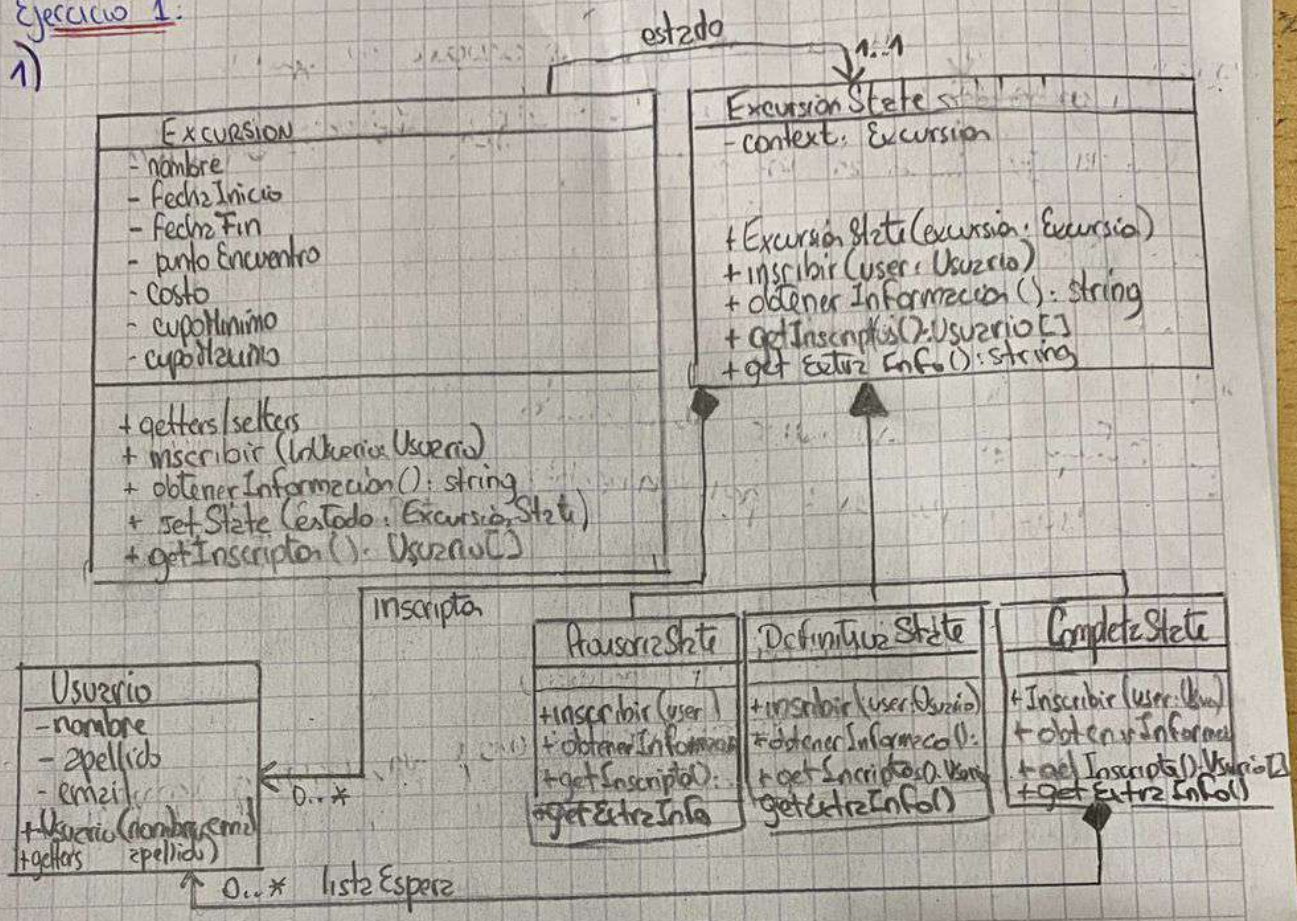
Ramero Matías 82187 | POO 2 - Primera Fecha | HOJA Nº 1/3 | FECHA 25/06/22

Ejercicio 1:

1)



Utilicé el patron State ya que hay comportamiento de la excursión que depende de estado interno en el que esté, y también está definida la secuencia en que cambian estos estados, segun diferentes condiciones. También Template Method, ya que hay código en comun en la jerarquía y basteria con especificar una parte en cada clase concreta.

2) package ar2.parcial;
3) import *;

```
public class Usuario {
    private string nombre;
    private string apellido;
    private string email;
    public Usuario(string nombre, string apellido, string email) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.email = email;
    }
```

NOTA

```
    public string getNombre() {
        return this.nombre;
    }

    public void setNombre(string nombre){
        this.nombre = nombre;
    }

    /* setters getters */
}

public class Excursion {
    private ExcursionStateI estado;
    private string nombre;
    /* otros fields */

    public Excursion () {
        this.estado = new AvailableState(this);
    }

    public void inscribir(Usuario unUsuario) {
        this.estado.inscribir(unUsuario);
    }

    public string obtenerInformacion() {
        return this.estado.obtenerInformacion();
    }

    public void setState(ExcursionStateI estado) {
        this.estado = estado;
    }

    public List<Usuario> getInscripta() {
        return this.estado.getInscripta();
    }
}

public class ExcursionState {
    private Excursion context;
    private List<Usuario> inscripta;
    public ExcursionState(Excursion context) {
        this.context = context;
        this.inscripta = new ArrayList<Usuario>();
    }

    public void InscribirUsuario(Usuario user) {
    }
    public string obtenerInformacion() {
        return this.context.getNombre() + "," + this.context.getCosto() + "," + this.context.getFechaInicio()
        + "," + this.context.getFechaFin() + "," + this.context.getPuntoEncuentro() + getExtraInfo();
    }
    public string getExtraInfo() {
        return "";
    }

    public List<Usuario> getInscripta() {
        return this.inscripta;
    }
```

