FACULTAD DE INFORMATICA | UNIVERSIDAD NACIONAL DE LA PLATA

Orientación a Objetos 2
Cuadernillo Semestral de Actividades
- Patrones de diseño -

Actualizado: 14 de abril de 2025

El presente cuadernillo **estará en elaboración** durante el semestre y tendrá un compilado con todos los ejercicios que se usarán durante la asignatura. Se irán agregando ejercicios al final del cuadernillo para poder poner en práctica los contenidos que se van viendo en la materia.

Cada semana les indicaremos cuáles son los ejercicios en los que deberían enfocarse para estar al día y algunos de ellos serán discutidos en la explicación de práctica.

Recomendación importante:
Los contenidos de la materia se incorporan y fijan mejor cuando uno intenta aplicarlos - no alcanza con ver un ejercicio resuelto por alguien más. Para sacar el máximo provecho de los ejercicios, es importante asistir a las consultas de práctica habiendo intentado resolverlos (tanto como sea posible). De esa manera, las consultas estarán más enfocadas y el docente podrá dar un mejor feedback.

Ejercicio 1: Friday the 13th en Java

Nota: Para realizar este ejercicio, utilice el material adicional que se encuentra en el siguiente link. Allí encontrará un proyecto Maven que contiene el código fuente de las clases Biblioteca, Socio y VoorheesExporter.

La clase Biblioteca implementa la funcionalidad de exportar el listado de sus socios en formato JSON. Para ello define el método **`exportarSocios()`** de la siguiente forma:

```
/**
 * Retorna la representación JSON de la colección de socios.
 */
public String exportarSocios() {
    return exporter.exportar(socios);
}
```

La Biblioteca delega la responsabilidad de exportar en una instancia de la clase VoorheesExporter que dada una colección de socios, retorna un texto con la representación de la misma en formato JSON. Esto lo hace mediante el mensaje de instancia **`exportar(List<Socio>)`**.

De un socio se conoce el nombre, el email y el número de legajo. Por ejemplo, para una biblioteca que posee una colección con los siguientes socios:

• Nombre: Arya Stark • e-mail:needle@stark.com • legajo: 5234-5	• Nombre: Tyron Lannister • e-mail:tyron@thelannisters.com • legajo: 2345-2

Ud. puede probar la funcionalidad ejecutando el siguiente código:

```
Biblioteca biblioteca = new Biblioteca();
biblioteca.agregarSocio(new Socio("Arya Stark", "needle@stark.com", "5234-5"));
biblioteca.agregarSocio(new Socio("Tyron Lannister", "tyron@thelannisters.com",
"2345-2"));
System.out.println(biblioteca.exportarSocios());
```

Al ejecutar, el mismo imprimirá el siguiente JSON:

```
[
    {
        "nombre": "Arya Stark",
        "email": "needle@stark.com",
        "legajo": "5234-5"
    },
    {
        "nombre": "Tyron Lannister",
        "email": "tyron@thelannisters.com",
        "legajo": "2345-2"
    }
]
```

Note los corchetes de apertura y cierre de la colección, las llaves de apertura y cierre para cada socio y la coma separando a los socios.

Tareas:

1. Analice la implementación de la clase Biblioteca, Socio y VoorheesExporter que se provee con el material adicional de esta práctica (Archivo biblioteca.zip).
2. Documente la implementación mediante un diagrama de clases UML.
3. Programe los Test de Unidad para la implementación propuesta.

Ejercicio 1.b: Usando la librería JSON.simple

Su nuevo desafío consiste en utilizar la librería JSON.simple para imprimir en formato JSON a los socios de la Biblioteca en lugar de utilizar la clase VoorheesExporter. Pero con la siguiente condición: **nada de esto debe generar un cambio en el código de la clase Biblioteca.**

La librería JSON.simple es liviana y muy utilizada para leer y escribir archivos JSON.
Entre las clases que contiene se encuentran:

- JSONObject : Usada para representar los datos que se desean exportar de un objeto. Esta clase provee el método **`put(Object, Object)`** para agregar los campos al mismo. Aunque el primer argumento sea de tipo Object, usted debe proveer el nombre del atributo como un string. El segundo argumento contendrá el valor del mismo. Por ejemplo, si point es una instancia de JSONObject, se podrá ejecutar point.put("x", 50);

- JSONArray: Usada para generar listas. Provee el método **`add(Object)`** para agregar los elementos a la lista, los cuales, para este caso, deben ser JSONObject.

