```
public abstract class Module {
  private String tipoConeccion;
  private List<Modulo> colModSig;
  private int sig;

  public void recibirInput(List<Float> colD) {
        //colD es una colección de datos a procesar y enviar al/a los módulos conectados
        List<Float> result = this.procesarInput(colD);
        this.enviarAlSiguiente(result);
  }

  abstract public List<Float> procesarInput(List<Float> colD);

  public void enviarAlSiguiente(List<Float> colD) {
        int candidate;
        Random rand ;
        if (tipoConeccion == "simple")
              colModSig.get(1).recibirInput(colD);
        if (tipoConeccion == "AND")
              colModSig.forEach(mod -> mod.recibirInput(colD));
        if (tipoConeccion == "OR") {
              rand = new Random();
              candidate = rand.nextInt(colModSig.size()) ;
              colModSig.get(candidate).recibirInput(colD);
        }
        if (tipoConeccion == "NEXT") {
              (colModSig.get(sig).recibirInput(colD);
              if (sig < colModSig.size())
                    sig = sig +1;
              else
                    sig = 1;
        }
  }}
```

Code Smells:
Método Largo
Condicionales

Luego su compañero se ofrece a hacer el refactoring y vuelve al rato con el siguiente código que aparece en la próxima hoja, diciendo "solo falta que agregues la clase para el Next."
Elija una opción en cada inciso que sigue con respecto a la solución que escribió su compañero. Además justifique en los casos que no esté totalmente de acuerdo y corrija el código según su justificación. También agregue el código para la clase NextConnection.

d- Creo que la jerarquía que definió es correcta
Totalmente de acuerdo ◯ Parcialmente de acuerdo En desacuerdo

e- Creo que aplicó el refactoring que indicó
Totalmente de acuerdo ◯ Parcialmente de acuerdo En desacuerdo

f- Creo que las variables de instancia quedaron en el lugar correcto y no faltan ni sobran parámetros
Totalmente de acuerdo ◯ Parcialmente de acuerdo ◯ En desacuerdo

g- Creo que no quedan más refactorings por aplicar
Totalmente de acuerdo Parcialmente de acuerdo ◯ En desacuerdo

Apellido y Nombre:

Objetos 2 - Examen final

Ejercicio 1. Responda verdadero (V) o falso (F) en cada caso:

( ) Si un método tiene más de 3 o 4 parámetros es un mal olor, y hay varios refactorings que pueden aplicarse.
( ) El patrón State tiene una estructura de clases similar a la de Decorator F
( ) El patrón Adapter es uno de los patrones conocidos como"wrappers". V
( ) El refactoring "Move Method" soluciona únicamente el mal olor de código duplicado. F
( ) El refactoring "Form Template Method" permite eliminar el code smell de código duplicado. V
( ) Las clases en un framework son dependientes unas de otras; no puedo por lo general reutilizar una sola clase de un framework de manera independiente. V

Ejercicio 2. Suponga que debe refactorizar un sistema que permite aplicar distintas funciones o filtros sobre datos de entrada. El sistema está compuesto de módulos que se conectan entre sí de diferente forma para poder ir aplicando las funciones sobre los datos en una secuencia. Existe una jerarquía de distintos tipos de Módulos, cuya raíz es la clase Module (subclases SelectionModule, ArithmeticModule, SortingModule, etc.).

Suponga que tiene que realizar el refactoring con un compañero de trabajo. Ambos se encuentran con el código que está en la página siguiente y su compañero le dice:

a- "El único code smell que tiene este código es el uso de sentencias condicionales"
Totalmente de acuerdo Parcialmente de acuerdo ○En desacuerdo

b- "Este smell se resuelve aplicando el refactoring Extract Composite, que te lleva a tener el patrón Composite. Hacemos una jerarquía de conexiones, con una superclase Connection, una subclase SimpleConnection que cumpliría el rol de hoja, y una subclase CompositeConnection que a su vez tiene subclases ANDConnection, ORConnection y NEXTConnection."
○ Totalmente de acuerdo Parcialmente de acuerdo En desacuerdo

c- "Deberíamos mover el método enviarAlSiguiente() a la jerarquía de Conexion. Asi el módulo delegaría en su conexión el envío de los datos a el/los modulos siguientes"
○ Totalmente de acuerdo Parcialmente de acuerdo En desacuerdo

