ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

CCPG1009 – DISEÑO DE SOFTWARE

SEGUNDA EVALUACIÓN - I TÉRMINO 2019

Nombre: _____ ***Paralelo:*** ____

COMPROMISO DE HONOR: Al firmar este compromiso, reconozco que el presente examen está diseñado para ser resuelto de manera individual, que puedo usar un lápiz o esferográfico; que sólo puedo comunicarme con la persona responsable de la recepción del examen; y, cualquier instrumento de comunicación que hubiere traído, debo apagarlo y depositarlo en la parte anterior del aula, junto con algún otro material que se encuentre acompañándolo. Además, no debo usar calculadora alguna, consultar libros, notas, ni apuntes adicionales a los que se entreguen en esta evaluación. Los temas debo desarrollarlos de manera ordenada. Firmo el presente compromiso, como constancia de haber leído y aceptado la declaración anterior. "Como estudiante de ESPOL me comprometo a combatir la mediocridad y actuar con honestidad, por eso no copio ni dejo copiar". _____ Firma	--- 100 _____ Firma

TEMA 1 – CONCEPTOS (25 PUNTOS)

Por cada ítem, seleccione o escriba la(s) respuesta(s) correcta(s), según considere conveniente. En caso de ser falso, debe justificar su respuesta.

1) Seleccione uno de los siguientes malos olores de programación y explique en qué consiste: (3 pt)
   a. Grupos de datos. “Data clumps”
   b. Cambio divergente. “Divergent change”
   c. Legado forzado. “Refused bequest”

2) Indique como se puede evitar el mal olor de programación “Primitive Obsession”. (2 pt)

3) ¿Qué patrón de diseño aplicaría para utilizar una clase incluida en un archivo JAR pero que no funciona exactamente como lo hacen otras clases del sistema que está implementando? Explique las ventajas de utilizar el patrón elegido (3 pt)

4) Decorator, es un patrón de diseño que permite agregar _____ a un objeto de forma _____. (4 pt)

5) Si las pruebas unitarias incrementan el tiempo de desarrollo de un sistema. ¿Por qué se aconseja realizarlas? (3 pt)

6) Las pruebas unitarias pueden evitar que el sistema desarrollado esté libre de fallos (bugs). Explique. (3 pt)
   a. Verdadero.
   b. Falso. Porque:

~~7) ¿Qué mal olor se poder corregir al aplicar la técnica de refactorización “Refused bequest”? (2 pts)~~

8) Explique al menos 2 de las ventajas de utilizar el patrón de diseño **Cadena de responsabilidad “Chain of Responsibility”**. (5 pts)

TEMA 2 – REFACTORING (35 PUNTOS)

