Orientación a Objetos 2 – Curso 2022

Práctica 2

Fecha de última edición: 7 marzo 2022

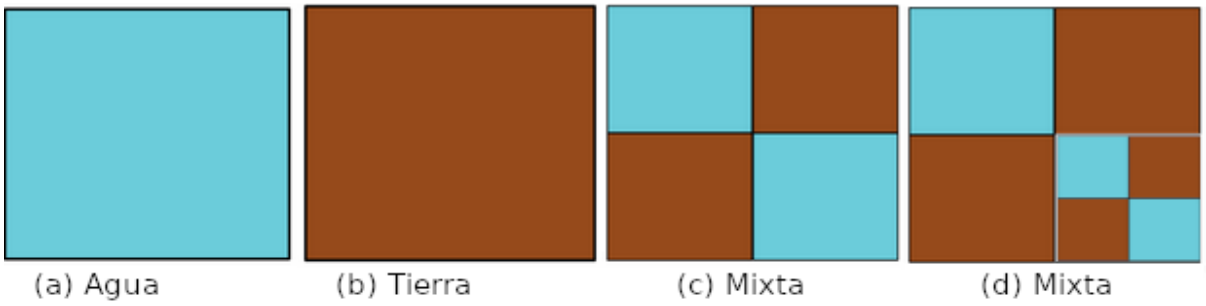
Ejercicio 1: Topografías

Un objeto Topografía representa la distribución de agua y tierra de una región cuadrada del planeta, la cual está formada por porciones de “agua” y de “tierra”. La siguiente figura muestra:

(a) el aspecto de una topografía formada únicamente por agua.

(b) otra formada sólamente por tierra.

(c) y (d) topografías mixtas.



Una topografía mixta está formada por partes de agua y partes de tierra (4 partes en total). Éstas a su vez podrían descomponerse en 4 más y así siguiendo.

La proporción de agua de una topografía sólo agua es 1. La proporción de agua de una topografía sólo tierra es 0. La proporción de agua de una topografía compuesta está dada por la suma de la proporción de agua de sus componentes dividida por 4. En el ejemplo, la proporción de agua es: (1 + 0 + 0 + 1) / 4 = 1/2. La proporción siempre es un valor entre 0 y 1.

1. Diseñe e implemente las clases necesarias para que sea posible:
    1. crear Topografías,
    2. calcular su proporción de agua y tierra,
    3. comparar igualdad entre topografías. Dos topografías son iguales si tienen exactamente la misma composición. Es decir, son iguales las proporciones de agua y tierra, y además, para aquellas que son mixtas, la disposición de sus partes es igual.

       Pista: notar que la definición de igualdad para topografías mixtas corresponde exactamente a la misma que implementan las listas en Java.

https://docs.oracle.com/javase/8/docs/api/java/util/AbstractList.html#equals-java.lang.Object-

2. Diseñe e implemente test cases para probar la funcionalidad implementada. Incluya en el set up de los tests, la topografía compuesta del ejemplo.

Ejercicio 2: Más Topografías

Extienda el ejercicio anterior para soportar (además de Agua y Tierra) el terreno Pantano. Un pantano tiene una proporción de agua de 0.7 y una proporción de tierra de 0.3. No olvide hacer las modificaciones necesarias para responder adecuadamente la comparación por igualdad.

