PROYECTO DE PROGRAMACIÓN ORIENTADO A OBJETOS – GESTIÓN TARJETAS DE CRÉDITO

Descripción del proyecto

En este proyecto tendrás que desarrollar una **aplicación Java en modo consola que permita gestionar un conjunto de tarjetas de crédito** empleando los fundamentos de la Programación Orientada a Objetos. La descripción completa y detallada de la implementación de este proyecto se encuentra en las siguientes secciones.

Debes aplicar todas las buenas prácticas aprendidas, entre otras:

- Implementar los métodos de las clases realizando control de errores.
- Implementar los constructores y métodos de la clase reutilizando todo el código posible.
- Implementar las clases sin modificar la interfaz pública de la misma.
- Validar todos los datos de entrada, tanto los introducidos por teclado por el usuario, como los parámetros de las funciones.
- Evitar código duplicado y reutilizar código mediante funciones.
- Nombrar las variables, clases y métodos de acuerdo a las convenciones del lenguaje, y usar identificadores descriptivos y representativos de lo que almacena o realiza la variable/función.
- Seguir las especificaciones descritas en este guion sin cambiarlas, salvo justificación y previa consulta con el profesor.

Importante: No se corregirá ningún proyecto en el que haya errores de compilación.

La tarea se califica entre 0 y 10 puntos desglosados de la siguiente manera:

- Clase ***Movimiento*** (1,5 puntos).
- Clase ***TarjetaCredito*** (3,5 puntos).
- Documentación Javadoc de las clases (1 punto)
- Programa principal ***AplicacionTarjetasCredito*** (4 puntos).

Se trata de desarrollar una aplicación Java en consola que permita gestionar una lista de tarjetas de crédito. El programa mostrará un primer menú con las siguientes opciones:

1. **Crear tarjeta de crédito**.
2. **Eliminar tarjeta de crédito**.
3. **Gestionar tarjeta de crédito**.
4. **Consultar gastos totales**.
5. **Salir del programa**.

En caso de seleccionar el menú de gestión de tarjeta de crédito, se mostrará el siguiente **submenú** para la tarjeta seleccionada.

1. **Mostrar el número de tarjeta completo.**
2. **Mostrar el nombre del titular de la tarjeta.**
3. **Mostrar la fecha de caducidad.**
4. **Modificar el PIN.**
5. **Realizar un pago.**
6. **Consultar movimientos.**
7. **Consultar gasto total.**
8. **Volver al menú principal.**

Se creará un proyecto llamado **GestionTarjetasCredito** que contendrá tres ficheros: **TarjetaCredito.java**, **Movimiento.java** y **AplicacionTarjetasCredito.java**.

- El primero de ellos contendrá la clase `TarjetaCredito` con los atributos y métodos necesarios para trabajar con un objeto ***TarjetaCredito***.
- El segundo contendrá la clase `Movimiento` con los atributos y métodos necesarios para trabajar con un objeto ***Movimiento***.
- El tercero contendrá la función principal *main* que gestionará la aplicación.

Especificación del diseño de las clases

A continuación se detalla el diseño de cada una de las clases de la aplicación.

La clase Movimiento

Un movimiento bancario se compone de los siguientes atributos privados: *cantidad*, *concepto* y *fecha*.

A continuación se muestran los requisitos que deben cumplir los atributos:

- La cantidad debe ser un valor positivo.
- El concepto es una cadena de máximo 50 caracteres de longitud.
- La fecha en la que se ha realizado el movimiento. Se establecerá automáticamente con la fecha actual del sistema.

Se implementarán los siguientes constructores y métodos públicos:

- **(0,5 puntos)** Constructor con los parámetros del objeto. Si alguno de los parámetros no es válido se lanzará la excepción `IllegalArgumentException`.

```
public Movimiento(double cantidad, String concepto);
```

- **(0,5 puntos)** Constructor copia.

```
public Movimiento(Movimiento m);
```

- **(0,10 puntos)** Métodos ***get*** para los atributos: *cantidad*, *concepto*, *fecha.*
- **(0,10 puntos)** Métodos ***set*** para los atributos: *concepto*. Se debe comprobar la validez de cada uno

de ellos y no modificar el atributo si no se verifica.