Si en alguna de las 3 expresiones de arriba marcó Parcialmente de acuerdo o En desacuerdo justifique.

```
public abstract class Module {
  private Connection con;
  private List<Modulo> colModSig;
  private int sig;

  public void recibirInput(List<Float> colD) {
        //colD es una colección de datos a procesar y enviar al/a los módulos conectados
        List<Float> result = this.procesarInput(colD);
        con.enviarAlSiguiente(colModSig, sig, result);
  }                                          T null

  abstract public List<Float> procesarInput(List<Float> colD);
}

public abstract class Connection {
  abstract public void enviarAlSiguiente(List<Module> colModSig, int sig, List<Float> colD);
}

public class SimpleConnection extends Connection {
  public void enviarAlSiguiente(List<Module> colModSig, int sig , List<Float> colD) {
        colModSig.get(1).recibirInput(colD);
  }
}

public abstract class CompositeConnection extends Connection {
}

public class ANDConnection extends CompositeConnection {
  public void enviarAlSiguiente(List<Module> colModSig, int sig , List<Float> colD) {
        colModSig.forEach(mod -> mod.recibirInput(colD));
  }
}

public class ORConnection extends CompositeConnection {
  public void enviarAlSiguiente(List<Module> colModSig, int sig , List<Float> colD) {
        int candidate;
        Random rand = new Random();
        candidate = rand.nextInt(colModSig.size()) ;
        colModSig.get(candidate).recibirInput(colD);
  }
}
```

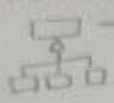
public class NEXTConnection extends CompositeConnection{

public void enviarAlSiguiente(

Apellido y Nombre:

Objetos 2 - Examen final

1) (3p)
Identifique las semejanzas y diferencias entre los patrones State y Strategy en todos los aspectos que pueda (por ej., propósito, modelo de clases y modelo de instancias, cuestiones de implementación, etc.)?

similitudes:
* cambian comportamiento dinámicamente / en tiempo de ejecución
* polimorfismo  → estados/estrategia abstracta
→ estados/estrategias concretas

diferencias:
State → el cambio es interno y automático, no puede forzar cambio de estados
Strategy → el cambio es por setters de instanciaciones que el cliente hace

* Prototipo del Strategy puede cambiar un algoritmo por uno todo de decisión específico
" "State se parece más a una máquina de estados finita

STRATEGY → hace a los distintos objetos - estrategia intercambiables

STATE → hace a los estados internos de un objeto intercambiables

en Strategy son objetos mas independientes entre sí, no tienen que conocerse el uno con el otro; mientras que el STATE, cada estado concreto puede y en algunos casos debe conocerse entre sí para poder funcionar en conjunto

2. Resuelva el siguiente ejercicio (7p)

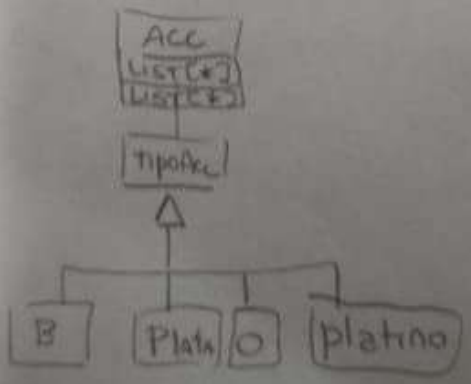
Considere el código que se presenta a continuación, correspondiente a la clase **E-Shop** que representa un sitio de compras online.

```
E-Shop>>upgradeAccount: cu
    "Recibe una instancia de Customer como parámetro. Actualiza la cuenta de cu por una mejor"

    cu account type = 'Bronce'
        ifTrue: [
                cu account: (Account new type: 'Plata').
                cu resetPoints.
                ^cu history add: (HistoryEvent new text: 'Cuenta actualizada a Plata'
                                                        on: Date today) ].
    cu account type = 'Plata'
        ifTrue: [ | s |
                s := cu account transactions inject: 0 into: [: t | t amount].
                s < 1000
                    ifTrue: [
                            cu account: (Account new type: 'Oro').
                            cu points: (cu points * 1.10).
                            ^cu history add: (HistoryEvent new text: 'Cuenta actualizada a Oro'
                                                        on: Date today) ]
                    ifFalse: [ "s > 1000"
                            cu addToShopCart: (self pdfForBook: 'Enterprise Scrum' value: 0).
                            cu account: (Account new type: 'Platino').
                            cu points: ( cu points * 1.25).
                            ^cu history add: (HistoryEvent new text: 'Cuenta actualizada a Platino'
                                                        on: Date today)]]
```

