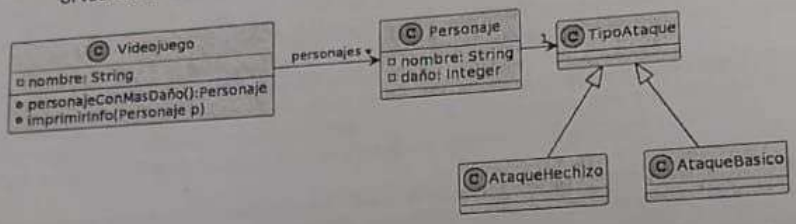
Tareas:
1- Modele una solución para el problema planteado. Si utiliza algún patrón, indique cuál
2- Implemente en Java
3- Implemente un test para validar la configuración del ejemplo 2, asumiendo que en el momento de la ejecución del mismo, los sensores arrojan los valores del ejemplo.

Ejercicio 2 - Refactoring

Para el siguiente código, realice las siguientes tareas:

(i) indique qué mal olor presenta
(ii) indique el refactoring que lo corrige
(iii) aplique el refactoring mostrando únicamente el código que cambió, detallando cada paso intermedio.

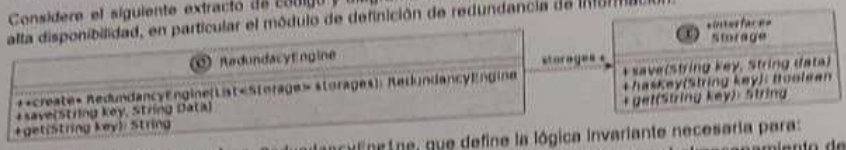
Si vuelve a encontrar un mal olor, retorne al paso (i).




```
public class Videojuego{
    // ...
    //
public Personaje personajeConMasDaño() {
      Personaje temp = null;
      double max= 0;
      for (Personaje p : personajes) {
            double daño = p.getTipoAtaque().calcularDaño(p.getDaño());
            if (daño > max){
                  temp = p;
                  max = daño;
              }
       }
       return temp;
 }

public void imprimirInfo(Personaje p){
     System.out.println(p.getNombre() + "tiene como daño " + p.getDaño());
     if (p.getTipoAtaque().getClass() == AtaqueHechizo.class) {
          System.out.println("Ataque tipo hechizo");
          System.out.println("Este ataque dobla tu fuerza");
    } else{
          System.out.println("Ataque tipo Ataque Básico");
          System.out.println("Este ataque mantiene tu fuerza");
          }
}
```

Ejercicio 3 - Frameworks

Considere el siguiente extracto de código y diagrama de clases UML de un framework para sistemas de alta disponibilidad, en particular el módulo de definición de redundancia de información.



Este framework provee una clase, `RedundancyEngine`, que define la lógica invariante necesaria para: (i) almacenar la información en más de un espacio de almacenamiento, esto es, el almacenamiento de cada clave/valor en todos los *Storage* provistos, y,
(ii) recuperar el valor correspondiente a una clave desde el primer storage que posea dicha información.
Por otro lado, la interface *Storage*, también provista por el framework, define el protocolo necesario para que `RedundancyEngine` pueda almacenar y recuperar la información solicitada.

Es responsabilidad de las personas que utilizan el framework definir el comportamiento específico para almacenar la información en, por ejemplo, un archivo, en memoria, o cualquier otro soporte. Siempre mediante la implementación de la interface *Storage*.
Considerando que, como usuarios/as del framework, ya hemos definido dos clases que implementan la interface *Storage* llamadas `FileStorage` e `InMemoryStorage`, deberíamos usar `RedundancyEngine` para acceder y almacenar información de la siguiente manera:

```
List<Storage> storages = new ArrayList<Storage>();
storages.add(new InMemoryStorage(...));
storages.add(new FileStorage(...));

RedundancyEngine redundancyEngine = new RedundancyEngine(storages);

redundancyEngine.save("name", "Diego Maradona");

// La linea siguiente deberia retornar el String "Diego Maradona"
redundancyEngine.get("name");

public class RedundancyEngine{
    List<Storage> storages;
    public RedundancyEngine(List<Storage> storages){
        super();
        this.storages = storages;
    }
    public void save(String key, String data){
        for (InformationStorage storage : this.getStorages()){
            storage.save(key, data)
        }
    }
    public String get(String key){
        for (InformationStorage storage : this.getStorages()){
            if (storage.hasKey(key)){
                return storage.get(key);
            }
        }
        return null;
    }
}
public interface Storage{
    public void save(String key, String data);
    public Boolean hasKey(String key);
    public String get(String key);
}
```