Ejercicio 3: FileSystem

Un file system contiene un conjunto de directorios y archivos organizados jerárquicamente mediante una relación de inclusión. De cada archivo se conoce el nombre, fecha de creación y tamaño en bytes. De un directorio se conoce el nombre, fecha de creación y contenido (el tamaño es siempre 32kb). Modele el file system y provea la siguiente funcionalidad:

```
public class Archivo {
  /**
      * Crea un nuevo archivo con nombre <nombre>, de <tamano> tamano
      * y en la fecha <fecha>.
  */
  public Archivo (String nombre, LocalDate fecha, int tamano)

}

public class Directorio {
      /**
      * Crea un nuevo Directorio con nombre <nombre> y en la fecha <fecha>.
      */
    public Directorio(String nombre, LocalDate fecha)

   /**
      * Retorna el espacio total ocupado, incluyendo su contenido.
      */
    public int tamanoTotalOcupado()
```

```
    /**
      * Retorna el archivo con mayor cantidad de bytes en cualquier nivel del
      * filesystem contenido por directorio receptor
      */
    public Archivo archivoMasGrande()



  /**
      * Retorna el archivo con fecha de creación más reciente en cualquier
nivel
      * del filesystem contenido por directorio receptor.
      */
    public Archivo archivoMasNuevo()

}
```

Tareas:

1. Diseñe y represente un modelo UML de clases de su aplicación, identifique el patrón de diseño empleado (utilice estereotipos UML para indicar los roles de cada una de las clases en ese patrón).
2. Diseñe, implemente y ejecute test cases para verificar el funcionamiento de su aplicación. En el archivo DirectorioTest.java del material adicional se provee la clase DirectorioTest que contiene tests para los métodos arriba descritos y la definición del método setUp. Utilice el código provisto como guía de su solución y extienda lo que sea necesario.
3. Implemente completamente en Java.

Ejercicio 4: Cálculo de sueldos

Sea una empresa que paga sueldos a sus empleados, los cuales están organizados en tres tipos: Temporarios, Pasantes y Planta. El sueldo se compone de 3 elementos: sueldo básico, adicionales y descuentos.

	Temporario	Pasante	Planta
básico	$ 20.000 + cantidad de horas que trabajo * $ 300.	$20.000	$ 50.000
adicional	$5.000 si está casado	$2.000 por examen que rindió	$5.000 si está casado $2.000 por cada hijo

	$2.000 por cada hijo		$2.000 por cada año de antigüedad
descuento	13% del sueldo básico 5% del sueldo adicional	13% del sueldo básico 5% del sueldo adicional	13% del sueldo básico 5% del sueldo adicional

Tareas:

1. Diseñe la jerarquía de Empleados de forma tal que cualquier empleado puede responder al mensaje #sueldo.
2. Desarrolle los test cases necesarios para probar todos los casos posibles.
3. Implemente en Java.

Orientación a Objetos 2 – Curso 2022

Práctica 3

Fecha de última edición: 4 abril 2022

Ejercicio 1: ToDoItem

Se desea definir un sistema de seguimiento de tareas similar a Jira.
En este sistema hay tareas en las cuales se puede definir el nombre y una serie de comentarios. Las tareas atraviesan diferentes etapas a lo largo de su ciclo de vida y ellas son: *pending*, *in-progress*, *paused* y *finished*. Cada tarea debe estar modelada mediante la clase ToDoItem con el siguiente protocolo:

```
public class ToDoItem {
	/**
	* Instancia un ToDoItem nuevo en estado pending con <name> como nombre.
	*/
    public ToDoItem(String name)


   /**
	* Pasa el ToDoItem a in-progress (siempre y cuando su estado actual sea
	* pending, si se encuentra en otro estado, no hace nada)
```

```
      */
    public void start()


  /**
      * Pasa la tarea a paused si su estado es in-progress, o a in-progress si
su
      * estado es paused. Caso contrario (pending o finished) genera un error
      * informando la causa específica del mismo
      */
    public void togglePause()
  /**
      * Pasa el ToDoItem a finished (siempre y cuando su estado actual sea
      * in-progress o pausada, si se encuentra en otro estado, no hace nada)
      */
    public void finish()
  /**
      * Retorna el tiempo que transcurrió desde que se inició la tarea (start)
      * hasta que se finalizó. En caso de que no esté finalizada, el tiempo que
      * haya transcurrido hasta el momento actual. Si la tarea no se inició,
      * genera un error informando la causa específica del mismo.
      */
    public Duration workedTime()


      /**
      * Agrega un comentario a la tarea siempre y cuando no haya finalizado.
Caso
      * contrario no hace nada."
      */
    public void addComment(String comment)
}
```