Ambas clases implementan el mensaje **`toJSONString()`** el cual retorna un String con la representación JSON del objeto.

- JSONParser : Usada para recuperar desde un String con formato JSON los elementos que lo componen.

Tareas:

1. Instale la librería JSON.simple agregando la siguiente dependencia al archivo pom.xml de Maven

```
<dependency>
    <groupId>com.googlecode.json-simple</groupId>
    <artifactId>json-simple</artifactId>
    <version>1.1.1</version>
</dependency>
```

2. Utilice esta librería para imprimir, en formato JSON, los socios de la Biblioteca en lugar de utilizar la clase VoorheesExporter, sin que esto genere un cambio en el código de la clase Biblioteca.
    a. Modele una solución a esta alternativa utilizando un diagrama de clases UML. Si utiliza patrones de diseño indique los roles en las clases utilizando estereotipos.
    b. Implemente en Java la solución incluyendo los tests que crea necesarios.

3. Investigue sobre la librería Jackson, la cual también permite utilizar el formato JSON para serializar objetos Java. Extienda la implementación para soportar también esta librería.

Ejercicio 2: 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.

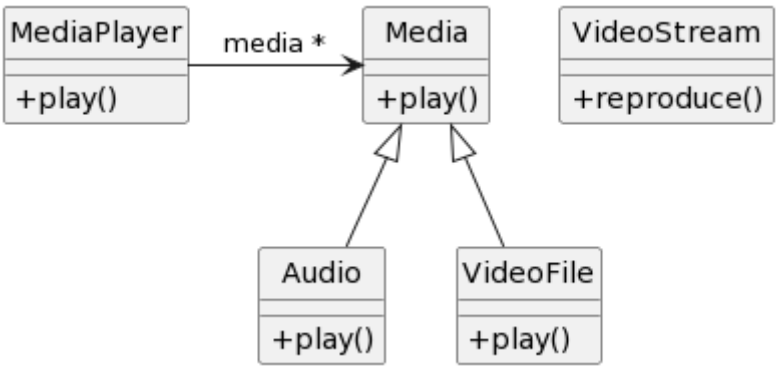
	Temporario	Pasante	Planta
básico	$ 20.000 + cantidad de horas que trabajó * $ 300.	$20.000	$ 50.000
adicional	$5.000 si está casado $2.000 por cada hijo	$2.000 por examen que rindió	$5.000 si está casado $2.000 por cada hijo $2.000 por cada año de antigüedad
descuento	13% del sueldo básico 5% del sueldo 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.

Ejercicio 3: Media Player

Usted ha implementado una clase Media player, para reproducir archivos de audio y video en formatos que usted ha diseñado. Cada Media se puede reproducir con el mensaje play(). Para continuar con el desarrollo, usted desea incorporar la posibilidad de reproducir Video Stream. Para ello, dispone de la clase VideoStream que pertenece a una librería de terceros y usted no puede ni debe modificarla. El desafío que se le presenta es hacer que la clase MediaPlayer pueda interactuar con la clase VideoStream.
La situación se resume en el siguiente diagrama UML:



Tareas:

1. Modifique el diagrama de clases UML para considerar los cambios necesarios. Si utiliza patrones de diseño indique los roles en las clases utilizando estereotipos.
2. Implemente en Java

Ejercicio 4: ToDoItem

Se desea definir un sistema de seguimiento de tareas similar a Jira[1].
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 el ToDoItem 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.
	*/
```

[1] https://www.atlassian.com/es/software/jira

