BOLETÍN DE EJERCICIOS TEMA 2

PROFESOR: MARTÍN GARCÍA FIGUEIRA
martin@ciclosmontecastelo.com

BLOQUE I – POO

1. **Comprobando el código**

   ¿Qué sucedería si eliminamos la declaración local de la variable n en el siguiente código?

```
public partial class Form1 : Form
{
    private int n = 8;
    private void MiMétodo()
    {
        int n;
        n = 3;
    }
}
```

2. **Comprobando el código II**

   ¿Qué error encuentras en el siguiente código?

```
Public Form1()
{
    label1.Text = "38";
    InitializeComponent();
}
```

3. **Comprobando el código III**

   ¿Qué funcionalidad tiene el siguiente código?

```
int a;
a = label1.Width * label1.Height;
label1.Height = label1.Width;
```

4. Comprobando el código IV

¿Qué función tiene el siguiente código?

```
int número = 0;
while (número <= 5)
{
    textBox1.AppendText(Convert.ToString(número * número) + " ");
    número++;
}
```

5. Comprobando el código V

¿Qué aparece en pantalla con el siguiente código?

```
int n, m;
n = 10;
m = 5;
while ((n > 0) || (m > 0))
{
    n = n - 1;
    m = m - 1;
}
MessageBox.Show("n = " + Convert.ToString(n) + " m = " + Convert.ToString(m));
```

6. Creando métodos y propiedades

Imagina que queremos hacer una clase de un reproductor MP3; ¿qué métodos y propiedades podría tener? ¿Cuáles de estos métodos y propiedades podrían ser miembros?

7. Creando clases I

Entré en un concesionario y después de ver varios coches compré un Ford de 2010, con capacidad para 5 personas, de 100CV y de color azul:

- ¿Cómo sería el constructor Coche?
- ¿El constructor cuenta con parámetros?
- ¿Cómo instanciarías la clase Coche para representar el ejemplo?

8. <u>Creando clases II</u>

Amplía el objeto Globo de manera que tenga una variable que describa el color del globo.

```
public class Globo
{
    private int x = 50;
    private int y = 50;
    private int diámetro = 20;
}
```

9. <u>Creando clases III</u>

- Escribe un método que mueva el globo hacia arriba por una cantidad que se proporcionará como parámetro. Usa el nombre MoverArriba.
- Escribe un método que permita cambiar el color del globo.
- Escribe el método Mostrar, de manera que despliegue un globo de color.
- Escribe una propiedad para permitir que un usuario sólo tenga acceso get a la coordenada “y” de un globo.
- Escriba un método constructor para crear un nuevo globo, especificando únicamente su diámetro.
- Escribe una lista de operaciones que puedan realizarse con un objeto de la clase Globo, y da ejemplos de cómo usarlas.

10. <u>Creando clases IV</u>

- Crea una clase ‘Cuadrado’ con propiedades ‘Altura’ y ‘Ancho’ y un método ‘Area()’. Controla que no permita usar valores negativos.
- Crea un objeto de la clase Cuadrado y modifica sus propiedades. Visualiza el área del mismo.

- Sustituye el método por defecto 'ToString()', para que muestre información válida del objeto.

11. ¿Método o propiedad?

¿Cuáles de los siguientes elementos deberían ser métodos y cuáles propiedades al diseñar una clase llamada Cuenta para representar una cuenta bancaria?

- AbonarEnCuenta
- RetirarDeCuenta
- SaldoActual
- CalcularInterés
- Nombre

12. Creando un objeto II

Queremos crear un objeto que:

- Tenga propiedades 'Tipo' (Bus, Coche, Tren, etc), 'Fabricante', 'N_Ruedas', 'Pasajeros', 'consumo', etc.
- Tenga una variable pública y estática 'PrecioGasolina'
- Tenga un método estático PrecioPorKm()'.

Crea la clase que nos permita instanciarla para crear ese objeto.