Nota: para generar o levantar un error debe utilizar la expresión

```
      throw new RuntimeException("Este es mi mensaje de error");
```

El mensaje de error específico que se espera en este ejercicio debe ser descriptivo del caso. Por ejemplo, para el método togglePause() , el mensaje de error debe indicar que el ToDoItem no se encuentra en in-progress o paused:

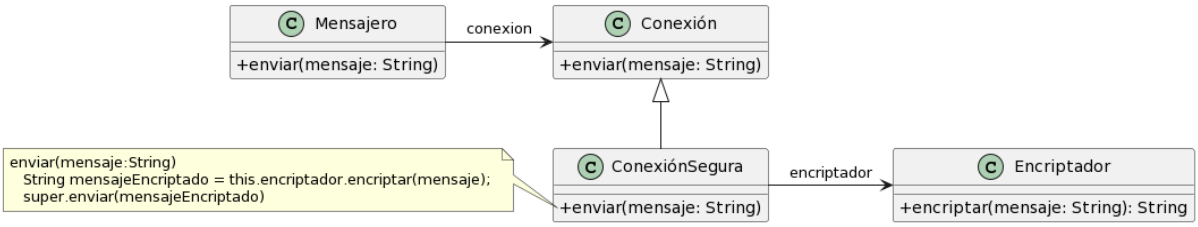
```
  throw new RuntimeException("El objeto ToDoItem no se encuentra en pause o in-
progress");
```

Tareas:

1. Modele una solución orientada a objetos para el problema planteado utilizando un diagrama de clases UML. Si utilizó algún patrón de diseño indique cuáles son los participantes en su modelo de acuerdo a Gamma et al.
2. Implemente su solución en Java. Para comprobar cómo funciona recomendamos usar test cases.

Ejercicio 2: Encriptador

En un sistema de mensajes instantáneos (como Hangouts) se envían mensajes de una máquina a otra a través de una red. Para asegurar que la información que pasa por la red no es espiada, el sistema utiliza una conexión segura. Este tipo de conexión encripta la información antes de enviarla y la desencripta al recibirla. La siguiente figura ilustra un posible diseño para este enunciado.



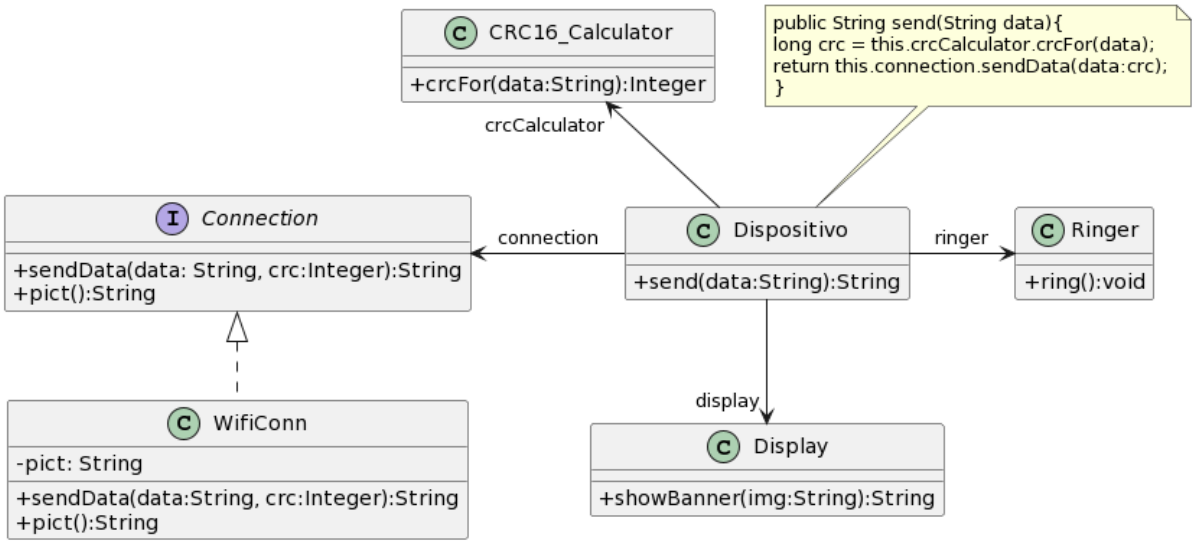
El encriptador utiliza el algoritmo RSA. Sin embargo, se desea agregar otros algoritmos (diferentes algoritmos ofrecen distintos niveles de seguridad, overhead en la transmisión, etc.).