Responda las siguientes preguntas, basándose en el subconjunto de clases y métodos del framework presentado anteriormente:

1. ¿El comportamiento variable del framework (hotspots), está implementado mediante herencia o composición? Justifique su respuesta.
2. ¿Cuáles son los hook methods?
3. ¿Cuál es el Frozen Spot?

Ejercicio 2

Para el siguiente código, realice las siguientes tareas:

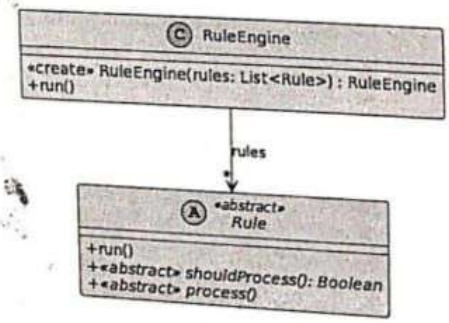
(i) indique qué mal olor presenta
(ii) indique el refactoring que lo corrige
(iii) aplique el refactoring (modifique el código)

Si vuelve a encontrar un mal olor, retorne al paso (i).

```
class SuperMarioBros extends Game
{
    public initialize()
    { … }
    public startPlay()
    { … }
    public endPlay()
    { … }
    public play()
    {
    initialize();
    startPlay();
    endPlay();
    }
}

class DonkeyKong extends Game
{
    public init()
    { … }
    public startPlay()
    { … }
    public endPlay()
    { … }
    public jugar()
    {
    init();
    startPlay();
    endPlay();
    }
}
```

Considere el siguiente extracto de código y diagrama de clases de un framework para construir aplicaciones, en particular el módulo de definición de reglas de negocio. Éste define una clase RuleEngine, la cual debe ser creada con una lista de instancias de la clase Rule y ejecutada invocando al mensaje run(). Esta clase siempre ejecutará todas las reglas con las que fue inicializada, enviándole a cada una de ellas el mensaje run(). El framework también provee una clase abstracta Rule, la cual debe ser subclasificada por la persona que utilice el framework, proveyendo la implementación de los métodos shouldProcess() y process(); toda regla se procesará siempre y cuando la respuesta de shouldProcess() sea verdadera.

```
public class RuleEngine{
    List<Rule> rules;
    public RuleEngine (List<Rules> rules){
        super();
        this.rules = rules;
    }
    public void run(){
        for (Rule rule : this.rules){
            rule.run()
        }
    }
}

abstract class Rule{
    public void run(){
        if (this.shouldProcess()){
            this.process();
        }
    }

    public abstract Boolean shouldProcess();
    public abstract void process();
}
```



Responda las siguientes preguntas, basándose en el subconjunto de clases y métodos que conoce del framework:

1. Dado que para utilizar este framework usted tiene que implementar una subclase de Rule, la ejecución del código de esta subclase, ¿se realiza mediante inversión de control? Justifique su respuesta de forma concisa.
2. ¿Cuáles son los hook methods?
3. Describa, de forma concisa, el frozen spot del extracto del framework presentado.

gente, que responderían en el primer punto??

Orientación a Objetos II - 10/6/2023

Ejercicio 1 - Patrones

Se quiere implementar una aplicación para seguros de vehículos. Cada seguro estará asociado a un vehículo en específico, y una persona podrá contar con múltiples vehículos asegurados. La compañía brinda 3 tipos de seguros.Cada seguro incluye una cobertura de vida, una cobertura de daños a otros vehículos y una cobertura de daños al vehículo propio, según se muestra a continuación:

Seguro contra terceros	Seguro terceros completo	Seguro todo riesgo
Costo del seguro de vida: contempla solo el **seguro del conductor**: **100** pesos multiplicado por la edad del **asegurado**.	Costo del seguro de vida: Al **seguro del conductor** se le suma un monto de **5.000** pesos multiplicado por la cantidad máxima de ocupantes que posee el vehículo	Costo del seguro de vida: Al **seguro del conductor** se le suma un monto de **9.000** pesos multiplicado por la antigüedad la antigüedad del vehículo asegurado
Costo de la cobertura de daños a otros vehículos: **1.000** pesos más el 1% del valor del vehículo.	Costo de la cobertura de daños a otros vehículos: **4.000** pesos multiplicado por la antigüedad del auto si el auto tiene más de 4 años o **10.000** pesos si el auto tiene hasta 4 años.	Costo de la cobertura de daños a otros vehículos: **100.000** pesos dividido la edad del conductor.
Costo de la cobertura de destrucción total del vehículo propio por accidente: **0.5%** del valor del auto asegurado	Costo de la cobertura de destrucción total del vehículo propio por accidente o incendio: **0.5%** del valor del auto asegurado + **10.000** pesos	Costo de la cobertura de destrucción total o parcial del vehículo propio por accidente o incendio: **0.5%** del valor del auto asegurado + **1.000** pesos por la antigüedad del auto

La empresa cuenta con promociones disponibles para sus asegurados, pero planea agregar nuevas promociones en un futuro. Las promociones son: (i) promoción por múltiples pólizas, en donde se le descuenta al asegurado un 10% a cada una de los seguros contratados, si tiene al menos 2 pólizas, (ii) promoción por campaña excepcional, se le descuenta el 50% del costo del seguro más económico que tiene contratado y (iii) sin promoción, es decir, no se le realiza ningún descuento.

A los nuevos asegurados, la empresa asigna una promoción específica. Sin embargo, la empresa puede cambiarla en cualquier momento.

Usted debe brindar una solución para calcular el monto a abonar correspondiente a un asegurado. Este costo final es la suma de los costos de sus seguros aplicando la bonificación por la promoción que posea.

***Ayuda**: Para calcular los años entre dos fechas puede utilizar la siguiente expresión*

```
ChronoUnit.YEARS.between(fecha1, fecha2);
```

Tareas:

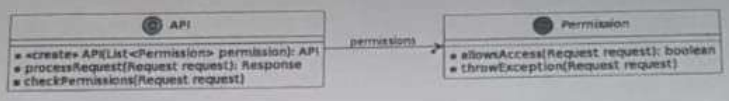
1. Modele una solución para el problema planteado. Si utiliza patrones, indique nombre y las ventajas que agrega en este diseño en particular
2. Implemente en Java la funcionalidad requerida.
3. Implemente un **test** para el siguiente escenario. Un asegurado de 50 años cumplidos, no tiene promoción, y asegura dos autos con seguro contra Terceros: un Renault 11 año 1988 valuado en 680.000 pesos, con capacidad de 4 personas y un Renault Clio año 2001 valuado en 1.200.000 pesos con capacidad de 5 personas.

Notas:

- Implemente **todos** los constructores que considere necesarios.
- Si necesita getters y setters, puede implementar uno de cada uno y asumir la existencia del resto.

Ejercicio 3: Frameworks

Considere el siguiente extracto de código y diagrama de clases UML de un framework para desarrollo de API Rests, en particular el módulo de autorización de acceso.



Este framework provee una clase, API, que define la lógica invariante necesaria para:
(i) procesar un request HTTP, siempre verificando que se cumplan con todos los permisos de acceso, y,
(ii) retornar el estado HTTP 200 OK para el caso en que se cumplan todos los permisos de acceso o retornar el estado HTTP 403 FORBIDDEN en caso de que alguno de los permisos no se cumpla.

Es responsabilidad de quienes utilizan el framework implementar la interfaz *Permission* para definir el comportamiento específico para chequear los permisos de accesos (por ejemplo, para cualquier usuario registrado, sólo para usuarios administradores, etc). Eso incluye también la lógica para disparar AccessDeniedException o alguna subclase de ésta.

```
public class API {
  private List<Permission> permissions;

  public API(List<Permission> permissions){
    super();
    this.permissions = permissions;
  }

  public Response processRequest(Request request){
    ...
    try{
      this.checkPermissions(request);
    }
    catch (AccessDeniedException e){
      return new Response(HTTPStatus.403);
    }
    return new Response(HTTPStatus.200);
  }

  private void checkPermissions(Request request) throws AccessDeniedException{
    for (Permission permission: this.getPermissions()){
      if (!permission.allowsAccess(request)){
        permission.throwException(request);
      }
    }
  }
}
```

```
public interface Permission{
  public boolean allowsAccess(Request request);
  public void throwException(Request request) throws AccessDeniedException;
}
```