13. Problema I: Pantalla de amplificador

Algunos amplificadores estereofónicos cuentan con un dispositivo visual que muestra el volumen de salida. Ese dispositivo aumenta y disminuye su nivel de acuerdo con el volumen en cualquier momento dado. De igual manera, ciertas pantallas tienen indicadores que muestran los valores máximo y mínimo alcanzados desde que el amplificador se encendió.

Escribe un programa que muestre en cuadros de texto los valores máximo y mínimo a los que se haya establecido una barra de seguimiento.

Escribe el código que tenga los valores y los compare en una clase. Esta clase debe tener un método llamado NuevoValor junto con las propiedades MenorValor y MayorValor.

14. Problema II: Cuenta bancaria

Escribe un programa que simule una cuenta bancaria. Un cuadro de texto debe permitirnos realizar depósitos (un número positivo) en la cuenta y hacer retiros (un número negativo) de la misma. El estado de la cuenta debe mostrarse de manera continua, y si entra en números rojos (saldo negativo) desplegar un mensaje apropiado.

Crea una clase llamada Cuenta para representar cuentas bancarias. Debe tener los métodos Depositar y Retirar, junto con una propiedad llamada SaldoActual.

15. Problema III: Guardar puntuación

Diseña y escribe una clase que sirva para guardar la puntuación para un juego de ordenador.

Debe mantener un solo entero: la puntuación.

Además, es necesario que proporcione métodos para inicializar la puntuación en cero, para incrementar la puntuación, para reducir la puntuación, y para devolver la puntuación.

Escribe instrucciones para crear y utilizar un solo objeto.

16. Problema IV: Dados

Diseña y escribe una clase que actúe como un dado que se pueda lanzar para obtener un valor del 1 al 6.

Primero escribe la clase de manera que siempre obtenga el valor 6.

Crea un programa para crear y utilizar un objeto dado. La pantalla debe mostrar un botón que al ser oprimido “lance” el dado y despliegue su valor.

Después modifica la clase dado, de manera que proporcione un valor un punto mayor al que tenía la última vez que se lanzó; por ejemplo, 4 si la última ocasión cayó en 3.

Luego transforma una vez más la clase, de manera que utilice el generador de números aleatorios de la biblioteca.

Algunos juegos, como el backgammon y el monopolio, requieren dos dados. Escribe instrucciones de C# para crear dos instancias del objeto dado, lanzarlos y mostrar los resultados.

17. Problema V: Generador de números aleatorios

Escribe tu propio generador de números aleatorios, creando

para ello una clase que utilice una fórmula para obtener el siguiente número pseudoaleatorio a partir del anterior.

Los programas de números aleatorios funcionan empezando con cierto valor de "semilla". A partir de ese momento, el número aleatorio actual se utiliza como base para obtener el siguiente. Esto se lleva a cabo realizando ciertos cálculos con el número actual para convertirlo en alguno otro (aparentemente aleatorio).

Una fórmula que podemos usar para los enteros es:

siguienteA = ((anteriorA * 25173) + 13849) % 65536;

Esta fórmula produce números en el rango de 0 a 65,535. Los números específicos derivados de la misma han demostrado su capacidad para producir buenos resultados de tipo aleatorio.

18. Problema VI: Números complejos

Escribe una clase llamada Complejo para representar números complejos

(junto con sus operaciones).

Los números complejos consisten de dos partes: una parte real (un double) y una imaginaria (un double).