Tareas:

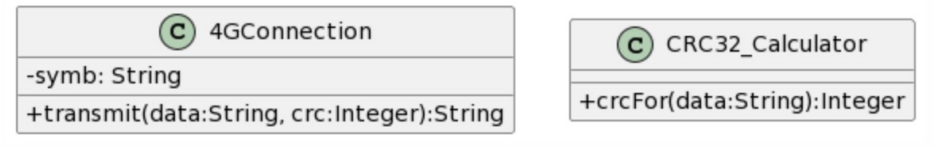
1. Modifique el diseño para que el objeto Encriptador pueda encriptar mensajes usando los algoritmos Blowfish y RC4, además del ya soportado RSA.
2. Documente mediante un diagrama de clases UML indicando los roles de cada clase.

Ejercicio 3 - Dispositivo móvil y conexiones

Sea el software de un dispositivo móvil que utiliza una conexión WiFi para transmitir datos. La figura muestra parte de su diseño:



Nuevas clases a utilizar:



El dispositivo utiliza, para asegurar la integridad de los datos emitidos, el mecanismo de cálculo de redundancia cíclica que le provee la clase CRC16_Calculator que recibe el mensaje `crcFor(data: String)` con los datos a enviar y devuelve un valor numérico. Luego el dispositivo envía a la conexión el mensaje `sendData` con ambos parámetros (los datos y el valor numérico calculado).

Se desea hacer dos cambios en el software. En primer lugar, se quiere que el dispositivo tenga capacidad de ser configurado para utilizar conexiones 4G. Para este cambio se debe utilizar la clase 4GConnection.
Además se desea poder configurar el dispositivo para que utilice en distintos momentos un cálculo de CRC de 16 o de 32 bits. Es decir que en algún momento el dispositivo seguirá utilizando CRC16_Calculator y en otros podrá ser configurado para utilizar la clase CRC32_Calculator. Se desea permitir que en el futuro se puedan utilizar otros algoritmos de CRC.

Cuando se cambia de conexión, el dispositivo muestra en pantalla el símbolo correspondiente (que se obtiene con el getter `pict()` para el caso de WiFiConn y `symb()` de 4GConnection) y se utiliza el objeto Ringer para emitir un `ring()`.

Tanto las clases existentes como las nuevas a utilizar pueden ser ubicadas en las jerarquías que corresponda (modificar la clase de la que extienden o la interfaz que implementan) y se les pueden agregar mensajes, pero no se pueden modificar los mensajes que ya existen porque otras partes del sistema podrían dejar de funcionar.

Dado que esto es una simulación, y no dispone de hardware ni emulador para esto, la signatura de los mensajes se ha simplificado para que se retorne un String descriptivo de los eventos que suceden en el dispositivo y permitir de esta forma simplificar la escritura de los tests.

Modele los cambios necesarios para poder agregar al protocolo de la clase Dispositivo los mensajes para

- cambiar la conexión, ya sea la 4GConnection o la WifiConn. En este método se espera que se pase a utilizar la conexión recibida, muestre en el display su símbolo y genere el sonido.
- poder configurar el calculador de CRC, que puede ser el CRC16_Calculator, el CRC32_Calculator, o pueden ser nuevos a futuro.

