Producto 2. Implementación de clases en Java

Empezar actividad

Fecha de entrega 1 de nov por 23:59 **Puntos** 20 **Submitting** un archivo cargado **Intentos** 0 **Intentos permitidos** 1

Descripción

A partir de los diagramas desarrollados en el producto anterior, realizaremos un programa en Java que implemente estos diagramas aplicando el patrón de diseño MVC.

Objetivos

- El objetivo principal de la actividad es:
  - Implementar estructuras dinámicas de datos y el modelo estático de clases en lenguaje Java, aplicar el patrón de diseño MVC, y realizar pruebas unitarias utilizando *JUnit*, todo ello utilizando un sistema de control de versiones.

Pasos a seguir

Los pasos a seguir para llevar a cabo el producto son:

1. Leer detenidamente estas instrucciones e identificar los requisitos de la actividad.
2. Revisar detenidamente la rúbrica de evaluación.
3. Consultar los recursos necesarios facilitados en el aula.
4. Registrar y configurar una herramienta *Git*.
5. En el desarrollo del proyecto deberá utilizarse el sistema de control de versiones *Git*.
6. Implementar el modelo estático de clases en lenguaje Java.
7. En la implementación deberán utilizarse como mínimo las Clases Genéricas o Java Generics.
8. Elegir el Tipo de Colecciones en Java óptimo para cada caso en función de la necesidad o requisito a implementar.
9. Se debe tener en cuenta la correcta gestión de Excepciones, así como implementar Excepciones personalizadas.
10. Realizar un programa en Java en modo de consola que almacene la información en estructuras dinámicas de datos.
11. La estructura y requisitos de la creación de la aplicación en java se describen en el Anexo 1.
12. En el Anexo 2, se describen el contenido de las clases iniciales.
13. Realizar pruebas unitarias utilizando el entorno *JUnit*. Elegir dos métodos que apliquen lógica de negocio y realizar las pruebas unitarias con *JUnit* para evaluar su correcto funcionamiento.
14. Realiza un video mostrando la ejecución de todo el programa en todas sus opciones. Utiliza una herramienta de grabación de videos de captura de pantalla.

¿Qué debe contener el informe?

- Enlace al vídeo en youtube o la plataforma que hayáis elegido.
- Enlace a la página de Github de vuestro proyecto.
- Autoevaluación numérica entre 0 y 20 con un párrafo explicativo y justificativo para la nota. También podéis copiar y pegar la rúbrica que aparece en esta descripción y rellenarla según vuestro criterio.

Se requiere

Los requisitos indispensables para realizar el producto son:

1. Registrarse en la herramienta online *Git*.
2. Disponer del entorno de programación en Java.
3. Disponer del entorno *JUnit*.
4. Investigar otras fuentes de información para ampliar los recursos facilitados en el aula virtual.

Recursos Básicos

Para llevar a cabo la actividad puedes apoyarte en los materiales y fuentes de información disponibles el el documento "**Recursos de aprendizaje asociados a los productos (https://aula.uoc.edu/courses/11010/pages/recursos-de-aprendizaje-vinculados-a-los-productos)** ".

Criterios de evaluación ✔

La puntuación máxima del producto es un 20.
La puntuación mínima para superarlo es de 10 sobre 20.

Indicaciones para la entrega de la actividad ✔✔

Para la entrega de la actividad deberás:

Hacerlo por el canal indicado en el aula virtual.

Formato: zip o rar, en el contenido se deberá incluir el documento PDF con la resolución de los ejercicios y los ficheros del programa en java así como el resto de ficheros relacionados en este producto.

Nombre del archivo: producto2_FP058_NombreGrupo

Extensión recomendada: 5-7 páginas.