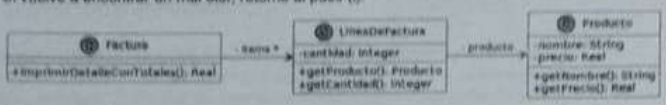
Tareas:
Responda las siguientes preguntas, basándose en el subconjunto de clases y métodos del framework presentado anteriormente.

1. ¿El comportamiento variable del framework (hotspots), está implementado mediante herencia o composición? Justifique su respuesta.
2. ¿Observa hook methods? ¿Cuáles?
3. ¿Qué parte de lo que se dice anteriormente respecto al framework se corresponde con el frozenspot?
4. En lo que se describe anteriormente y lo que se indica debe hacer quien utiliza el framework, ¿observa inversión de control? ¿donde?

Ejercicio 2 - Refactoring

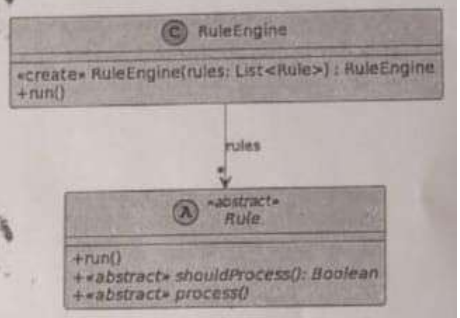
En forma iterativa, realice los siguientes pasos:

(i) indique el mal olor,
(ii) indique el refactoring que lo corrige,
(iii) aplique el refactoring (modifique el código),
(iv) asegúrese de que los tests provistos corran exitosamente.
Si vuelve a encontrar un mal olor, retorne al paso (i).




```
public class Factura{
    private List<LineaDeFactura> items;

    public void imprimirDetalleConTotales() {

        String message = "Detalle de los items facturados";

        Iterator<LineaDeFactura> lineaDeFacturaIterator = items.iterator();
        while (lineaDeFacturaIterator.hasNext()) {
          LineaDeFactura lineaDeFactura = lineaDeFacturaIterator.next();
          message = message + String.format("Producto: %s",
          lineaDeFactura.getProducto().getNombre())
          + String.format(" Cantidad: %s", lineaDeFactura.getCantidad())
          + String.format("Precio unitario: %s", lineaDeFactura.getProducto().getPrecio())
          + String.format("Total por producto: %s", lineaDeFactura.getCantidad() *
                          lineaDeFactura.getProducto().getPrecio())

        message = message + "Total sin impuestos";
        double totalSinImpuestos = items.stream().mapToDouble(lineaDeFactura ->
  lineaDeFactura.getProducto().getPrecio() * lineaDeFactura .getCantidad()).sum();
        message = message + String.format("Total: %s", totalSinImpuestos);

        message = message + "Total con IVA";
        message = message + String.format("Total: %s", totalSinImpuestos*1.21);

        System.out.println(message);
    }
}

public class LineaDeFactura{
    private Producto producto;
    private int cantidad;

    public Producto getProducto() {
        return this.producto;
    }

    public int getCantidad() {
        return this.cantidad;
    }
}

public class Producto {
    private String nombre;
    private double precio;

    public double getPrecio() {
        return this.precio;
    }
    public String getNombre() {
        return this.nombre;
    }
}
```

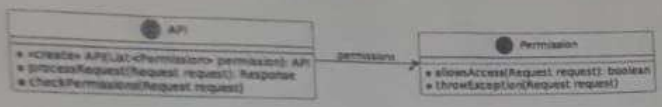
```
public class RuleEngine{
    List<Rule> rules;
    public RuleEngine (List<Rules> rules){
        super();
        this.rules = rules;
    }
    public void run(){
        for (Rule rule : this.rules){
            rule.run()
        }
    }
}

abstract class Rule{
    public void run(){
        if (this.shouldProcess()){
            this.process();
        }
    }

    public abstract Boolean shouldProcess();
    public abstract void process();
}
```



aca la inversión de control se da en e

Ejercicio 3: Frameworks

Considere el siguiente extracto de código y diagrama de clases UML de un framework para desarrollo de API Rests, en particular el módulo de autorización de acceso.



