Parcial de Orientación a Objetos II
Primera fecha - 25/06/2022

Ejercicio 1 - Patrones

Sea una aplicación que ofrece excursiones como por ejemplo "dos días en kayak bajando el Paraná". Una excursión posee nombre, fecha de inicio, fecha de fin, punto de encuentro, costo, cupo mínimo y cupo máximo.

La aplicación ofrece las excursiones pero éstas sólo se realizan si alcanzan el cupo mínimo de inscriptos. Un usuario se inscribe a una excursión y si aún no se alcanzó el cupo mínimo, la inscripción se considera provisoria. Luego, cuando se alcanza el cupo mínimo, la inscripción se considera definitiva y podrá llevarse a cabo. Finalmente, cuando se alcanza el cupo máximo, la excursión solo registrará nuevos inscriptos en su lista de espera.

De los usuarios inscriptos, la aplicación registra su nombre, apellido e email.

Por otro lado, en todo momento la excursión ofrece información de la misma, la cual consiste en una serie de datos que varían en función de la situación.

- Si la excursión no alcanza el cupo mínimo, la información es la siguiente: nombre, costo, fechas, punto de encuentro, cantidad de usuarios faltantes para alcanzar el cupo mínimo.
- Si la excursión alcanzó el cupo mínimo pero aún no el máximo, la información es la siguiente: nombre, costo, fechas, punto de encuentro, los mails de los usuarios inscriptos y cantidad de usuarios faltantes para alcanzar el cupo máximo.
- Si la excursión alcanzó el cupo máximo, la información solamente incluye nombre, costo, fechas y punto de encuentro.

En una primera versión, al no contar con una interfaz de usuario y a los efectos de *debugging,* este comportamiento puede implementarlo en un método que retorne un String con la información solicitada.

Tareas:

1.- Realice un diseño UML. Si utiliza algún patrón indique cuál(es) y justifique su uso.

2.- Implemente lo necesario para instanciar una excursión y para instanciar un usuario.

3.- Implemente los siguientes mensajes de la clase Excursion:

(i) public void inscribir (Usuario unUsuario)

(ii) public String obtenerInformacion().

4.- Escriba un test para inscribir a un usuario en la excursión "Dos días en kayak bajando el Paraná", con cupo mínimo de 1 persona y cupo máximo 2, con dos personas ya inscriptas. Implemente todos los mensajes que considere necesarios.

Ejercicio 2 - Refactoring

Dado el siguiente código, **solamente para el método comprar de la clase Cliente**, 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 Cliente {

        private String nombre;
        private String tipo;
        private List<Compra> compras;

        public Cliente(String unNombre) {
                this.nombre = unNombre;
                this.tipo = "basico";
                this.compras = new ArrayList<Compra>();
        }

        public Compra comprar(List<Producto> productos) {
                double temp1 = 0;
                if (this.tipo.equals("basico")) {
                        temp1 = 0.1;
                } else if (this.tipo.equals("premium")) {
                        temp1 = 0.05;
                } else if (this.tipo.equals("advance")) {
                        temp1 = 0;
                }
                double subtotal = productos.stream().mapToDouble(p -> p.getPrecio()).sum();
                double costoEnvio = subtotal * temp1;
                Compra n = new Compra(productos, subtotal, costoEnvio);
                this.compras.add(n);

                if (this.montoAcumuladoEnCompras() > 10000) {
                        this.tipo = "advance";
                } else if (this.montoAcumuladoEnCompras() > 5000) {
                        this.tipo = "premium";
                }
                return n;
        }
        public double montoAcumuladoEnCompras() {...}
}

public class Compra {
        private List<Producto> productos;
        private double subtotal;
        private double envio;
        private String estado;
}

public class Producto {
        private String descripcion;
        private double precio;
}
```

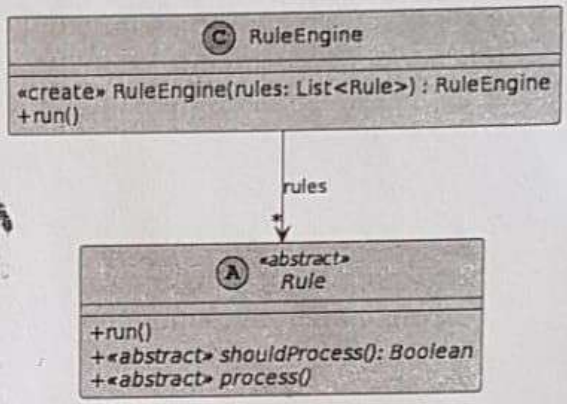
Ejercicio 3 - Frameworks

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.