Trabajo Práctico individual de presentación obligatoria.

Continuando con el **Trabajo Práctico N° 2**, agregue los siguientes ejercicios al proyecto java creado en la clase anterior.

Continue con el versionamiento del código utilizando el repositorio creado para este trabajo práctico.

Desde Spring tool suite crear una rama para cada ejercicio, nombrar a cada rama de la siguiente forma:

feature/ejercicio5

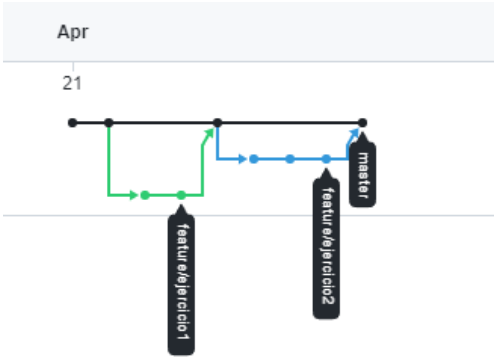
feature/ejercicio6

feature/ejercicio7

Desarrollar el ejercicio e ir subiendo los cambios sobre la rama creada, cuando finalice el desarrollo fusionar la rama de ejercicio con la rama main (master) remota (ejecutar el merge pull request en github).

Mantenga actualizada la rama master local.

Por cada ejercicio nuevo se debe generar una nueva rama a partir de la rama master local actualizada.



5. Uso de Interface.

Dentro del paquete creado para este ejercicio cree los siguientes paquetes: **main**, **model** e **interfaces**

Dentro del paquete **interfaces** cree una interfaz con el nombre **Pago** que contenga los siguientes métodos:

realizarPago(double monto): Método que toma un monto como parámetro y realiza el pago correspondiente.

imprimirRecibo(): Método que imprime un recibo del pago realizado.

En el paquete model crear las clases: **PagoTarjeta** y **PagoEfectivo**, estas clases deben implementar la interfaz **Pago**, es decir que deben implementar los métodos realizarPago() e imprimirRecibo().

Los atributos para PagoTarjeta son:

Número de tarjeta (String)

Fecha de pago (LocalDate)

Monto pagado (double)

Los atributos para PagoEfectivo son:

Monto pagado (double)

Fecha de pago (LocalDate)

Para el pago con tarjeta en el método realizarPago(), se actualiza el valor del atributo montoPagado con el monto más el 15% de recarga.

Para el pago con tarjeta el método imprimirRecibo() muestra la información del objeto PagoTarjeta:

Número de tarjeta: 4021123454786010
Fecha de pago: 26/04/24
Monto pagado: 645000,75

Para el pago con efectivo, en el método realizarPago(), se actualiza el valor del atributo montoPagado con el monto menos un 10% de descuento.

Para el pago con efectivo el método imprimirRecibo() muestra la información del objeto PagoEfectivo:

Fecha de pago: 27/04/24
Monto pagado: 35000,50

Dentro del paquete main, cree la clase Main con el correspondiente método main().

Realice la precarga de un arrayList, donde guarde 15 productos de la clase **Producto** utilizada en el ejercicio 1, agregándole el atributo estado de tipo boolean (este atributo indica true para productos disponibles y false para productos que no tienen stock).

Muestre al usuario el siguiente menú de opciones:

1 – Mostrar productos

2 – Realizar compra

3 – Salir

En la opción 2, luego de seleccionar los productos que se compran (almacenarlos en un arrayList), muestre al usuario las 2 opciones de pago:

1 – Pago efectivo

2- Pago con tarjeta.

El monto que recibe el método realizarPago(double monto), es la sumatoria de los precios de los productos que se seleccionaron para la compra.

Luego muestre el resultado de invocar al método imprimirRecibo().

Realizar el manejo de excepciones de forma conveniente.

6. Uso de @FunctionalInterface.

Este ejercicio tiene como objetivo ejemplificar el uso de una interfaz funcional.

Dentro del paquete creado para este ejercicio cree los siguientes paquetes: **main**, **model**, **interfaces.funcionales**

Dentro del paquete model crear las clases java: **FelinoDomestico** y **FelinoSalvaje**, los objetos de ambas clases conservan similitud en sus atributos y algunos comportamientos. Para el presente ejercicio declare los siguientes atributos en ambas clases:

- Nombre (String)
- Edad (byte)
- Peso (float)

En el paquete **interfaces.funcionales** cree una interfaz el nombre **Converter**, utilice la anotación @FunctionalInterface la declaración de la interfaz debe quedar de esta forma:

```
@FunctionalInterface
public interface Converter<T,T1> {
    T1 convert(T t);


    static <T> boolean isNotNull(T t){
           return t != null;
    }


    default void mostrarObjeto(T1 t1) {
           System.out.println("Objeto - " + t1.toString());
    }

}
```

Dentro del paquete main, cree la clase **Main** con el correspondiente método main().

Utilice la interfaz funcional para realizar la conversión de felinos.

Ejemplo:

Garfield (Felino doméstico) está manifestando comportamientos salvajes, entonces vamos a realizar la conversión a felino salvaje de la siguiente forma:

```
FelinoDomestico gato = new FelinoDomestico(“Garfield”, (byte)45, 12f);

//definición de expresión lambda que define el convertidor de FelinoDomestico a
//FelinoSalvaje.

Converter<FelinoDomestico, FelinoSalvaje> converter = x -> new FelinoSalvaje(x.getNombre(),
x.getEdad(), x.getPeso());

//se realiza la conversión

FelinoSalvaje felino1 = converter.convert(gato);

//mostramos los datos del objeto felino salvaje felino1

converter.mostrarObjeto(felino1);
```

En la consola se muestra:

Realice la conversión de un objeto felino salvaje a felino doméstico.

Nombre = Tanner
Edad = 20
Peso = 186

Antes de realizar la conversión utilice el método estático de la interfaz isNotNull() para verificar que el objeto a convertir no es nulo.

Utilice el método mostrarObjeto() para mostrar los datos del objeto.

7. Uso Stream Collection.

Dentro del paquete creado para este ejercicio cree el paquete **main**.

Cree la clase Main que contenga el método main().

Utilice un arrayList para realizar la precarga de 15 objetos de la clase **Producto** utilizada en el ejercicio 5

Luego confecciones un menú de opciones que se mantenga visible de forma iterativa, con las siguientes opciones:

1 – Mostrar productos (solo se muestran los productos con estado true).

2 – Mostrar los productos faltantes (muestra productos con estado false)

3 – Incrementar los precios de los productos en un 20%

4 – Mostrar los productos que corresponden a la categoría Electrohogar y estén disponibles para la venta.

5 – Ordenar los productos por precio de forma descendente.

6 - Mostrar los productos con los nombres en mayúsculas.

Para la opción 1 utilizar la interfaz funcional Consumer.

Para la opción 2 utilizar Predicate y filter().

Para la opción 3 utilizar Function, map(), el resultado debe almacenarse en un nuevo ArrayList productosIncrementados.

Para la opción 4 utilizar Predicate y filter().

Para la opción 5 utilizar sort() de la interface List, Comparator.comparing()

Para la opción 6 utilizar Function y map().