El siguiente código fuente corresponde a un sistema de una máquina expendedora llamada BarExpress, en donde se elije un producto y se ingresa el valor en centavos correspondiente al costo del mismo, luego el usuario obtiene el producto y el cambio en caso de que se pueda realizar la operación.
Nota: entre las líneas 76 y 96 se ha ocultado el código fuente, pero hace lo que indica el comentario.

```
1   import java.util.ArrayList;
2   import java.util.Collections;
3   import java.util.List;
4   import java.util.Objects;
5
6   public class BarExpress {
7       private Map<Coin, Integer> monedas = new Map<Coin, Integer>();
8       private Map<Item, Integer> productos = new Map<Item, Integer>();
9       private long totalVendido;
10      private Item prodEscogido;
11      private long dineroIngresado;
12
13      public BarExpress(){
14          for(Coin c : Coin.values()){ monedas.put(c, 5);  }
15          for(Item i : Item.values()){ productos.put(i, 5);  }
16      }
17      public long setItemGetPrice(Item item) throws Exception {
18          Integer value = productos.get(item);
19          value = value == null? 0 : value ;
20          if(value > 0){ // Hay almenos un item
21              prodEscogido = item;
22              return prodEscogido.getPrice();
23          }
24          throw new Exception("Agotado, Seleccione otro item");
25      }
26      public void insertCoin(Coin coin) {
27          dineroIngresado = dineroIngresado + coin.getDenomination();
28          int count = monedas.get(item);
29          monedas.put(item, count+1);
30      }
31      public List<Objects> collectItemAndChange() {
32          Item item = collectItem();
33          totalVendido += prodEscogido.getPrice();
34          List<Coin> change = collectChange();
35          return List.of(item, change);
36      }
37      public List<Coin> refund(){
38          List<Coin> refund = getChange(dineroIngresado);
39          updatemonedas(refund);
40          dineroIngresado = 0;
41          prodEscogido = null;
42          return refund;
43      }
44      private Item collectItem() throws Exception{
45          long amount = dineroIngresado - prodEscogido.getPrice();
46          if(dineroIngresado >= prodEscogido.getPrice() &&
47          hasSufficientChangeForAmount(amount)){
48              Integer value = productos.get(prodEscogido);
49              value = value == null? 0 : value ;
50              if(value > 0){ // Hay almenos un item
51                  int count = productos.get(prodEscogido);
52                  productos.put(prodEscogido, count - 1);
53              }
54              return prodEscogido;
55          }
56          else{
57              throw new Exception("Monedas insuficientes para cambio");
58          }
59          long remain = prodEscogido.getPrice() - dineroIngresado;
60          throw new Exception("Falta dinero : ", remain);
61      }
62      private List<Coin> collectChange() {
63          long changeAm = dineroIngresado - prodEscogido.getPrice();
64          List<Coin> change = getChange(changeAm);
65          updatemonedas(change);
66          dineroIngresado = 0;
67          prodEscogido = null;
68          return change;
69      }
70      private List<Coin> getChange(long amount) throws Exception{
71          List<Coin> changes = Collections.EMPTY_LIST;
72          if(amount > 0){
73              changes = new ArrayList<Coin>();
74              long balance = amount;
75              // Cuantas monedas de cada denominacion
76  >           while(balance > 0){ ···
96              }
97              return changes;
98      }
99      private boolean hasSufficientChangeForAmount(long amount){
100         boolean hasChange = true;
101         try{
102             getChange(amount);
103         }catch(Exception nsce){
104             return hasChange = false;
105         }
106         return hasChange;
107     }
108     private void updatemonedas(List change) {
109         for(Coin c : change){
110             Integer value = monedas.get(c);
111             value = value == null? 0 : value ;
112             if(value > 0){ // Hay almenos un item
113                 int count = monedas.get(c);
114                 monedas.put(c, count - 1);
115             }
116         }
117     }
118     // Getters y Setters
119 }
120 public enum Item{ WATER("Water", 30), COKE("Coke", 25),
121     PEPSI("Pepsi", 35), SODA("Soda", 45);
122     private String name;
123     private int price;
124     private Item(String name, int price){
125         this.name = name; this.price = price; }
126     // Getters y Setters
127 }
128 public enum Coin { NICKLE(5), DIME(10), QUARTER(25);
129     private int denomination;
130     private Coin(int denomination){
131         this.denomination = denomination;
132     }
133     // Getters y Setters
134 }
```

9) Dado el código anterior, señale los malos olores de programación e indique su nombre (5 pt), explique por qué sería un problema (10 pt) e indique las técnicas de refactorización que utilizaría para mejorar el código, justifique. (10 pt). No es necesario escribir código.

10) Realizar 2 pruebas unitarias para el método **setItemGetPrice**() y 1 prueba para los métodos **insertCoin**() y **refund**(). (10 pt)

TEMA 3 – APLICACIÓN DE PATRONES (40 PUNTOS)

11) Se solicita lo siguiente:

- Identificar los patrones de diseño que puede aplicar para implementar lo solicitado. Justifique. (10 pt)
- Diagrama de clases (solo los modelos del MVC) aplicando los patrones de diseño que considere apropiados. No olvide indicar herencias, multiplicidades, visibilidad de atributos y métodos y de indicar claramente si las entidades corresponden a clases concretas, abstractas o interfaces. (25 pt)
- Separe en paquetes según los patrones usados. (5 pt)

Requerimiento:

En un sistema de ventas en locales y envíos a domicilio, los **gerentes** pueden consultar todo lo que maneje el sistema (usuarios, envíos, artículos, ventas). También pueden asignar las funciones de administración a un usuario del sistema de forma dinámica, para que siempre haya alguien en el local con estas funciones. Dichas funciones de administración conllevan todos los permisos CRUD, excepto eliminar (manejar eliminado lógico). En el caso de las bodegas, estas funciones de administración deben ser asignadas al **jefe de bodega** siempre y cuando no exista otro usuario con dichas funciones en esa localidad. Por otro lado, los **vendedores** son los que realizan las ventas en los locales y les dan la opción a los clientes de poder realizar un envío a domicilio o retirarlos en dicho local.