```
    public void togglePause()

    /**
    * Pasa el ToDoItem a finished, siempre y cuando su estado actual sea
    * in-progress o paused. Si se encuentra en otro estado, no hace nada.
    */
    public void finish()

    /**
    * Retorna el tiempo que transcurrió desde que se inició el ToDoItem (start)
    * hasta que se finalizó. En caso de que no esté finalizado, el tiempo que
    * haya transcurrido hasta el momento actual. Si el ToDoItem no se inició,
    * genera un error informando la causa específica del mismo.
    */
    public Duration workedTime()


    /**
    * Agrega un comentario al ToDoItem 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 5: 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 similares a alguna película que reprodujo, ordenadas de más a menos reciente.

(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.

Ejercicio 6: Excursiones

Sea una aplicación que ofrece excursiones como por ejemplo “dos días en kayak bajando el Paraná”. Una excursión posee nombre, fecha de inicio, fecha de fin, punto de encuentro, costo, cupo mínimo y cupo máximo.

La aplicación ofrece las excursiones pero éstas sólo se realizan si alcanzan el cupo mínimo de inscriptos. Un usuario se inscribe a una excursión y si aún no se alcanzó el cupo mínimo, la inscripción se considera provisoria. Luego, cuando se alcanza el cupo mínimo, la inscripción se considera definitiva y podrá llevarse a cabo. Finalmente, cuando se alcanza el cupo máximo, la excursión solo registrará nuevos inscriptos en su lista de espera.

De los usuarios inscriptos, la aplicación registra su nombre, apellido e email.

Por otro lado, en todo momento la excursión ofrece información de la misma, la cual consiste en una serie de datos que varían en función de la situación.

- Si la excursión no alcanza el cupo mínimo, la información es la siguiente: nombre, costo, fechas, punto de encuentro, cantidad de usuarios faltantes para alcanzar el cupo mínimo.
- Si la excursión alcanzó el cupo mínimo pero aún no el máximo, la información es la siguiente: nombre, costo, fechas, punto de encuentro, los mails de los usuarios inscriptos y cantidad de usuarios faltantes para alcanzar el cupo máximo.
- Si la excursión alcanzó el cupo máximo, la información solamente incluye nombre, costo, fechas y punto de encuentro.

En una primera versión, al no contar con una interfaz de usuario y a los efectos de *debugging,* este comportamiento puede implementarlo en un método que retorne un String con la información solicitada.

Tareas:

1.- Realice un diseño UML. Si utiliza algún patrón indique cuál(es) y justifique su uso.

2.- Implemente lo necesario para instanciar una excursión y para instanciar un usuario.

3.- Implemente los siguientes mensajes de la clase Excursion:

(i) public void inscribir (Usuario unUsuario)

(ii) public String obtenerInformacion().

4.- Escriba un test para inscribir a un usuario en la excursión “Dos días en kayak bajando el Paraná”, con cupo mínimo de 1 persona y cupo máximo 2, con dos personas ya inscriptas. Implemente todos los mensajes que considere necesarios.

Ejercicio 7: Calculadora

Se desea diseñar e implementar una calculadora que realice operaciones matemáticas básicas, similar a las calculadoras tradicionales. Esta calculadora permitirá al usuario ingresar valores numéricos y realizar operaciones de suma, resta, multiplicación y división. Además, contará con la posibilidad de borrar la entrada actual y reiniciar los cálculos.
La calculadora debe responder a los siguientes mensajes

```
/**
   * Devuelve el resultado actual de la operación realizada.
   * Si no se ha realizado ninguna operación, devuelve el valor acumulado.
   * Si la calculadora se encuentra en error, devuelve “error”
 */
public String getResultado() {...}


/**
 * Pone en cero el valor acumulado y reinicia la calculadora
 */
public void borrar() {...}


/**
 * Asigna un valor para operar.
 * si hay una operación en curso, el valor será utilizado en la operación
 */
public void setValor(double unValor) {...}


 /**
 * Indica que la calculadora debe esperar un nuevo valor.
 * Si a continuación se le envía el mensaje setValor(), la calculadora sumará el valor recibido como parámetro, al valor actual y guardará el resultado
```

```
 */
public void mas() {...}
```