El método constructor debe crear un nuevo número complejo mediante el uso de los valores double que se proporcionen como parámetros, de la siguiente forma:

Complejo c = new Complejo(1.0, 2.0);

Escribe las propiedades Real e Imaginaria para obtener las partes respectivas de un número complejo, mismas que deben utilizarse de la siguiente manera:
double x = c.Real;
Crea un método para sumar dos números complejos y devolver el resultado. La parte real es la suma de las dos partes reales; la parte imaginaria es la suma de las dos partes imaginarias.
Una invocación al método tendría esta forma:

Complex c = c1.Sum(c2);

Escribe un método para calcular el producto de dos números complejos.
Si un número tiene los componentes x1 e y1, y el segundo tiene los componentes x2 e y2:

- Parte real del producto = $x_1 \times x_2 + y_1 \times y_2$
- Parte imaginaria del producto = $x_1 \times y_2 + x_2 \times y_1$

19. Problema VII: Persona

Crea una clase llamada Persona que siga las siguientes condiciones:

- Sus atributos son: nombre, edad, DNI, sexo (H hombre, M mujer), peso y altura. No queremos que se accedan directamente a ellos. Piensa que modificador de acceso es el más adecuado, también su tipo. Si quieres añadir algún atributo puedes hacerlo.
- Por defecto, todos los atributos menos el DNI serán valores por defecto según su tipo (0 números, cadena vacía para String, etc.). Sexo sera hombre por defecto, usa una constante para ello.
- Se implantarán varios constructores:

- Un constructor por defecto.
- Un constructor con el nombre, edad y sexo, el resto por defecto.
- Un constructor con todos los atributos como parámetro.

- Los métodos que se implementaran son:
  - calcularIMC(): calculara si la persona está en su peso ideal (peso en kg/(altura^2 en m)), si esta fórmula devuelve un valor menor que 20, la función devuelve un -1, si devuelve un número entre 20 y 25 (incluidos), significa que está por debajo de su peso ideal la función devuelve un 0 y si devuelve un valor mayor que 25 significa que tiene sobrepeso, la función devuelve un 1. Te recomiendo que uses constantes para devolver estos valores.
  - esMayorDeEdad(): indica si es mayor de edad, devuelve un booleano.
  - comprobarSexo(char sexo): comprueba que el sexo introducido es correcto. Si no es correcto, sera H. No sera visible al exterior.
  - toString(): devuelve toda la información del objeto.
  - generaDNI(): genera un número aleatorio de 8 cifras, genera a partir de este su número su letra correspondiente. Este método será invocado cuando se construya el objeto. Puedes dividir el método para que te sea más fácil. No será visible al exterior.
  - Métodos set de cada parámetro, excepto de DNI.

Ahora, crea una clase ejecutable que haga lo siguiente:

- Pide por teclado el nombre, la edad, sexo, peso y altura.
- Crea 3 objetos de la clase anterior, el primer objeto obtendrá las anteriores variables pedidas por teclado, el segundo objeto obtendrá todos los anteriores menos el peso y la altura y el último por defecto, para este último utiliza los métodos set para darle a los atributos un valor.

- Para cada objeto, deberá comprobar si esta en su peso ideal, tiene sobrepeso o por debajo de su peso ideal con un mensaje.
- Indicar para cada objeto si es mayor de edad.
- Por último, mostrar la información de cada objeto.

20. Problema VIII: Password

Crea una clase llamada Password que siga las siguientes condiciones:

- Que tenga los atributos longitud y contraseña . Por defecto, la longitud será de 8.
- Los constructores serán los siguiente:
  - Un constructor por defecto.
  - Un constructor con la longitud que nosotros le pasemos. Generara una contraseña aleatoria con esa longitud.
- Los métodos que implementa serán:
  - esFuerte(): devuelve un booleano si es fuerte o no, para que sea fuerte debe tener mas de 2 mayúsculas, mas de 1 minúscula y mas de 5 números.
  - generarPassword(): genera la contraseña del objeto con la longitud que tenga.
  - Método get para contraseña y longitud.
  - Método set para longitud.

Ahora, crea una clase clase ejecutable:

- Crea un array de Passwords con el tamaño que tú le indiques por teclado.
- Crea un bucle que cree un objeto para cada posición del array.
- Indica también por teclado la longitud de los Passwords (antes de bucle).
- Crea otro array de booleanos donde se almacene si el password del array de Password es o no fuerte (usa el bucle anterior).