- **(0,10 puntos)** Método ***toString*** para representar en modo texto el contenido del objeto.
- **(0,20 puntos)** Método ***equals*** para comparar dos objetos de tipo *Movimiento*, que se consideran iguales si todos sus atributos son iguales.

La clase TarjetaCredito

Una tarjeta de crédito se compone de los siguientes atributos privados: *titular*, *nif*, *pin*, *limite*, *mesCaducidad*, *añoCaducidad*, *numeroTarjeta*, *CVV* y *movimientos.*

A continuación se muestran los requisitos que deben cumplir los atributos:

- El nombre del titular debe validarse adecuadamente mediante una expresión regular y debe tener una longitud máxima de 80 caracteres y una longitud mínima de 15 caracteres.
- El NIF debe validarse adecuadamente mediante una expresión regular y podrá ser tanto un NIF, CIF o NIE.
- El PIN debe validarse adecuadamente mediante una expresión regular y deberá tener una longitud mínima de 4 dígitos.
- El límite de gasto será un valor entre 500 y 5000 euros.
- El mes de caducidad coincidirá con el mes actual (se obtendrá de la fecha actual).
- El año de caducidad será 5 años después del año actual (se calculará a partir de la fecha actual).
- El número de tarjeta consta de 16 dígitos: el primer dígito identifica el sector al que pertenece la compañía que emite la tarjeta (1 y 2 aerolíneas, 3 Viajes y entretenimiento, 4 VISA, 5 Mastercard, 6 Discovery Card, 7 industrial del petroleo, 8 telecomunicaciones y 9 reservado para el país donde opera la tarjeta). Los seis dígitos siguientes identifican la tarjeta (INN), y los dígitos del 7 al 15 identifican de forma privada a la persona titular de la tarjeta. El último dígito es un dígito de control que ayuda a comprobar si todos los demás dígitos de la tarjeta son correctos. Este dígito de control se calcula mediante el algoritmo de Luhn.
- El número CVV será un valor aleatorio de 3 dígitos.
- El atributo movimientos será un array de 50 elementos de objetos *Movimiento*.

Información números de tarjetas de crédito
https://www.bbva.es/finanzas-vistazo/ef/tarjetas/numero-de-tarjeta-de-credito.html
https://www.giromatch.com/es/tarjetas-de-credito/numeros-de-tarjetas-de-credito
https://es.wikipedia.org/wiki/Algoritmo_de_Luhn

Se implementarán los siguientes constructores y métodos públicos:

- **(1 punto)** Constructor con los parámetros de la tarjeta de crédito. Si alguno de los parámetros y/o el número de tarjeta no es válido se lanzará la excepción `IllegalArgumentException`.

```
public TarjetaCredito(String nombre, String nif, String pin, int limite,
```

```
String numeroTarjeta);
```

- **(0,5 puntos)** Constructor copia.

```
public TarjetaCredito(TarjetaCredito tarjeta);
```

- **(0,10 puntos)** Métodos ***get*** para los atributos: *titular*, *nif*, *pin*, *limite*, *mesCaducidad*, *añoCaducidad*, *numeroTarjeta y CVV.*
- **(0,10 puntos)** Métodos ***set*** para los atributos: *pin, limite*. Se debe comprobar la validez de cada uno de ellos y no modificar el atributo si no se verifica.
- **(0,4 puntos)** Método **pagar**. Se realizará un pago con la tarjeta. La cantidad debe ser positiva, en caso contrario se lanzará la excepción `IllegalArgumentException`. Debe comprobar que con el nuevo pago no se supera el límite de gasto de la tarjeta. Este pago se añadirá como un nuevo movimiento de la tarjeta.

```
public boolean pagar(double cantidad, String concepto);
```

