

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Boletín de ejercicios	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Boletín de Cadenas, excepciones y nuevos conceptos de POO

Nota: A partir de este tema habrá control de excepciones en los sitios necesarios para que el programa no falle.

1. Toma el ejercicio 4 (funciones de cadenas) del tema anterior y modifica las siguientes funciones:

- Retoma el método *subCadena* y modifícalo de forma que si se le pasa parámetros no válidos lanzará una excepción del tipo *IllegalArgumentException*.
- Coge el método *muestraCentrado* y modifícalo de forma que si el String parámetro tiene más de 80 caracteres lanzará la excepción definida por ti del tipo *BufQueLargaException* con el texto "Cadena demasiado larga".
- En el main pide datos al usuario para probar las dos funciones anteriores y captura las excepciones.

2. (Validar los apartados a y b por separado)

a) Se desea crear una colección de figuras geométricas. Para ello se propone el siguiente esquema de clases:

Pacackage interfaz:

Clase Librería con las funciones (robustas ante excepciones):

pedirEntero(): Pide un número entero al usuario, si es algo distinto de un entero repite la petición. Finalmente devuelve el número que introdujo el usuario.

PedirReal(): Similar a *pedirEntero()* pero para pedir números reales.

Interface InterfazUsuario: Con las funciones *pedirDatos()* y *mostrarDatos()*.

Package geometria:

Clase Punto:

- x,y: Públicas. Simplemente guarda dos reales que indican la posición de un punto en dos dimensiones. Sin get ni set.
- Dispone de dos constructores: Uno con parámetros que inicializa las dos coordenadas y otro sin parámetros que inicializa a (0,0) llamando al anterior.

Clase Figura: Clase que contiene una posición denominada *origen* (será un Punto), un nombre (String) y un constructor que inicializa las dos propiedades y otro sin parámetros que inicializa origen a (0,0) y el nombre a "" llamando al primero. Cumple el interface InterfazUsuario de forma que tendrá también un método de introducción de datos que pide al usuario el nombre y posición y otro que muestra ambas propiedades. Se requiere set para nombre de forma que siempre lo guarde en mayúsculas y sin los espacios extremos.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Boletín de ejercicios	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Clase Línea: Hereda de Figura y se le añade la propiedad puntoFinal. Sobreescribe el método pedirDatos y mostrarDatos llamando a los de la clase padre y completándolos. Tendrá un constructor con dos puntos como parámetro. Inicializa el origen con el primer punto y el nombre a "línea" llamando al padre y con el segundo punto inicializa puntoFinal.

b) (se valida aparte) también en geometría:

Clase Polígono: Hereda de Figura y contiene un array de puntos público. El origen no forma parte de la figura, sería su centro. El constructor tiene como parámetros el nombre, el origen y la cantidad de puntos con lo que inicializa las propiedades correspondientes y el tamaño del array (no es necesario meter datos de los puntos). Añade un constructor sin parámetros que inicialice un triángulo que tu decidas llamando al otro constructor y completando los puntos. Amplía pedirDatos y mostrarDatos mediante sobreescritura. No es necesario realizar ninguna comprobación sobre los puntos (da igual que estén repetidos, en línea...)

Clase Circunferencia: hereda de Figura y almacena el radio. Su constructor tiene como parámetros un punto y un radio. Si el radio es negativo lanzará una excepción. Llama al de la clase padre para inicializar el punto y luego inicializa el radio. Un segundo constructor sin parámetros inicializa a centro (0,0) y radio 1 llamando al primer constructor. Cumple el interface *InterfazUsuario*.

Programa principal para la jerarquía de objetos anterior: **Estará en un package distinto a las clases previas.** Crear una colección de objetos tipo Figura. Un menú dará opciones de insertar (siempre al final) una línea, un triángulo un cuadrado o una circunferencia. Mostrar los elementos de la colección (sólo el nombre) y mostrar los datos de un elemento de la colección (buscando por el índice). Recuerda el control de excepciones necesario.