- Al final, muestra la contraseña y si es o no fuerte (usa el bucle anterior). Usa este simple formato:
  - contraseña1 valor_booleano1
  - contraseña2 valor_bololeano2
  - …

21. Visibilidad

Coloca dos botones sobre un formulario y cambia el código en el button_Click como se indica a continuación. Observa las regiones de código donde existen las variables tipo string.

```
public class Form1: Form
{
    string strClassString = "Clase String";
    private void button1_Click(object sender, EventArg e)
    {
        string strMetodoString = "Método String";
        MessageBox.Show(strClassString); // OK
        MessageBox.Show(strMetodoString); // OK
    }
    private void button2_Click(object sender, EventArg e)
    {
        MessageBox.Show(strClassString); // OK
        MessageBox.Show(strMetodoString); // Error
    }
}
```

Localiza las regiones de código donde puedes declarar string como public o private.

22. Declara una segunda ‘strClassString’ en la primera línea de ‘button2_Click’, y observa qué ocurre:

```
string strClassString = "String Nuevo";
```

BLOQUE II – Herencia, polimorfismo e interfaces

1. Creando una clase I

- Crea una clase 'Cuadrado' con propiedades 'Altura' y 'Ancho' y un método 'Area()'.
- Crea una nueva clase llamada 'Caja', heredada de 'Cuadrado' que disponga de una nueva propiedad 'Profundidad' y del método 'Volumen()', que sustituya al método 'Area()'.

2. Problema I: Electrodomésticos

Crearemos una superclase llamada Electrodoméstico con las siguientes características:

- Sus atributos son precio base, color, consumo energético (letras entre A y F) y peso. Indica que se podrán heredar.
- Por defecto, el color será blanco, el consumo energético será F, el precioBase es de 100 € y el peso de 5 kg. Usa constantes para ello.
- Los colores disponibles son blanco, negro, rojo, azul y gris. No importa si el nombre está en mayúsculas o en minúsculas.
- Los constructores que se implementarán serán:
  - Un constructor por defecto.
  - Un constructor con el precio y peso. El resto por defecto.
  - Un constructor con todos los atributos.
- Los métodos que implementará serán:
  - Métodos get de todos los atributos.
  - comprobarConsumoEnergetico(char letra): comprueba que la letra es correcta, sino es correcta usara la letra por defecto. Se invocará al crear el objeto y no sera visible.

- comprobarColor(String color): comprueba que el color es correcto, sino lo es usa el color por defecto. Se invocará al crear el objeto y no será visible
- precioFinal(): según el consumo energético, aumentara su precio, y según su tamaño, también. Esta es la lista de precios:

LETRA	PRECIO
A	100 €
B	80 €
C	60 €
D	50 €
E	30 €
F	10 €

TAMAÑO	PRECIO
Entre 0 y 19 kg	10 €
Entre 20 y 49 kg	50 €
Entre 50 y 79 kg	80 €
Mayor que 80 kg	100 €

Crearemos una subclase llamada Lavadora con las siguientes características:

- Su atributo es carga, además de los atributos heredados. Por defecto, la carga es de 5 kg. Usa una constante para ello.
- Los constructores que se implementarán serán:
  - Un constructor por defecto.
  - Un constructor con el precio y peso. El resto por defecto.
  - Un constructor con la carga y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.
- Los métodos que se implementarán serán:
  - Método get de carga.
  - precioFinal():, si tiene una carga mayor de 30 kg, aumentara el precio 50 €, sino es así no se incrementara el precio. Llama al método padre y añade el código necesario. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Crearemos una subclase llamada Television con las siguientes características:

- Sus atributos son resolución (en pulgadas) y sintonizador TDT (booleano), ademas de los atributos heredados. Por defecto, la resolución sera de 20 pulgadas y el sintonizador sera false.
- Los constructores que se implementaran serán:
    - Un constructor por defecto.
    - Un constructor con el precio y peso. El resto por defecto.
    - Un constructor con la resolución, sintonizador TDT y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.
- Los métodos que se implementara serán:
    - Método get de resolución y sintonizador TDT.
    - precioFinal(): si tiene una resolución mayor de 40 pulgadas, se incrementara el precio un 30% y si tiene un sintonizador TDT incorporado, aumentara 50 €. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Ahora crea una interfaz gráfica que permita:

- Crear Lavadoras y Televisiones.
- Los almacene en Lists y los muestre por pantalla.
- Saber el precio final de cada objeto seleccionado
- Ahora, recorre estas listas y ejecuta el método precioFinal().
- Deberás mostrar el precio de cada clase, es decir, el precio de todas las televisiones, por un lado, el de las lavadoras por otro y la suma de los Electrodomésticos. Recuerda el uso operador instanceof.
- Por ejemplo, si tenemos un Electrodoméstico con un precio final de 300, una lavadora de 200 y una televisión de 500, el resultado final será de

1000 (300+200+500) para electrodomésticos, 200 para lavadora y 500 para televisión.

3. Problema II: Series

Crearemos una clase llamada Serie con las siguientes características:

- Sus atributos son título, numero de temporadas, entregado, genero y creador. Por defecto, el numero de temporadas es de 3 temporadas y entregado false. El resto de atributos serán valores por defecto según el tipo del atributo.
- Los constructores que se implementarán serán:
  - Un constructor por defecto
  - Un constructor con el titulo y creador. El resto por defecto
  - Un constructor con todos los atributos, excepto de entregado.

- Los métodos que se implementara serán:
  - Métodos get de todos los atributos, excepto de entregado
  - Métodos set de todos los atributos, excepto de entregado.
  - Sobrescribe los métodos toString.

Crearemos una clase Videojuego con las siguientes características:

- Sus atributos son titulo, horas estimadas, entregado, genero y compañia. Por defecto, las horas estimadas serán de 10 horas y entregado false. El resto de atributos serán valores por defecto según el tipo del atributo.
- Los constructores que se implementaran serán:
  - Un constructor por defecto.
  - Un constructor con el titulo y horas estimadas. El resto por defecto.
  - Un constructor con todos los atributos, excepto de entregado.

- Los métodos que se implementara serán:
    - Métodos get de todos los atributos, excepto de entregado.
    - Métodos set de todos los atributos, excepto de entregado.
    - Sobrescribe los métodos toString.

Como vemos, en principio, las clases anteriores no son padre-hija, pero si tienen en común, por eso vamos a hacer una interfaz llamada Entregable con los siguientes métodos:

- entregar(): cambia el atributo prestado a true.
- devolver(): cambia el atributo prestado a false.
- isEntregado(): devuelve el estado del atributo prestado.
- Método compareTo (Object a), compara las horas estimadas en los videojuegos y en las series el número de temporadas. Como parámetro que tenga un objeto, no es necesario que implementes la interfaz Comparable. Recuerda el uso de los casting de objetos.

Implementa los anteriores métodos en las clases Videojuego y Serie.

Ahora crea una interfaz gráfica que permita:

- Crear Series y Videojuegos
- Guardarlos en listas y mostrarlos en pantalla
- Entrega algunos Videojuegos y Series con el método entregar().
- Cuenta cuántas Series y Videojuegos hay entregados. Al contarlos, devuélvelos.
- Por último, indica el Videojuego tiene más horas estimadas y la serie con mas temporadas. Muéstralos en pantalla con toda su información (usa el método toString()).