Tareas:

1. Realice un diagrama UML de clases para su solución al problema planteado. Indique claramente el o los patrones de diseño que utiliza en el modelo y el rol que cada clase cumple en cada uno.
2. Implemente en Java todo lo necesario para asegurar el envío de datos por cualquiera de las conexiones y el cálculo adecuado del índice de redundancia cíclica.
3. Implemente test cases para los siguientes métodos de la clase Dispositivo:
    1. `send`
    2. `conectarCon`
    3. `configurarCRC`

En cuanto a CRC16_Calculator, puede utilizar la siguiente implementación:

```
public long crcFor(String datos) {
	int crc = 0xFFFF;
	for (int j = 0; j < datos.getBytes().length; j++) {
		crc = ((crc >>> 8) | (crc << 8)) & 0xffff;
		crc ^= (datos.getBytes()[j] & 0xff);
		crc ^= ((crc & 0xff) >> 4);
		crc ^= (crc << 12) & 0xffff;
```

```
            crc ^= ((crc & 0xFF) << 5) & 0xffff;
        }
        crc &= 0xffff;
        return crc;
}
```

Nota: para implementar CRC32_Calculator utilice la clase java.util.zip.CRC32 de la siguiente manera:

```
CRC32 crc = new CRC32();
String datos = "un mensaje";
crc.update(datos.getBytes());
long result = crc.getValue();
```

Ejercicio 4 - Decodificador de películas

Sea una empresa de cable *on demand* que entrega decodificadores a sus clientes para que miren las películas que ofrece. El decodificador muestra la grilla de películas y también sugiere películas.

Usted debe implementar la aplicación para que el decodificador sugiera películas. El decodificador conoce la grilla de películas (lista completa que ofrece la empresa), como así también las películas que reproduce. De cada película se conoce título, año de estreno, películas similares y puntaje. La similaridad establece una relación recíproca entre dos películas, por lo que si A es similar a B entonces también B es similar a A.

Cada decodificador puede ser configurado para que sugiera 3 películas (que no haya reproducido) por alguno de los siguientes criterios:

(i) novedad: las películas más recientes.

(ii) similaridad: las películas más nuevas son similares a alguna película que reprodujo.

(iii) puntaje: las películas de mayor puntaje, para igual puntaje considera las más recientes.

Tenga en cuenta que la configuración del criterio de sugerencia del decodificador no es fija, sino que el usuario la debe poder cambiar en cualquier momento. El sistema debe soportar agregar nuevos tipos de sugerencias aparte de las tres mencionadas.

Sea un decodificador que reprodujo Thor y Rocky, y posee la siguiente lista de películas:

Thor, 7.9, 2007 (Similar a Capitan America, Iron Man)

Capitan America, 7.8, 2016 (Similar a Thor, Iron Man)

Iron man, 7.9, 2010 (Similar a Thor, Capitan America)

Dunkirk, 7.9, 2017

Rocky, 8.1, 1976 (Similar a Rambo)

Rambo, 7.8, 1979 (Similar a Rocky)

Las películas que debería sugerir son:

(i) Dunkirk, Capitan America, Iron man

(ii) Capitán América, Iron man, Rambo

(iii) Dunkirk, Iron man, Capitan America

Nota: si existen más de 3 películas con el mismo criterio, retorna 3 de ellas sin importar cuales. Por ejemplo, si las 6 películas son del 2018, el criterio (i) retorna 3 cualquiera.

Tareas:

1. Realice el diseño de una correcta solución orientada a objetos con un diagrama UML de clases.
2. Si utiliza patrones de diseño indique cuáles y también indique los participantes de esos patrones en su solución según el libro de Gamma et al.
3. Escriba un test case que incluya estos pasos, con los ejemplos mencionados anteriormente:
    - configure al decodificador para que sugiera por similaridad (ii)
    - solicite al mismo decodificador las sugerencias
    - configure al mismo decodificador para que sugiera por puntaje (iii)
    - solicite al mismo decodificador las sugerencias
4. Programe su solución en Java. Debe implementarse respetando todas las buenas prácticas de diseño y programación de POO.

