Trabajo Práctico 5 – Adapter y Decorator

1. Un sistema trabaja con diferentes tipos de motores (Común, Económico) que comparten características comunes así como su funcionamiento, que está dado por operaciones como **arrancar**, **acelerar** y **apagar**.

Se desea incorporar al sistema una clase ya existente de tipo motor eléctrico con un funcionamiento diferente al de los demás. Estos motores pueden realizar operaciones como **conectar** y **activar**, **moverMasRapido**, **detener** y **desconectar**.

Se debe adaptar la nueva clase de forma que no se vea afectada la lógica de la aplicación cliente que interactúa con las clases que representan los motores.

a) Aplique el patrón **Adapter,** utilizando un adaptador de objetos, para diseñar el modelo de clases que de solución al problema planteado.

b) Implemente la solución en Java, utilizando mensajes informativos por pantalla de la operación que se está realizando.

2. Supongamos la siguiente clase Reporte:

```
class Report {

 private String reporte;

 public Report(String reporte) {
          this.reporte = reporte;
 }
 void export(File file) {
   if (file == null) {
     throw new IllegalArgumentException(
       "File es NULL; no puedo exportar..."
     );
   }
   if (file.exists()) {
     throw new IllegalArgumentException(
       "El archivo ya existe..."
     );
   }
   // Exportar el reporte a un archivo.
  }
}
```

a. Implemente la exportación.

b. Escriba los casos de test para testear la clase completa.

b. Utilice el pattern Decorador para reescribir la funcionalidad de Reporte, de modo tal que le permita escribir Reportes que exporten sin verificar si el archivo existe (o sea, lo sobreescriba) y Reportes que no permitan sobrescribir el archivo. Luego, modifique los test: Las líneas de asserts NO deberian cambiar.

3. Un restaurante de comidas rápidas ofrece 3 tipos de combos (Combo Básico, Combo Familiar, Combo Especial). De cada combo podemos conocer su descripción que nos detalla el contenido del combo, y por otro lado podemos conocer su precio.

El restaurante también ofrece la posibilidad de aumentar el pedido mediante diferentes porciones adicionales (Tomate, Papas, Carne, Queso). Cada porción que se agrega al combo tiene un costo adicional.

Se desea crear un sistema de pedidos que permita al usuario seleccionar el combo deseado, así como armar su propio pedido con las porciones adicionales que desee. El sistema deberá informar sobre el pedido del usuario detallando su descripción y el valor total del mismo.

a) Aplique el patrón **Decorator** para diseñar el modelo de clases que de solución al problema planteado.

b) Implemente la solución en Java, especificando en el programa principal el armado de 2 combos distintos con al menos dos adicionales cada uno.

4. La siguiente clase, es un ejemplo de como consumir un servicio Web:

```
import java.io.IOException;

import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;

public class RestCall {

 private String url;

 public RestCall(String url) {
  this.url = url;
 }

 public String run() {
  OkHttpClient client = new OkHttpClient();
  Request request = new Request.Builder().url(this.url).build();

  try (Response response = client.newCall(request).execute()) {
   return response.body().string();
  } catch (IOException e) {
   throw new RuntimeException(e);
  }
 }
}
```

Y asi lo invocamos:

```
public class Main {

 public static void main(String[] args) {
  RestCall rest = new RestCall(
    "https://jsonplaceholder.typicode.com/posts");
  System.out.println(rest.run());
 }
}
```

Este Main realiza la llamada al servicio web e imprime la salida por consola. El servicio web: https://jsonplaceholder.typicode.com/posts responde un JSON con una lista de posts que diferentes usuarios hicieron en un Blog (son datos falsos).

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
     "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
  },
 ...
]
```

El ejemplo utiliza la librería llamada OkHttp, que la pueden incorporar utilizando Maven, o sino descargando y agregando al build path los siguientes jars:

https://mvnrepository.com/artifact/com.squareup.okhttp3/okhttp/4.9.1

https://mvnrepository.com/artifact/com.squareup.okio/okio/2.8.0

https://mvnrepository.com/artifact/org.jetbrains.kotlin/kotlin-stdlib/1.4.10

https://mvnrepository.com/artifact/org.jetbrains.kotlin/kotlin-stdlib-common/1.4.0

https://mvnrepository.com/artifact/org.jetbrains/annotations/13.0

Se pide:

Utilice el patron decorador para crear un decorador que guarda la lista de items del servicio web en un archivo de texto plano y otro decorador que guarde la lista en una tabla de una base de datos relacional. Luego, desde un Main muestre las diferentes combinaciones que puede realizar.