- **(0,20 puntos)** Método **gastado**. Devuelve la cantidad total gastada con la tarjeta.

```
public double gastado();
```

- **(0,20 puntos)** Método **movimientos**. Devuelve la información en formato String de los movimientos de la tarjeta. Recibe por parámetro el número de los últimos movimientos que se desean consultar. Si no es un valor válido se lanzará la excepción `IllegalArgumentException`.

```
public String movimientos(int numero);
```

- **(0,10 puntos)** Método **número de movimientos**. Devuelve cuántos movimientos se han realizado en la tarjeta.

```
public int numeroMovimientos();
```

- **(0,20 puntos)** Comprobar la validez de un número de tarjeta de crédito. El método devolverá true o false indicando si el número es válido o no.

```
public static boolean comprobarNumeroTarjeta(String numero);
```

- **(0,5 puntos)** Calcular el dígito de control de un número de tarjeta. El método devuelve el dígito de control dado el número de la tarjeta.

```
public static String obtenerDigitoControl(String numero);
```

- **(0,10 puntos)** Método **String toString()**. Devolverá una cadena con los datos del titular, el NIF, mes y año de caducidad, el número de la tarjeta, el límite de saldo y la cantidad de dinero que se ha gastado.
- **(0,10 puntos)** Método **boolean equals(TarjetaCredito t)**. Compara dos tarjetas de crédito, que serán iguales si tienen el mismo número de tarjeta.

Puedes implementar los **atributos y métodos privados** que creas necesarios para mejorar la encapsulación y evitar la duplicación de código.

Clase AplicacionTarjetasCredito

El programa principal se encarga de gestionar la solicitud de datos al usuario y la creación de los objetos en un **array de objetos *TarjetaCredito***. Para ello, el programa creará inicialmente un array vacío de **10 elementos de capacidad**. Durante la gestión del programa, se deberá comprobar en todo momento que no se supera el límite de creación de tarjetas de crédito, avisando al usuario en caso de que desee crear una nueva tarjeta de crédito y no haya más espacio de almacenamiento.

Las opciones de la aplicación se mostrarán en un menú como el mostrado al principio de esta tarea. Se implementará del siguiente modo:

- **(1 punto) Crear tarjeta de crédito**. Se llamará a una función (si hay espacio en el array) que se encargará de solicitar todos los datos al usuario y devolverá un objeto *TarjetaCredito.* El programa guardará el objeto en la última posición libre del array siempre y cuando no exista otra tarjeta creada con el mismo NIF.
- **(1,5 puntos) Eliminar tarjeta de crédito**. Se pedirá al usuario el NIF de la tarjeta que se desea borrar. Se utilizará una función que recibirá por parámetro el NIF y el array de tarjetas, y devolverá la posición del array donde se ha localizado (buscar una tarjeta). A continuación, se llamará a otra función para eliminar la tarjeta que recibirá por parámetro el array de tarjetas y el índice del elemento a borrar. La función de borrado deberá mover los objetos para ocupar el hueco y compactar el array.
- **(1,25 puntos) Gestionar tarjeta de crédito**. Pedirá al usuario el NIF de la tarjeta que desea gestionar. Para buscar la tarjeta se reutilizará la función creada anteriormente. Una vez seleccionado el objeto, se mostrará al usuario el submenú de opciones de gestión de tarjetas que se mostró al inicio de la descripción del proyecto. Este submenú es bastante autoexplicativo con lo que debe realizarse en cada una de sus opciones.
- **(0,25 puntos) Consultar gastos totales**. Se utilizará una función que recibe por parámetro el array de tarjetas y devolverá el gasto total realizado en el programa.
- **Salir del programa**. Finaliza la aplicación.