3. Se desea simular una clase parecida a la String denominada Cadena pero teniendo en cuenta que sólo se pueden usar los métodos **charAt()** y **length** y con las siguientes características:

- Tiene una propiedad privada denominada cadena que es una colección (ArrayList) de caracteres. Tendrá sólo un set que admite un String como parámetro y colocará cada uno de los elementos del String en la colección. Si hay espacios en los extremos los elimina.


- Sobreescribe toString() para que devuelva la colección de caracteres como una cadena.

- Sobreescribe equals(Object) para que devuelva true si el objeto tipo Cadena que se le pasa como parámetro contiene los mismos caracteres que la de la clase. También admitirá un objeto tipo String o un vector de char como parámetro de forma que debes aplicar polimorfismo para que funcione.

Si se le pasa algo que no sea Cadena, String o char[] lanzará la excepción *IllegalArgumentException*

- función eliminar() se le pasa un carácter y elimina todas las veces que aparece dicho carácter. Además devuelve la cantidad de caracteres que ha eliminado.

(opcional) Implementa la interfaz Comparable<Cadena> de forma que indique orden alfabético. Echa un vistazo al apéndice III.

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Boletín de ejercicios	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

4. Retoma del primer tema de POO las clases Empleado y Directivo y rehazlas usando Herencia de la siguiente forma. Crea una clase abstracta denominada Persona que tenga los elementos comunes de Empleado y Directivo. Además dispondrá del método abstracto firmaMail. Este método lo implementarás luego en Empleado mostrando simplemente nombre y apellidos del mismo y en Directivo nombre apellidos y departamento rodeado de asteriscos.

Revisa ambas clases de forma que minimices la repetición de código con llamadas entre constructores.

Sobreescribe toString() de forma que devuelva nombre y apellidos. Piensa donde realizar esta función.

5. (Validar los apartados por separado) Realiza las siguientes jerarquías de clases/interfaces con las características básicas que se han de programar. Añade otros métodos y/o propiedades que consideres prácticos para tu programa y aplica los conocimientos de POO (constructores, set/get, polimorfismo, encapsulación, sobreescritura, sobrecarga, toString, Comparable, etc.) que resulten convenientes para gestionar todos los objetos de forma eficiente y segura.

Este ejercicio no busca exclusivamente cumplir el enunciado, si no que se realice un diseño orientado a objetos completo con clases versátiles y seguras: minimizar lo público, setters y getters, múltiples constructores... Por supuesto si lo consideras y/o te apetece puedes añadir más clases e interfaces.

a)

- Interface *TransportePersonas* que defina las funciones
 - int entraPasaje(int cantidad): cantidad son las personas que entran. Devuelve el total tras la entrada
 - void mostrarPasajeros()
- Clase abstracta *Vehiculo* con propiedades marca, modelo, y velocidadMaxima.
- Clase *Coche*: hereda de vehículo y añade las propiedades color (String), numeroPlazas(int), motor(Diesel, Gasolina, Híbrido o Eléctrico hazlo con un enumerado sencillo, mira el apéndice I) y una colección de strings (ABS, Climatizador, Xenon, ESP, GPS, Manos Libres, Antiniebla,...) denominada equipacion (si prefieres hazla también con enumerados).
- *Taxi*: Hereda de Coche, implementa la interfaz TransportePersonas. Dispone de una propiedad booleana libre (indica si el taxi está libre si está a true) y de otra plazasLibres. En el constructor inicializa plazasLibres a numeroPlazas-1 (se le resta el taxista). Respecto a la interfaz:
 - entraPasaje: Si no llegan las plazas no entra nadie, si llegan pues entran todos.
 - mostrarPasajeros: Solo muestra por pantalla la cantidad de pasajeros.
- *Moto*: hereda de Vehiculo y dispone de un enumerado indicando el tipo de moto: Turismo, Sport, Trail, Cross.

b) Barco: Con propiedades nombre (String), bandera (String que indica el país), manga y eslora reales.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Boletín de ejercicios	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