Considere que a futuro se planean tener nuevos tipos de cuenta.

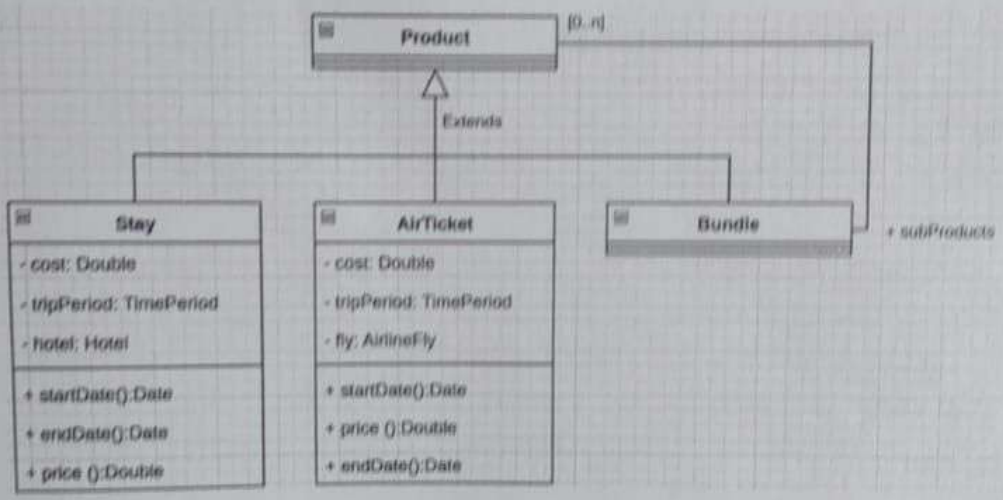
2.1) Enumere todos los malos olores que puede identificar en el código.
2.2) Por cada uno de los problemas identificados, describa cómo lo solucionaría en términos de refactoring. Si utiliza algún patrón en su solución justifique su elección.
2.3) Dibuje el diagrama de clases de su solución final y escriba el código resultante después de aplicar todos los refactorings de 2.2.



3. Evalue el siguiente caso (70pts)

Una empresa de turismo ha contratado el desarrollo de un módulo que permite crear Presupuestos (Quote) de estadia en Hoteles (Stays), Tickets Aereos (AirTickets) o Combos (Bundles) de hoteles y TicketAereos. Un Presupuesto puede tener multiples Productos. Com parte de las especificaciones se ha aclarado que todos los objetos Stay, AirTicket y Bundle deben ser polimórficos respecto a los métodos: startDate(), endDate(), price()

El equipo de desarrollo ha propuesto implementar el patrón de diseño Composite. El siguient Diagrama de Clases UML documenta el diseño propuesto. Ud. debe evaluar el diseño



Conteste según su criterio

A. (10pts)La selección del patrón es adecuada considerando el problema (**Verdadero** o **Falso**)
B. (10pts)Las variables de instancia de la clase Stay y AirTicket estan repetidas. Sería apropiado aplicar el refactoring "move up" para que esten definidas en Product? (**Verdadero** o **Falso**)
C. (50pts)La estructura del patrón (según el diagrama) es erronea e incompleta. Presente un diagrama de clases con las correcciones que considere necesarias.