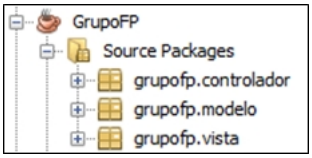
Anexo

Creación del proyecto

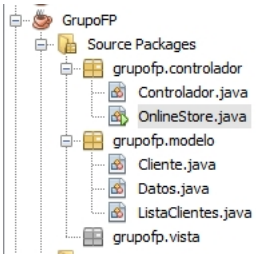
La aplicación debe ser de tipo *Java Aplicattion* y el nombre del proyecto el nombre del grupo. En los ejemplos que se muestran a continuación utilizaremos el nombre de grupo *GrupoFP*.

Paquetes del Proyecto

El proyecto que se ha de desarrollar estará basado en el patrón Modelo-Vista-Controlador (MVC). En la estructura del patrón MVC, toda la interacción con el usuario se realiza a través de las clases de la Vista, así pues, las instrucciones para introducir datos por teclado o mostrar datos por pantalla estarán ubicadas en las clases de la Vista y en ningún caso se realizará en las clases del Controlador ni del Modelo. Las clases de la Vista no deben acceder directamente a las clases del Modelo, sino que lo hará a través de la clase del Controlador. En consecuencia, el proyecto estará formado por 3 paquetes:



- **grupofp.vista**: la vista contendrá todas las clases relacionadas con el manejo del menú de opciones, así como la entrada y visualización de los datos. Este paquete contendrá las clases OnlineStore y GestionOS.
- **grupofp.controlador**: el paquete controlador únicamente contendrá la clase Controlador, que hará de puente entre la vista y el modelo. La vista sólo podrá utilizar esta clase para acceder a la información del modelo.
- **grupofp.modelo**: el modelo contendrá todas las clases que modelan los datos que deben gestionarse dentro de la aplicación. Dentro del modelo, merece especial mención la clase Datos, que contendrá todos los datos de la aplicación y llevará a cabo todas las acciones que afectan a las mismas. Este paquete contendrá las clases Datos, Cliente, ListaClientes así como el resto de clases necesarias en la capa Modelo.



Clases del Proyecto

A continuación, se describen las principales clases que deben implementarse:

Clase Artículo

La clase Artículo almacena toda la información relevante de los artículos.

Se debe añadir un constructor que permita inicializar los valores, los *getters* y *setters* y el *toString*.

Clase Cliente y Clases Heredadas

La clase Cliente guardará la información de los clientes. Esta información incluye el correo electrónico, nombre y dirección de cada cliente. Debe declararse la clase Cliente como clase abstracta, incorporando un constructor que inicialice sus atributos, así como los *getters* y *setters* necesarios. Además, será

necesario declarar los métodos abstractos, que devuelven el tipo de cliente y los que calculan la cuota anual de cada cliente dependiendo del tipo de cliente, así como el descuento de gastos de envío:

```
public abstract String tipoCliente();

public abstract float calcAnual();

public abstract float descuentoEnv();
```

Las clases hijas, ClienteEstandard y ClientePremium, implementarán los métodos abstractos indicados.

Además, también se necesita definir un método *toString* con los datos de todos los atributos.

Clase Pedido

La clase Pedido almacena toda la información relacionada con cada pedido.

El articulo y el cliente del pedido no deben ser de tipo *String*, sino que deben ser de tipo Articulo y Cliente respectivamente.

Se debe añadir un constructor que permita inicializar los valores, los *getters* y *setters* y el *toString*.

También debe contener los métodos siguientes:

```
public boolean pedidoEnviado();

public float precioEnvio();
```

El método *toString* debe construir una cadena con los datos siguientes: número de pedido, fecha y hora del pedido, Nif y nombre del cliente, código y descripción del artículo, cantidad, precio artículo, coste envío, precio total y si ya se ha enviado.

Clase Lista

La aplicación que debemos implementar debe gestionar colecciones de tres tipos de objetos: Artículo, Cliente y Pedido. Dado que las operaciones que necesitamos realizar son muy similares para las tres colecciones, deberemos implementar la mayoría de estas operaciones en una clase genérica, y a partir de ella derivar clases para las tres listas que se necesitan.