Orientación a Objetos 2 – Práctica 4

Ejercicio 1: Acceso a la base de datos

Queremos acceder a una base de datos que contiene información sobre cómics. Este acceso está dado por el comportamiento de la clase DatabaseRealAccess con el siguiente protocolo y modelado como muestra la Figura 1.

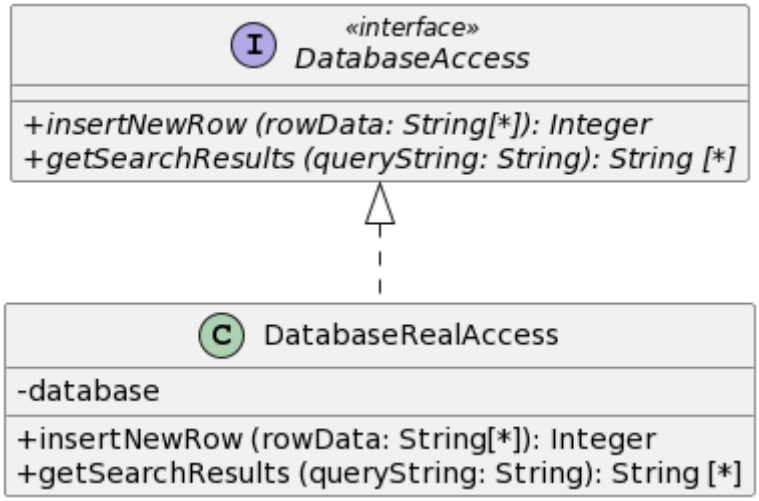


Figura 1

```
public interface DatabaseAccess {
  /**
   * Retorna una colección de acuerdo al texto que posee "queryString"
   *
   * @param queryString
   * @return
   */
  public Collection<String> getSearchResults(String queryString);

  /**
   * Realiza la inserción de nueva información en la base de datos y
   * retorna el id que recibe la nueva inserción
   *
   * @param rowData
```

```
   * @return
   */
  public int insertNewRow(List<String> rowData);
}
```

En este caso, ustedes recibirán una implementación prototípica de la clase **DatabaseRealAccess** (ver material extra) que simula el uso de una base datos de la siguiente forma (mire el código y los tests para entender cómo está implementada).

```
// Instancia una base de datos que posee dos filas
  database = new DatabaseRealAccess();

  // Retorna el siguiente arreglo: ['Spiderman' 'Marvel'].
  database.getSearchResults("select * from comics where id=1");

  // Retorna 3, que es el id que se le asigna
  database.insertNewRow(Arrays.asList("Patoruzú", "La flor"));

  // Retorna el siguiente arreglo: ['Patoruzú', 'La flor'], ya que lo
insertó antes
  database.getSearchResults("select * from comics where id=3");
```

Tareas

En esta oportunidad, usted debe proveer un *protection proxy* para que el acceso a la base de datos lo puedan realizar solamente usuarios que se hayan autenticado previamente. Su tarea es diseñar y programar en Java lo que sea necesario para ofrecer la funcionalidad antes descrita. Se espera que entregue los siguientes productos.

1. Diagrama de clases UML.
2. Implementación en Java de la funcionalidad requerida.
3. Implementación de los tests (JUnit) que considere necesarios.

Ejercicio 2: File Manager

En un **File Manager** se muestran los archivos. De los archivos se conoce:

- Nombre
- Extensión
- Tamaño
- Fecha de creación
- Fecha de modificación
- Permisos

Implemente la clase **FileOO2**, con las correspondientes variables de instancia y *accessors*.