Además, se debe tener en cuenta el siguiente comportamiento:

- Los mensajes menos(), por() y dividido(), actúan de manera similar al mensaje mas, pero realizan las operaciones correspondientes de resta, multiplicación y división .
- La calculadora entra en error en caso de que se intente dividir por cero.
- Si la calculadora está esperando un valor, y se le envía cualquier otro mensaje, entonces entra en error.
- Si se le envió previamente un mensaje de operación (mas, menos..) pero luego no se le envía el mensaje setValor(), la calculadora entra en error.
- Solo sale de error si se le envía el mensaje borrar().
- Cuando la calculadora está en error, el mensaje resultado() retorna el string Error.

El siguiente código muestra el uso de la clase Calculadora

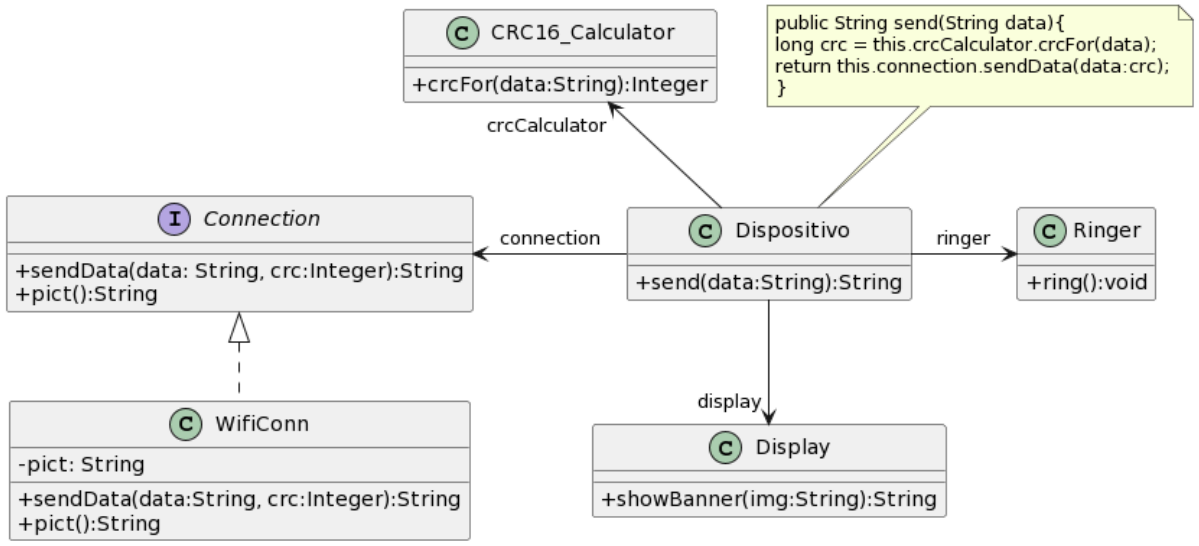
```
Calculadora calc = new Calculadora();
calc.setValor(5); // Establece el valor inicial
calc.mas(); // Prepara para sumar
calc.setValor(3); // Suma 3 al valor acumulado
System.out.println(calc.resultado()); // Imprimirá "8.0"
calculadora.por();
calculadora.setValor(2);
assertEquals(calculadora.resultado(), "16.0");
```

Tareas:

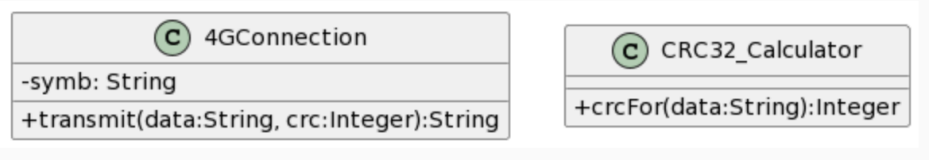
1. Realice un diseño UML. Si utiliza algún patrón indique cuál(es) y justifique su uso.
2. Programe su solución en Java. Debe implementarse respetando todas las buenas prácticas de diseño y programación de POO.
3. Programe los test para la implementación propuesta.