4. Problema III: Libros

Crearemos una clase Libro:

- Tiene que contener los siguientes atributos:
  - ISBN
  - Titulo
  - Autor
  - Número de páginas

- Crear sus respectivos métodos get y sets correspondientes para cada atributo.
- Crear el método toString() para mostrar la información relativa al libro con el siguiente formato:

"El libro con ISBN creado por el autor tiene páginas"

Ahora crea una interfaz gráfica que permita:

- Crear Libros
- Mostrar su información
- Por último, indicar cuál tiene más páginas.

5. Problema IV: Raíces

Vamos a realizar una clase llamada Raices, donde representaremos los valores de una ecuación de 2º grado.

- Tendremos los 3 coeficientes como atributos, llamémosles a, b y c.
- Hay que insertar estos 3 valores para construir el objeto.
- Las operaciones que se podrán hacer son las siguientes:
  - obtenerRaices(): imprime las 2 posibles soluciones
  - obtenerRaiz(): imprime única raíz, que será cuando solo tenga una solución posible.

- getDiscriminante(): devuelve el valor del discriminante (double), el discriminante tiene la siguiente formula, (b^2)-4*a*c
- tieneRaices(): devuelve un booleano indicando si tiene dos soluciones, para que esto ocurra, el discriminante debe ser mayor o igual que 0.
- tieneRaiz(): devuelve un booleano indicando si tiene una única solución, para que esto ocurra, el discriminante debe ser igual que 0.
- calcular(): mostrara por pantalla las posibles soluciones que tiene nuestra ecuación. En caso de no existir solución, mostrarlo también. Formula ecuación 2º grado: (-b±√((b^2)-(4*a*c)))/(2*a)

Ahora crea una interfaz gráfica que permita:

- Insertar los coeficientes
- Realizar los cálculos de las raíces

6. Problema V: Escuela

Queremos representar con programación orientada a objetos, un aula con estudiantes y un profesor.

- Tanto de los estudiantes como de los profesores necesitamos saber su nombre, edad y sexo.
- De los estudiantes, queremos saber también su calificación actual (entre 0 y 10) y del profesor que materia da.
- Las materias disponibles son matemáticas, filosofía y física.
- Los estudiantes tendrán un 50% de hacer novillos, por lo que si hacen novillos no van a clase, pero, aunque no vayan quedara registrado en el aula (como que cada uno tiene su sitio).

- El profesor tiene un 20% de no encontrarse disponible (reuniones, baja, etc.)
- Las dos operaciones anteriores deben llamarse igual en Estudiante y Profesor (polimorfismo).
- El aula debe tener un identificador numérico, el número máximo de estudiantes y para que esta destinada (matemáticas, filosofía o física). Piensa que más atributos necesita.
- Un aula para que se pueda dar clase necesita que el profesor esté disponible, que el profesor de la materia correspondiente en el aula correspondiente (un profesor de filosofía no puede dar en un aula de matemáticas) y que haya más del 50% de alumnos.
- El objetivo es crear un aula de alumnos y un profesor y determinar si puede darse clase, teniendo en cuenta las condiciones antes dichas.
- Si se puede dar clase mostrar cuantos alumnos y alumnas (por separado) están aprobados de momento (imaginad que les están entregando las notas).
- NOTA: Los datos pueden ser aleatorios (nombres, edad, calificaciones, etc.) siempre y cuando tengan sentido (edad no puede ser 80 en un estudiante o calificación ser 12).

Ahora crea una interfaz gráfica que permita:

- Crear alumnos, profesores y aulas
- Comprobar si es posible impartir clase

7. Problema VI: Cine

Nos piden hacer un programa orientado a objetos sobre un cine (solo de una sala) tiene un conjunto de asientos (8 filas por 9 columnas, por ejemplo).

- Del cine nos interesa conocer la película que se está reproduciendo y el precio de la entrada en el cine.
- De las películas nos interesa saber el título, duración, edad mínima y director.
- Del espectador, nos interesa saber su nombre, edad y el dinero que tiene.
- Los asientos son etiquetados por una letra (columna) y un número (fila), la fila 1 empieza al final de la matriz como se muestra en la tabla. También deberemos saber si está ocupado o no el asiento.