Se deben comprobar y manejar los errores de forma adecuada para evitar que el programa termine de forma inesperada. Por tanto se comprobará que el usuario introduce valores correctos en todo momento.

Se valorará el diseño modular y estructurado, la claridad y simplicidad del código y su documentación. Todas las validaciones de las cadenas de caracteres se realizarán mediante expresiones regulares. Para ello, se reutilizará lo máximo posible el código creando funciones en los lugares donde creas que son más adecuados para realizar esta tarea.

Entrega de la tarea

El proyecto se desarrollará con el IDE Netbeans. El nombre del proyecto, de las clases y métodos deben llamarse de la forma indicada en la descripción de la tarea. Se comprimirá la carpeta del proyecto Netbeans en un único fichero en formato .ZIP o 7z que se subirá al buzón de la tarea en la plataforma Moodle.

El archivo se nombrará siguiendo las siguientes pautas:

Apellido1_Apellido2_Nombre_PROG_ProyectoGestionTarjetasCredito.zip

Resultados de aprendizaje y criterios de evaluación relacionados

En esta actividad se evalúan los siguientes resultados de aprendizaje con los criterios de evaluación que se relacionan para cada uno de ellos:

- RA 3. Escribe y depura código, analizando y utilizando las estructuras de control del lenguaje.
  - a) Se ha escrito y probado código que haga uso de estructuras de selección.
  - b) Se han utilizado estructuras de repetición.
  - c) Se han reconocido las posibilidades de las sentencias de salto.
  - d) Se ha escrito código utilizando control de excepciones.
  - e) Se han creado programas ejecutables utilizando diferentes estructuras de control.
  - f) Se han probado y depurado los programas.
  - g) Se ha comentado y documentado el código.
- RA 4. Desarrolla programas organizados en clases analizando y aplicando los principios de la programación orientada a objetos.
  - a) Se ha reconocido la sintaxis, estructura y componentes típicos de una clase.
  - b) Se han definido clases.
  - c) Se han definido propiedades y métodos.
  - d) Se han creado constructores.
  - e) Se han desarrollado programas que instancien y utilicen objetos de las clases creadas anteriormente.
  - f) Se han utilizado mecanismos para controlar la visibilidad de las clases y de sus miembros.
  - h) Se han creado y utilizado métodos estáticos.

- RA 5. Realiza operaciones de entrada y salida de información, utilizando procedimientos específicos del lenguaje y librerías de clases.

  a) Se ha utilizado la consola para realizar operaciones de entrada y salida de información.

  b) Se han aplicado formatos en la visualización de la información.

  c) Se han reconocido las posibilidades de entrada / salida del lenguaje y las librerías asociadas.
- RA 6. Escribe programas que manipulen información seleccionando y utilizando tipos avanzados de datos.

  a) Se han escrito programas que utilicen arrays.

  g) Se han utilizado expresiones regulares en la búsqueda de patrones en cadenas de texto.

Rúbrica de evaluación

Cada apartado puntuable del proyecto se valorará con la siguiente rúbrica.

#	Criterio	Porcentaje
1	El programa/función implementado no cumple los requisitos, no soluciona de forma algorítmica el ejercicio o las soluciones obtenidas por el programa no son las esperadas, el código no compila o contiene errores.	0%
2	El programa/función implementado soluciona de forma algorítmica el ejercicio pero falla con datos de entrada no permitidos.	25%
3	El programa/función implementado tiene fallos inesperados en situaciones específicas o concretas, es decir, falla para un determinado caso o valor de entrada, pero en general el resultado obtenido es válido.	50%
4	El programa/función implementado cumple los requerimientos pero: • El código no es legible o no está bien estructurado.	75%
5	El programa/función implementado se ajusta perfectamente a la especificación: • Se validan los datos de entrada. • El resultado obtenido es válido para cualquier dato de entrada. • El código es modular y se emplean funciones/métodos adecuadamente. • El código es legible y usa comentarios relevantes y/o Javadoc.	100%