Ejercicio 8: 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:
    a. `send`
    b. `conectarCon`
    c. `configurarCRC`

En cuanto a CRC16_Calculator, puede utilizar la siguiente implementación:

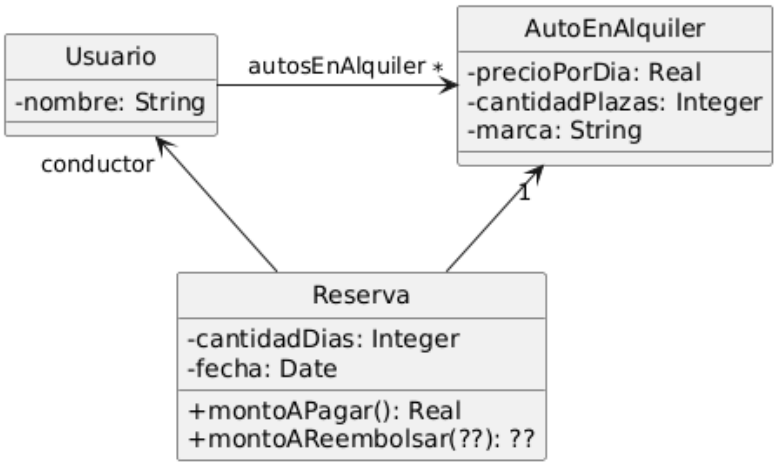
```
public long crcFor(String datos) {
	int crc = 0xFFFF;
	for (int j = 0; j < datos.getBytes().length; j++) {
		crc = ((crc >>> 8) | (crc << 8)) & 0xffff;
		crc ^= (datos.getBytes()[j] & 0xff);
		crc ^= ((crc & 0xff) >> 4);
		crc ^= (crc << 12) & 0xffff;
		crc ^= ((crc & 0xFF) << 5) & 0xffff;
	}
	crc &= 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 9: Alquiler de automóviles

En un sistema de alquiler de automóviles se quiere introducir funcionalidad para calcular el monto que será reembolsado (devuelto) si se cancela una reserva. Dicho reembolso podrá variar con respecto al monto total pagado, de acuerdo a la política de cancelación que sea determinada para el vehículo.



Se parte del siguiente diseño al que se necesita agregar la funcionalidad antes mencionada. El monto a pagar por una reserva se calcula como el precio por día del auto del cual se hizo la reserva, multiplicado por la cantidad de días.

Cada automóvil debe tener una política de cancelación que puede ser una de tres: flexible, moderada o estricta. Dichas políticas pueden cambiar con el tiempo en cualquier momento.

Se quiere calcular el monto a reembolsar de una reserva si se hiciera una cancelación. Dada una fecha tentativa de cancelación, se debe devolver el monto que sería reembolsado. El cálculo se hace de la siguiente manera.

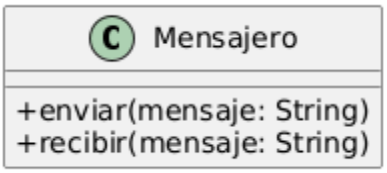
a) Si el automóvil tiene política de cancelación flexible, se reembolsará el monto total sin importar la fecha de cancelación (que de todas maneras debe ser anterior a la fecha de inicio de la reserva).
b) Si el automóvil tiene política de cancelación moderada, se reembolsará el monto total si la cancelación se hace hasta una semana antes y 50% si se hace hasta 2 días antes.
c) Si el automóvil tiene política de cancelación estricta, no se reembolsará nada (0, cero) sin importar la fecha tentativa de cancelación.

Tareas:

1. Modifique el diagrama de clases UML para considerar los cambios necesarios. Indique el patrón de diseño utilizado y las ventajas de su uso en este diseño en particular.
2. Muestre los roles del patrón utilizado en el diseño realizado.
3. Implemente en Java
4. Muestre en un snippet de código Java cómo crear un automóvil con una política de cancelación flexible y luego imprima en pantalla el valor de reembolso. Luego, cambie la política a cancelación moderada y vuelva a imprimir en pantalla el valor de reembolso.

Ejercicio 10: Mensajero

En un sistema de mensajería instantánea (similar a WhatsApp), los mensajes se transmiten entre dispositivos a través de la red. Para evitar que los mensajes puedan ser interceptados y leídos por terceros, se busca incorporar un mecanismo de cifrado: el mensaje se cifra antes de ser enviado y se descifra al recibirlo.
Actualmente, el diseño del sistema cuenta con una clase **Mensajero** que permite enviar y recibir mensajes de texto, como se muestra en el siguiente diagrama UML:

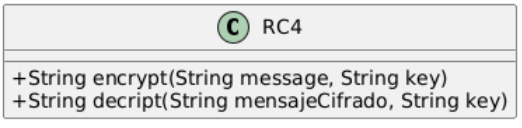