En el File Manager el usuario debe poder elegir cómo se muestra un archivo (instancia de la clase FileOO2), es decir, cuáles de los aspectos mencionados anteriormente se muestran, y en qué orden. Esto quiere decir que un usuario podría querer ver los archivos de muchas maneras. Algunas de ellas son:

- `nombre - extensión`
- `nombre - extensión - fecha de creación`
- `permisos - nombre - extensión - tamaño`

Para esto, el objeto o los objetos que representen a los archivos en el FileManager debe(n) entender el mensaje `prettyPrint().`

Es decir, un objeto cliente (digamos el FileManager) le enviará al objeto que Ud. determine, el mensaje `prettyPrint().` **De acuerdo a cómo el usuario lo haya configurado se deberá retornar un String con los aspectos seleccionados por el usuario en el orden especificado por éste**. Considere que un mismo archivo podría verse de formas diferentes desde distintos puntos del sistema, y que el usuario podría cambiar la configuración del sistema (qué y en qué orden quiere ver) en runtime.

Tareas

1) Discuta los requerimientos y diseñe una solución. Si aplica un patrón de diseño, indique cuál es y justifique su aplicabilidad.
2) Implemente en Java.
3) Instancie un objeto para cada uno de los ejemplos citados anteriormente y verifique escribiendo tests de unidad.

Orientación a Objetos 2 – Curso 2022

Práctica 7

Fecha de última edición: 31 mayo 2022

Ejercicio 1

Sea un lavarropas semiautomático que posee tres programas de lavado: lavado diario, ropa delicada y ropa muy sucia. Cada programa consiste de lo siguiente:

- Lavado diario: llena de agua hasta el 50%, agrega jabón en polvo, 20 minutos de lavado, cambia el agua, agrega enjuague, 7 minutos de enjuague, 3 minutos de centrifugado.
- Ropa delicada: llena de agua hasta el 100%, agrega jabón en polvo, 40 minutos de lavado, cambia el agua, agrega enjuague, 15 minutos de enjuague, no hay centrifugado
- Ropa muy sucia: llena de agua hasta el 100%, agrega jabón en polvo, 100 minutos de lavado, cambia el agua, agrega jabón en polvo, 30 minutos de lavado, cambia el agua, agrega enjuague, 15 minutos de enjuague, 10 minutos de centrifugado.

Cada programa al terminar su lavado, retorna la duración total del mismo en minutos.

Considere que el lavarropas posee los siguientes mensajes, los cuales puede utilizar sin implementar:

```
public class Lavarropas {

    public void agregarAgua(int porcentaje) {...}
    public void agregarJabonEnPolvo() {...}
    public void lavar(int minutos) {...}
    public void vaciarAgua() {...}
    public void agregarEnjuague() {...}
    public void enjuaga(int minutos) {...}
    public void centrifugar(int minutos) {...}
```

Usted debe implementar los mensajes para poder iniciar el lavado utilizando uno de los programas mencionados.

Tareas:

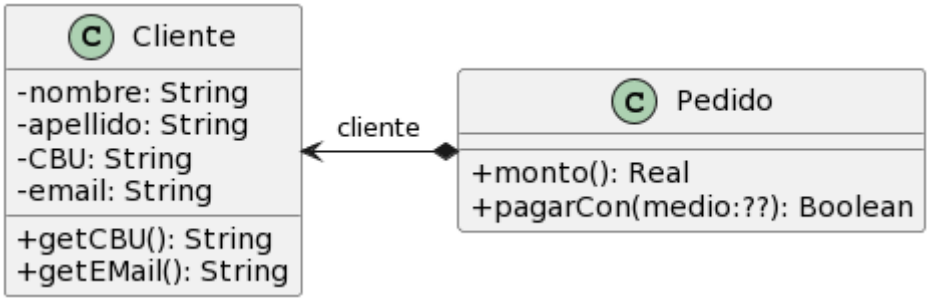
1. Modele una solución para el problema planteado. Si utiliza algún patrón, indique cuál
2. Implemente en Java
3. Implemente un test para poder realizar el lavado con el programa para lavado diario.

