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 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 aunque no se indique en el enunciado.

1. Modifica el ejercicio 1 del boletín 6a (excepciones) de forma que creas dos excepciones nuevas que vas a usar sustituyendo a IllegalArgumentException de la siguiente manera:

En el método **muestraCentrado** y si el String parámetro tiene más de 80 caracteres lanzará la **excepción definida por ti mediante herencia** de IllegalArgumentException del tipo **StringTooLongException** con el texto "Cadena demasiado larga" que se lo pasarás al constructor de la clase base mediante super.

En el método **subCadena**, si los parámetros son inválidos lanzará una excepción definida por ti mediante herencia de StringOutOfBoundsException pero con un constructor en el que le pasas un parámetro tipo String. Dicho String el constructor lo pasa, a su vez, al constructor de la clase base. Desde la función, cuando se lance la excepción, se le pasa un texto que indique el error de parámetros inválidos y además informa del valor de los parámetros y del tamaño de la cadena.

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 packages/clases:

Package 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.
- Cada vez que necesites pedir un dato entero o real en el resto del ejercicio usa estas funciones.

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

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

Package geometria:

Clase Punto:

- x,y: propiedades 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.

- Sobreescribe toString para que devuelva las coordenadas entre paréntesis separadas por punto y coma y un decimal (por ejemplo (3.0; 6.2))

Clase Figura:

- Clase que contiene como propiedades protegidas con set/get una posición denominada origen (será un Punto) y un nombre (String). Se requiere set para nombre de forma que siempre lo guarde en mayúsculas y sin los espacios extremos.

- 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.

Clase Linea:

- Hereda de Figura y se le añade la propiedad privada con set/get 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 "linea" llamando al padre y con el segundo punto inicializa puntoFinal.

- Un constructor sobrecargado sin parámetros que llama al primero con origen (0,0) y punto final (1,1).

b) (se valida aparte) también en package geometria:

Clase Poligono:

- Hereda de Figura y contiene un array de puntos público. El origen no forma parte de la figura, equivaldrá a su centro.

- Un 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. Los puntos del array los inicializas con valores aleatorios.

- Añade un constructor sin parámetros que inicialice un triángulo que tu decidas llamando al otro constructor y luego fijando los puntos (0,0), (1,1) y (1,0).

- 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 como propiedad con set/get. Si el radio es negativo lanzará una excepción IllegalArgumentException.

- Un constructor tiene como parámetros un punto y un radio. 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.

- Sobreescribe y amplía mostrarDatos y pedirDatos.

Package principal

Clase App:

Simplemente estará el main. En el debes crear una colección de objetos tipo Figura. Un menú dará opciones de:

- Insertar una línea, un triángulo un cuadrado o una circunferencia.
- Mostrar los elementos de la colección (sólo el nombre y, en el caso de circunferencias, también el radio).
- Mostrar los datos de un elemento de la colección (buscando por el índice).
- Borrar elementos de un tipo.
- Salir. Como siempre, solo acaba el programa al elegir esta opción.

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 concatenaciones y los métodos **charAt**() y **length,** y tendrá 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 que se le pasa como parámetro cumple:

    - Es de tipo Cadena y contiene los mismos caracteres y en las mismas posiciones que la colección de la instancia.
    - Es un objeto tipo String y contiene los mismos caracteres y en las mismas posiciones que la colección de la instancia.

- Es un vector de char y contiene los mismos caracteres y en las mismas posiciones que la colección de la instancia.

Si se le pasa un objeto null o un parámetro que no sea tipo Cadena, String o char[] lanzará la excepción IllegalArgumentException

- Método **eliminar(char):** 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. Mira el Apéndice III de los apuntes.

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** (no en Persona) en Empleado (devolviendo nombre y apellidos del mismo) y en Directivo (devolviendo 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.

Amplia con la comprobación del DNI en el set correspondiente de la siguiente forma:

- En el set se pasa un numero de dni y la letra. Puede ir sin separación o separado con un guión (123456789X o 123456789-X). Debe comprobar si la letra corresponde al DNI. El algoritmo es el siguiente:
  - Se coge el n.º y se calcula el resto de dividirlo entre 23.

- Este resultado da un número entre 0 y 22 que corresponde por posición a una letra de la siguiente **cadena** (ojo, no lo puedes meter en un vector):

  "**TRWAGMYFPDXBNJZSQVHLCKE**"

- Si el DNI no es un número válido de 9 dígitos (o no es un número) , o no hay correspondencia entre la letra y el DNI, se lanza una excepción creada por ti denominada **DNIExcepcion** que hereda de IllegalArgumentException.

5. Se desea realizar un gestor de cuerpos celestes. Para ello se pide generar el siguiente esquema de clases:

a) **Clase Astro:**

- Con propiedades privadas nombre (String) y radio (double).
- Al guardar el nombre lo hará en mayúsculas y eliminando posibles espacios en los extremos del nombre.
- Habrá una sobrecarga del get con un parámetro tipo carácter que devuelve el nombre con sus letras separadas por ese caracter. Por ejemplo si se llama getNombre('_') a un astro denominado "SOL" devuelve "S_O_L".
- Al guardar el radio se comprueba que sea positivo, si no fuera así lanza la excepción RadioNegativoException creada por ti.
- Sobreescribe equals de forma que se considera que dos astros son iguales si son iguales sus nombres y además son de la misma clase.
- Sobreescribe toString de forma que devuelva el nombre según getNombre y el radio con 2 decimales.

b) **Clase Planeta:**

- Hereda de Astro y dispone de una propiedad privada booleana denominada gaseoso con set y get.
- Otra propiedad pública que será una colección de Astros que serán los satélites.

- Tendrá dos constructores, uno que inicialice los miembros nombre, radio y gaseoso con parámetros y la colección de satélites vacía.

- El otro constructor sin parámetros que inicializa a "" nombre, 0 radio y false la propiedad gaseoso llamando al primer constructor.

c) En el **programa principal** (Mételo en otra clase y package distintos a las clases anteriores) se crea una **colección de Astros** y el siguiente menú:

- Añade Planeta: Pregunta si es gaseoso y pide el nombre y radio. También pregunta cantidad de lunas y se introducirán sus nombres y radios.

- Añade otro astro: Pide su nombre y radio.

- Mostrar datos: Muestra toda la colección detectando si es Astro o Planeta para mostrar todos sus datos. En el caso de Astro simplemente llama a toString() y lo muestra, en el caso de Planetas muestra todos los datos y en particular llamando a getNombre con parámetro '.'

- Elimina repetidos. Busca Astros/Planetas iguales (aprovecha el equals/indexOf/lastIndexOf) y elimina todas las apariciones menos la primera.

- Salir. Como siempre no sale de la aplicación hasta que se selecciona esta opción.

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. 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.

a) 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.

b) Tipo de Venta (Enumerado): Domicilio, Local, Recoger.

c) 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:

a) 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.

b) Despachar Pedido: Simplemente sacará un pedido de la cola y lo mostrará en pantalla.

c) Mostrar cola de pedidos: Mostrará solo el código y el tipo de venta por cada pedido.

El programa dispondrá de persistencia de datos, es decir, guardará los datos´en un archivo cuando se cierre y los cargará de nuevo al ejecutarse.

8. Continua o haz alguno de los opcionales planteados en temas previos pero aprovechando los recursos vistos en este tema.