NOTA

```
public class PromocionState extends ExcursionState {

    @Override
    public void inscribir(Usuario user) {
        this.inscripta.add(user);
        if(this.inscripta.length >= this.context.getCupoMinimo()) {
            this.context.setState(new DefinitivoState(this.context));
        }
    }

    @Override
    public String getExtraInfo() {
        return "," + (this.context.getCupoMinimo() - this.inscripta.length);
    }
}


public class DefinitivoState extends ExcursionState {

    @Override
    public void inscribir(Usuario user) {
        this.inscripta.add(user);
        if(this.inscripta.length >= this.context.getCupoMaximo()) {
            this.context.setState(new CompletoState(this.context));
        }
    }

    @Override
    public String getExtraInfo() {
        return "," + (this.context.getCupoMaximo() - this.inscripta.length);
    }
    public DefinitivoState(Excursion context) {
        super(context);
        this.inscripta.addAll(context.getInscripta());
    }
}
public class CompletoState extends ExcursionState {
    private List<Usuario> listaEspera = new ArrayList<Usuario>();
    public CompletoState(Excursion context) {
        super(context);
        this.inscripta.addAll(context.getInscripta());
    }
    @Override
    public void inscribir(Usuario user) {
        this.listaEspera.add(user);
    }
}
```

NOTA

4)
```
import *;
public class Tests {

@Test
public void InscripcionDeUsuario {

    Usuario u1 = new Usuario ("Matia", "Romero", "matia678@gmail.com");
    Usuario u2 = new Usuario ("Juan", "Perez", "email@email.com");
    Usuario u3 = new Usuario ("Estefeniz", "Barz", "email2@email.com");
    Excursion e = new Excursion ();
    e.setNombre("Dos días en kayak bajando el Paraná");
    e.setCupoMinimo(1);
    e.setCupoMaximo(2);
    e.inscribir(u1);
    e.inscribir(u2);
    e.inscribir(u3);

    String expected = "Dos día en kayak bajando el Paraná, 0,--.-";
    assertEquals(expected, e.obtenerInformacion);
}
```

Ejercicio 2

i- Metodo largo, variable temporal, condicional con polimorfismo, feature Envy

ii- Tendria que agregar polimorfismo para las condicionales, metodo que devuelve el calculo del temporal, Move; para mover logica a una nueva clase, Template Method, State.

```
public class ClienteBasico extends ClienteState {

@Override
public double getDiferencialEnvio () {
    return 0.1;
}
@Override
public void nuevoMontoAcumuladoEnCompras(double monto){
    if (monto > 15000)
        this.context.setState(new ClienteAdvance(this.context));
    }
}
public class ClientePremium extends ClienteState {
@Override
public double getDiferencialEnvio() {
    return 0,05;

@Override
public void nuevoMontoAcumuladoEnCompras (double monto){
if (monto > 10000) {
    this.context.setState(new ClienteAdvance(this.context));
}
}
```

NOTA

Romero Matas 82187 POO II - Examen

```
public class ClientAdvance extends ClientState {

    @Override
    public double getDiferencialEnvio() {
        return 0;
    }
}

public class ClientState {
    private Client context;

    public ClientState(Client context) {
        this.context = context;
    }

    public double getDiferencialEnvio() {
        return 0;
    }

    public void nuevoMontoAcumuladoEnCompras(double monto) {
    }
}

public class Compra {

    public Compra(List<Producto> productos, double diferencial)
        this.productos = productos;
        this.subtotal = productos.stream().mapToDouble(p -> p.getPrecio()).sum();
        this.costoEnvio = this.subtotal * diferencial);
    }

public class Client {
    ...
    private ClientState estado;
    ...


public Compra comprar(List<Producto> productos) {
    Compra n = new Compra(productos, this.estado.getDiferencialEnvio());
    this.compras.add(n);
    this.estado.nuevoMontoAcumuladoEnCompras(this.montoAcumuladoEnCompras());
    return n;
}
```

this.nombre;

Ejercicio 3

1) Si, ya que el Framework se ocupará de ejecutar las reglas, con un orden establecido, el desarrollador (usuario del framework) sólo podrá definir qué hacen en los métodos hooks. Las implementaciones de Rule van a ser tratada y ejecutada por el ruleEngine, tal cual lo definió el desarrollador del framework.

2) Los hooks methods son shouldProcess(); y process(); estos son los metodo que, uno como desarrollador y usuario del framework, debe redefinir.

3) El frozen spot es la parte invariable, en este caso particular:
   - El método run de Rule, que define en qué condición se corre una regla
   - La clase RuleEngine, completa, que define como se procesan las reglas.