Este framework provee una clase, API, que define la lógica invariante necesaria para
(i) procesar un request HTTP, siempre verificando que se cumplan con todos los permisos de acceso, y,
(ii) retornar el estado HTTP 200 OK para el caso en que se cumplan todos los permisos de acceso o retornar el estado HTTP 403 FORBIDDEN en caso de que alguno de los permisos no se cumpla.

Es responsabilidad de quienes utilizan el framework implementar la interfaz *Permission* para definir el comportamiento específico para chequear los permisos de accesos (por ejemplo, para cualquier usuario registrado, sólo para usuarios administradores, etc). Eso incluye también la lógica para disparar AccessDeniedException o alguna subclase de ésta.

```
public class API {
  private List<Permission> permissions;

  public API(List<Permission> permissions){
    super();
    this.permissions = permissions;
  }

  public Response processRequest(Request request){
    ...
    try{
      this.checkPermissions(request);
    }
    catch (AccessDeniedException e){
      return new Response(HTTPStatus.403);
    }
    return new Response(HTTPStatus.200);
  }

  private void checkPermissions(Request request) throws AccessDeniedException{
    for (Permission permission: this.getPermissions()){
      if (!permission.allowsAccess(request)){
        permission.throwException(request);
      }
    }
  }
}

public interface Permission{
  public boolean allowsAccess(Request request);
  public void throwException(Request request) throws AccessDeniedException;
}
```

Tareas:
Responda las siguientes preguntas, basándose en el subconjunto de clases y métodos del framework presentado anteriormente.

1. ¿El comportamiento variable del framework (hotspots), está implementado mediante herencia o composición? Justifique su respuesta.
2. ¿Observa hook methods? ¿Cuáles?
3. ¿Qué parte de lo que se dice anteriormente respecto al framework se corresponde con el frozenspot?
4. En lo que se describe anteriormente y lo que se indica debe hacer quien utiliza el framework, ¿observa inversión de control? ¿donde?

En el exámen que nos tomaron por ejemplo puse que había en 3 lugares por eso mismo, eso había marcado, no sé que tan bien esto porque no fui a ver la muestra, pero al menos aprobé.... Supongo que aprobé porque sabía que era composición, pero no sé si esa parte puede estar BN o mal

Ejercicio 3 - Frameworks

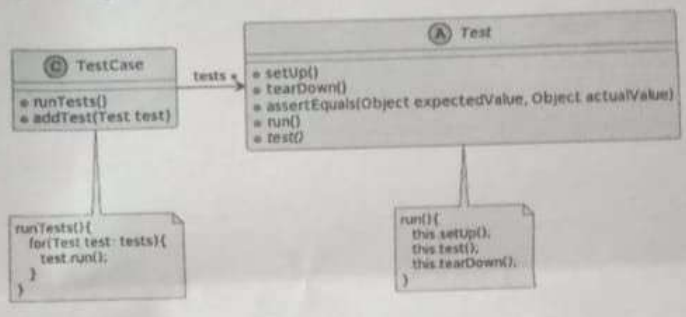
OO2 Parcial 2da fecha - 1/7/2023

Consideremos un framework muy simple para automatizar pruebas de software.

El framework permite crear casos de prueba. Cada caso de prueba ("test case" en inglés) incluye una o varias pruebas ("test" en inglés). Al correr un caso de prueba, se corren una a una, en secuencia, todas las pruebas incluidas en el caso. El orden en el que se ejecutan las pruebas es aleatorio. Si una prueba falla (es decir, no pasa) el caso se interrumpe. Un caso de pruebas pasa si pasan todas las pruebas que incluye. Las pruebas pueden requerir preparación (set up, en inglés) y desarmado (tear down en inglés). El framework se asegura de que la preparación (si existe) y el desarmado (si existe) se efectúan antes y después de cada test. El framework ofrece un conjunto fijo de asserts (chequeos) que pueden utilizarse en las pruebas.