Se desea extender este diseño para que el mensajero pueda utilizar diferentes algoritmos de cifrado, de forma intercambiable.

En este ejercicio se trabajará con dos algoritmos concretos: **FeistelCipher** y **RC4**, aunque el diseño debe contemplar la posibilidad de incorporar nuevos algoritmos en el futuro.

Cada algoritmo maneja las claves de cifrado de manera distinta:

- **FeistelCipher**: Requiere una clave en el momento de la creación del objeto. Luego utiliza esa clave internamente para cifrar y descifrar mensajes. Este algoritmo utiliza encode para cifrar y el mismo mensaje para descifrar
- **RC4**: Necesita que la clave se proporcione cada vez que se realiza una operación de cifrado o descifrado.



Nota: Para realizar este ejercicio, utilice el material adicional que se encuentra en el siguiente LINK. Allí encontrará un proyecto Maven que contiene el código fuente de las clases *FeistelCipher y RC4*. **Estas clases no deben ser modificadas**

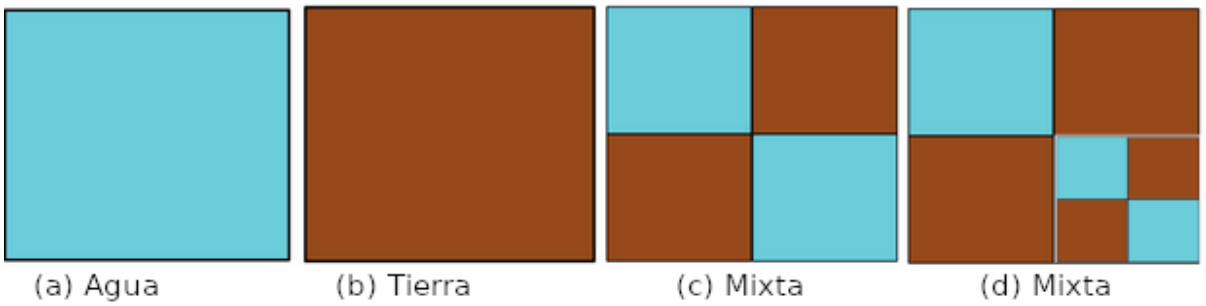
Tareas:

1. Diseñe una solución que permita al mensajero utilizar cualquiera de los algoritmos de cifrado de manera intercambiable. Sí la solución utiliza patrones de diseño indique cuales, y marque con estereotipos en el diagrama UML los roles de los participantes.
2. Al momento de enviar un mensaje, ¿con cuantos algoritmos de cifrado puede trabajar el mensajero al mismo tiempo?
3. Escriba un **ejemplo del código Java** necesario para instanciar un mensajero que envía un mensaje con cifrado **FeistelCipher** al que luego se le cambia la forma de cifrar a **RC4** y envía el mismo mensaje.
4. Implemente la solución en Java.

Ejercicio 11: Topografías

Un uso común de imágenes satelitales es el estudio de las cuencas hídricas que incluye saber la proporción entre la parte seca y la parte bajo agua. En general las imágenes satelitales están divididas en celdas. Las celdas son imágenes digitales (con píxeles) de las cuales se quiere extraer su “topología”.
Un objeto Topografía representa la distribución de agua y tierra de una celda satelital, 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 solamente por tierra.
(c) y (d) topografías mixtas.



Una topografía mixta está formada por partes de agua y partes de tierra (4 partes en total). Estas 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.

Tareas:

1. Diseñe e implemente las clases necesarias para que sea posible:
    a. crear Topografías,
    b. calcular su proporción de agua y tierra,
    c. 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 11b: 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 12: FileSystem

Un File System es un componente que forma parte del sistema operativo. Este está estructurado jerárquicamente en forma de árbol, comenzando con un directorio raíz.
Los elementos del file system pueden ser directorios o archivos. Los archivos contienen datos y los directorios contienen archivos u otros directorios. De cada archivo se conoce el nombre, fecha de creación y tamaño en bytes. De cada directorio se conoce el nombre, fecha de creación y contenido (el tamaño es siempre la cantidad inicial de 32kb más la suma del tamaño de su contenido). Modele el file system y provea la siguiente funcionalidad:

```
public class FileSystem {
	/**
	* Retorna el espacio total ocupado, incluyendo todo su contenido.
	*/
```

```
        public int tamanoTotalOcupado()

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

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

        /**
        * Retorna el primer elemento con el nombre solicitado contenido en
        cualquier
        * nivel del filesystem
        */
        public ?? buscar(String nombre)

        /**
        * Retorna la lista con los elementos que coinciden con el nombre solicitado
        * contenido en cualquier nivel del filesystem
        */
        public List<??> buscarTodos(String nombre)

        /**
        * Retorna un String con los nombres de los elementos contenidos en todos
los
        * niveles del filesystem. De cada elemento debe retornar el path completo
        * (similar al comando pwd de linux) siguiendo el modelo presentado a
        * continuación
              /Directorio A
              /Directorio A/Directorio A.1
              /Directorio A/Directorio A.1/Directorio A.1.1
              /Directorio A/Directorio A.1/Directorio A.1.2
              /Directorio A/Directorio A.2
              /Directorio B
        */
        public String listadoDeContenido()

}
```

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.
3. Implemente completamente en Java.

Ejercicio 13 - SubteWay

Sugerencia: no resuelva este ejercicio en ayunas.

Una cadena de comidas rápidas especializada en sándwiches necesita resolver el cálculo de precios de éstos. El cálculo es simple: el precio de un sándwich equivale a la suma del precio de cada uno de sus componentes; el problema es la dificultad para representar y crear cada uno de los sándwiches distintos.

Existen cuatro sandwiches distintos, pero podrían aparecer nuevos en el futuro.

Clásico: consta de pan brioche (100 pesos), aderezo a base de mayonesa (20 pesos), principal de carne de ternera(300 pesos) y adicional de tomate (80 pesos).

Vegetariano: consta de pan con semillas (120 pesos), sin aderezo, principal de provoleta grillada (200 pesos) y adicional de berenjenas al escabeche (100 pesos).

Vegano: consta de pan integral (100 pesos), aderezo de salsa criolla (20 pesos), principal de milanesa de girgolas (500 pesos), sin adicional.

Sin TACC: consta de pan de chipá (150 pesos), aderezo de salsa tártara (18 pesos), principal de carne de pollo (250 pesos) y adicional de verduras grilladas (200 pesos).

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. Escriba un script de código que permita crear y calcular el precio de cada una de las alternativas de sándwiches.
3. Implemente completamente en Java.