Ejercicio 2

En una cafetería se desea implementar el pago de los pedidos procesados mediante diferentes medios de pago (también conocidos como *gateways*). Por el momento, se requiere la posibilidad

de realizar el pago por medio de débito automático y DineroMail. Sin embargo, no se descarta la posibilidad de utilizar otros medios en el futuro.

En la siguiente figura se muestra el modelo del sistema que incluye la clase Cliente, con variables de instancia para el nombre, apellido, mail y CBU (Clave Bancaria Uniforme) y sus correspondientes *accessors*. Y, por otro lado, la clase Pedido, que conoce al cliente que lo solicitó mediante la variable de instancia cliente, un mensaje `monto()` que calcula el costo del pedido y un mensaje `pagarCon(medio: ??)`. Este último retorna true o false dependiendo de si pudo o no realizar el pago.

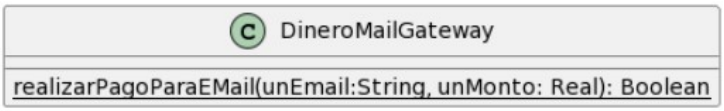


Cada uno de los medios de pago dispone de una API con su correspondiente procedimiento de uso:

- Débito Automático: el pago requiere una autorización previa donde se valida si el cliente dispone de fondos. Para realizarlo Ud. dispone de una clase llamada **DebitoGateway** con dos mensajes de clase:
    - autorizarMontoConCBU(unMonto: Real, cbu: String)
    - pagarMontoConCBU(unMonto: Real, cbu: String)

  El primero permite autorizar el monto de pago para un CBU retornando una colección de Strings que representan los errores de autorización. El segundo método, permite realizar el pago siempre y cuando no se hayan obtenido errores en la autorización anterior.

- DineroMail: el pago se realiza en un solo paso. Para realizarlo Ud. dispone de una clase llamada **DineroMailGateway** con un mensaje de clase:
    - realizarPagoParaEmail (unEmail: String, unMonto: Real) que recibe como parámetro el e-mail del cliente y el monto de pago y retorna un boolean indicando si la operación se pudo realizar correctamente.



Tenga en cuenta que las API de los medios de pago no pueden ser modificadas ya que son utilizadas por varios sistemas.

Tareas:

1. Diseñe una solución que permita soportar los diferentes medios de pago. Defina el comportamiento del mensaje que permite realizar el pago de un pedido. Si utiliza algún patrón de diseño, indique cuál mediante estereotipos. Realice el diagrama de clases UML correspondiente.
2. Implemente en Java su diseño.
3. Realice un test para mostrar cómo realizar el pago de un pedido por débito automático de un cliente que no dispone de fondos suficientes.

Ejercicio 3

Sea un sitio web que tiene por objetivo conseguir financiación para proyectos a través de una mecánica muy sencilla. Las personas que tienen proyectos (emprendedores) publicitan los mismos y las personas que desean invertir indican en qué proyecto desean hacerlo y con qué dinero. Cuando un proyecto logra el dinero necesario se registra como conformado. Por su parte, el emprendedor puede cancelar su proyecto sólo si no está conformado. Cuando se cancela el proyecto no se puede aportar más dinero. En cambio, cuando se conforma el proyecto, es posible seguir aportando.
El sitio debe proveer la siguiente funcionalidad:

- Crear un proyecto con un título, un emprendedor responsable y un monto a alcanzar.
- Invertir un cierto monto en un proyecto dado.
- Consultar el monto total recibido en inversiones para un proyecto.
- Cancelar proyecto.

Tareas:

1. Diseñe una solución. Si utiliza algún patrón de diseño, indique cuál mediante estereotipos. Realice el diagrama de clases UML correspondiente.
2. Implemente en Java su diseño.
3. Realice un test para aportar dinero al proyecto “Vacaciones Kathmandu 2023” que necesita 5000 pesos y se aportan 500 pesos.