El programador puede definir nuevas pruebas, puede combinar pruebas en casos de manera arbitraria, y puede correr los casos. Al definir una prueba, el programador determina cómo hacer la preparación (si es necesaria), los pasos de la prueba (usando los asserts que provee el framework), y el desarmado (si es necesario).

Considere el siguiente diagrama de clases que documenta el framework.



Para crear programas (casos de prueba) con el framework, el programador debe:

1. Programar una subclase de Test por cada prueba. En esa subclase debe programar el método abstracto test() utilizando el método assertEquals() heredado, el cual dispara una excepción en caso de recibir valores diferentes, haciendo que la prueba y todo el caso de prueba fallen. Si lo considera necesario, además, puede implementar los métodos setUp() y tearDown(), para hacer algo para preparar y/o desarmar el test.
2. Crear instancias de TestCase y agregarle tests (instancias de sus clases concretas de Test).
3. Ejecutar los casos enviándoles el mensaje runTests()

Tareas:
Responda las siguientes preguntas, basándose en el subconjunto de clases y métodos del framework presentado anteriormente:

1. Indique, para cada uno de los siguientes ítems, si es parte del frozen spot, si es parte de algún hotspot o si no corresponde a ninguno de los dos.
   a. El framework se asegura de que la preparación y el desarmado se efectúan antes y después de cada test.
   b. Al definir una prueba, el programador determina cómo hacer la preparación (si es necesaria).
   c. La clase Test es abstracta.
   d. Al correr un caso de prueba se corren todas las pruebas incluidas en el caso.
   e. Al definir una prueba el programador define los pasos de la prueba (usando los asserts que provee el framework).
2. ¿Observa métodos gancho? ¿Cuáles?
3. ¿Observa inversión de control? ¿Dónde?
4. ¿Caracterizaría el framework como caja blanca o caja negra? ¿Por qué?

Parcial de OO2 - 2022 - 8/ago/2022

Consideremos una empresa que brinda servicios y los gestiona a través de proyectos. Los proyectos tienen una fecha de inicio y de fin, un objetivo, un número de integrantes (quienes cobran un monto fijo por día) y un margen de ganancia. Durante el armado del proyecto, el mismo debe pasar por un proceso de aprobación que involucra las etapas: En construcción -> En evaluación -> Confirmada. Se desea implementar la siguiente funcionalidad:

Funcionalidad	Etapa del proyecto	Resultado esperado
Crear proyecto	-	Se crea el proyecto en etapa "En construcción" con nombre, fecha de inicio y fin, objetivo, margen de ganancia de 7%, un número de integrantes y el monto de pago por integrante por día.
Aprobar etapa	En construcción	El proyecto pasa a etapa "En evaluación" siempre y cuando su precio no sea 0 (cero). De lo contrario genera un error.
	En evaluación	El proyecto pasa a etapa "Confirmada"
	En otra situación	No produce efecto alguno en el proyecto.
Costo del proyecto	En cualquier etapa	Retorna la suma de los costos de las personas involucradas. Considerar que las personas trabajan todos los días que dura el proyecto.
Precio del proyecto	En cualquier etapa	Retorna el valor obtenido luego de aplicar el margen de ganancia al costo del proyecto.
Modificar margen de ganancia	En etapas "En construcción" y "En evaluación"	Actualiza el margen de ganancia si se encuentra en los siguientes valores: Para "En construcción" -> valores entre 8% y 10% Para "En evaluación" -> valores entre 11% y 15% Para valores fuera de los rangos permitidos no produce efecto alguno en el proyecto.
	Otra situación	No produce efecto alguno en el proyecto.
Cancelar proyecto	En cualquier etapa	Agrega "(Cancelado) " al objetivo del proyecto. Deja el proyecto cancelado.
	Si ya está Cancelado.	No produce efecto alguno en el proyecto.

Tareas:

1- Modele una solución y provea el diagrama de clases UML para el problema planteado. Si utiliza algún patrón, indique cuál.

2- Implemente en Java.

3- Implemente un test para aprobar un proyecto con las siguientes características: (i) se encuentra en evaluación, (ii) se llama "Vacaciones de invierno", (iii) tiene como objetivo "salir con amigos", y (iv) lo integran 3 personas.

Nota: para generar o levantar un error debe utilizar la expresión

```
throw new RuntimeException("Este es mi mensaje de error");
```