Las clases genéricas en Java constituyen una herramienta muy útil para reutilizar software. En concreto, son clases similares a las clases convencionales pero que reciben tipos como parámetro a la hora de instanciarse. El esqueleto de la clase genérica que se debe desarrollar en este producto se muestra en el Anexo II y se declara de la siguiente forma:

```
public class Lista<T> {

protected ArrayList<T> lista;

...

}
```

En la declaración anterior, T constituye un parámetro que incide en el tipo de elemento que se gestiona en el atributo lista, de tipo *ArrayList*.

A partir de la clase Lista, se definirán las clases derivadas: ListaArticulos, ListaClientes y ListaPedidos, según el tipo de objeto a gestionar. Por ejemplo, la definición de ListaArticulos se realizará de la siguiente forma:

```
public class ListaArticulos extends Lista<Articulo>{....}
```

Clase Datos

La clase Datos es la clase principal del paquete del modelo, puesto que contiene y gestiona todos los datos de la aplicación y es el enlace entre el controlador y el resto de las clases del modelo ya que el controlador solo llamará a los métodos de esta clase.

Clase Controlador

La clase Controlador será utilizada para intervenir entre la vista y los datos del modelo. Con este objetivo, la clase Controlador utilizará una instancia de la clase Datos, por ser la clase principal de la capa modelo.

ANEXO II (https://aula.uoc.edu/courses/11010/files/537537?wrap=1) ↓ (https://aula.uoc.edu/courses/11010/files/537537/download?download_frd=1)

FP058_Rubrica_Producto2

Criterios	Calificaciones				Puntos
Control de versiones con Git	**2 pts** **Implementa el control de versiones usando Git correctamente y de forma optimizada.**	**1,5 pts** **Implementa el control de versiones usando Git correctamente.**	**1 pts** **Implementa el control de versiones usando Git con algunos errores.**	**0,5 pts** **Presenta dificultades en su conjunto en la implementación del control de versiones usando Git.**	2 puntos
Generación de código a partir de un diagrama de clases	**6 pts** **Genera el código adecuado a partir de un diagrama de clases de forma optimizada.**	**4,5 pts** **Genera el código adecuado a partir de un diagrama de clases.**	**3 pts** **Genera código a partir de un diagrama de clases con algún error.**	**1,5 pts** **Presenta dificultades en su conjunto en la generación del código a partir de un diagrama de clases.**	6 puntos
Colecciones en Java	**6 pts** **Implementa las colecciones en Java óptimas en función de los requerimientos correctamente y de forma optimizada.**	**4,5 pts** **Implementa las colecciones en Java óptimas en función de los requerimientos correctamente.**	**3 pts** **Implementa las colecciones en Java óptimas en función de los requerimientos con algún error.**	**1,5 pts** **Presenta dificultades en su conjunto en la implementación de las colecciones en Java óptimas en función de los requerimientos.**	6 puntos
Patrón de diseño MVC	**4 pts** **Implementa el patrón de diseño MVC correctamente y de forma optimizada.**	**3 pts** **Implementa el patrón de diseño MVC correctamente.**	**2 pts** **Implementa el patrón de diseño MVC con algún error.**	**1 pts** **Presenta dificultades globales en la implementación del patrón de diseño MVC.**	4 puntos
Pruebas unitarias utilizando JUnit	**2 pts** **Realiza pruebas unitarias utilizando JUnit correctamente y de forma optimizada.**	**1,5 pts** **Realiza pruebas unitarias utilizando JUnit correctamente.**	**1 pts** **Realiza pruebas unitarias utilizando JUnit con algún error.**	**0,5 pts** **Presenta dificultades globales para realizar pruebas unitarias utilizando JUnit.**	2 puntos
					Puntos totales: 20