- Petrolero: Hereda de barco y añade la propiedad toneladasCarga también real.
- Remolcador: Hereda de barco y añade puerto de trabajo (String) y capacidad de tiro en toneladas (real).
- Trasatlántico: Hereda de Barco e implementa la interface TransportePersonas. Dispone de las propiedades capacidad(int que indica el número de pasajeros que caben), numeroPiscinas (entero), ruta (Colección de strings que indica los puertos por los que va) y pasajeros que lleva una colección del tipo Pasajero (clase con los campos Nombre (String) y Camarote (int) y una función denominada datosPasajero que pide ambos datos al usuario). Respecto a la interfaz:
 - entraPasaje: añade tantos pasajeros a la colección como se indica en el parámetro cantidad. Para pedir los datos de cada pasajero se llama a la función datosPasajero. Si la cantidad pasada como parámetro no cabe en el barco este se llena justo hasta su capacidad.
 - mostrarPasajeros: Muestra un listado con las personas del pasaje y el camarote en el que van.

c) En el programa principal se crea gestionarán tres colecciones: Una de medios de transportes de personas, otra de vehiculos y una tercera de barcos. Un menú permite como mínimo las siguientes opciones:

Insertar dato: Da a elegir entre coche, moto, taxi, petrolero, remolcador y trasatlántico. Pide los datos del objeto y los inserta al final de la colección correspondiente. Si entra en dos colecciones se mete en ambas (por ejemplo Taxi entra en vehiculo y transporte de personas)

Mostrar colección: Se elige la colección a mostrar y saca todos los datos por pantalla.


Por supuesto debes aplicar modularidad y no hacer todo en el *main*.

Opcionales:

6. Define un enumerado denominado Posicion y con valores NUMERO, PRINCIPIO y FIN. Crea una función a la que se le pasa una cadena, un carácter y un enumerado Posicion y devuelve la posición dónde está el carácter en primer lugar si el enumerado está a PRINCIPIO, la última posición del carácter si está a FIN y la cantidad de caracteres si está a NUMERO. Devuelve -1 en caso de no encontrar el carácter (sólo con PRINCIPIO y FIN). Realiza las pruebas JUnit que consideres necesarias.

7. Se desea realizar un programa para la gestión de pedidos para una empresa de comercialización de comidas rápidas. Se realizará mediante colecciones anidadas. La colección principal contendrá los pedidos de cada cliente y se gestionará como una cola clásica, es decir, se insertan nuevos pedidos al final de la colección y se extraen por el principio (Usa la clase Queue, veer Apéndice I tema anterior). Cada pedido de la cola **será un objeto** y contendrá los siguientes datos:

- Lista de productos: Será a su vez una **colección de objetos** con los siguientes campos:
 - Tipo base (enumerado): A elegir entre Pizza, Bocadillo, Refresco o Helado.
 - Lista de ingredientes: Colección que contiene un ingrediente adicional en cada posición.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Boletín de ejercicios	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Será una colección de String.

- Sabor (String): Sólo se especifica en el caso de refrescos y helados.
- Precio: Se calcula sumando al precio base el total de ingredientes por el precio por ingrediente en caso de ser bocadillo o pizza. Y solamente el precio del producto base en caso de refrescos o helados.
- Código de Pedido: Un número o referencia única. Puede ser tipo entero o cadena. Si es tipo entero se puede coger un número entre 1 y 1000 ya que se supone que nunca habrá más de 1000 pedidos en la cola. Crear una nueva excepción para controlar este rango.
- Tipo de Venta (Enumerado): Domicilio, Local, Recoger.
- Domicilio del Cliente: Se pide solo en el caso de que Tipo de Venta sea Domicilio (será un String).

El programa tendrá en principio un menú con tres opciones:

- Nuevo Pedido: Se pedirán todos los datos del pedido al cliente y el pedido pasará a la cola de pedidos. Antes de introducirlo en la cola se deberá mostrar todo el pedido en pantalla y preguntar si es correcto.
- Despachar Pedido: Simplemente sacará un pedido de la cola y lo mostrará en pantalla.
- Mostrar cola de pedidos: Mostrará solo el código y el tipo de venta por cada pedido.