8 A 8 B 8 C 8 D 8 E 8 F 8 G 8 H 8 I

7 A 7 B 7 C 7 D 7 E 7 F 7 G 7 H 7 I

6 A 6 B 6 C 6 D 6 E 6 F 6 G 6 H 6 I

5 A 5 B 5 C 5 D 5 E 5 F 5 G 5 H 5 I

4 A 4 B 4 C 4 D 4 E 4 F 4 G 4 H 4 I

3 A 3 B 3 C 3 D 3 E 3 F 3 G 3 H 3 I

2 A 2 B 2 C 2 D 2 E 2 F 2 G 2 H 2 I

1 A 1 B 1 C 1 D 1 E 1 F 1 G 1 H 1 I

- Realizaremos una pequeña simulación, en el que generaremos muchos espectadores y los sentaremos aleatoriamente (no podemos donde ya este ocupado).
- En esta versión sentaremos a los espectadores de uno en uno.
- Solo se podrá sentar si tienen el suficiente dinero, hay espacio libre y tiene edad para ver la película, en caso de que el asiento este ocupado le buscamos uno libre.

- Los datos del espectador y la película pueden ser totalmente aleatorios.

  Ahora crea una interfaz gráfica que permita:

  - Sentar a la gente en el cine
  - Visualizar los datos de las películas que hay
  - Visualizar los datos de los espectadores

8. Problema VI: Barajas

Vamos a hacer una baraja de cartas españolas orientado a objetos.

- Una carta tiene un número entre 1 y 12 (el 8 y el 9 no los incluimos) y un palo (espadas, bastos, oros y copas)
- La baraja estará compuesta por un conjunto de cartas, 40 exactamente.
- Las operaciones que podrá realizar la baraja son:
  - barajar: cambia de posición todas las cartas aleatoriamente
  - siguienteCarta: devuelve la siguiente carta que está en la baraja, cuando no haya más o se haya llegado al final, se indica al usuario que no hay más cartas.
  - cartasDisponibles: indica el número de cartas que aún puede repartir
  - darCartas: dado un número de cartas que nos pidan, le devolveremos ese número de cartas (piensa que puedes devolver). En caso de que haya menos cartas que las pedidas, no devolveremos nada pero debemos indicárselo al usuario.
  - cartasMonton: mostramos aquellas cartas que ya han salido, si no ha salido ninguna indicárselo al usuario
  - mostrarBaraja: muestra todas las cartas hasta el final. Es decir, si se saca una carta y luego se llama al método, este no mostrara esa primera carta.

Ahora crea una interfaz gráfica que permita:

- Realizar cada una de las operaciones para una baraja

9. Problema V: Liga de fútbol

Estando en un grupo de amigos, se planea hacer una porra de la liga de fútbol.

A nosotros se nos ocurre hacer un programa en POO para simular como podría desarrollarse la porra.

Cada jugador de la porra, pone un 1 euro cada jornada y decide el resultado de los partidos acordados.

Si nadie acierta en una jornada los resultados, el bote se acumula.

En principio, deben acertar el resultado de dos partidos para llevarse el dinero del bote de la porra.

Todos empiezan con un dinero inicial que será decidido por el programador (ya sea como parámetro o como constante). Si el jugador no tiene dinero en una jornada no podrá jugar la porra.

Para esta versión, entre jugadores podrán repetir resultados repetidos, por lo que el jugador que primero diga ese resultado (tal como estén de orden) se llevara primero el bote.

Los resultados de la porra serán generados aleatoriamente, tanto los de jugador como los de los partidos (no es necesario nombre, solo resultados).

Al final del programa, se deberá mostrar el dinero que tienen los jugadores y el número de veces que han ganado.

Para este ejercicio, se recomienda usar interfaces para las constantes y métodos que sean necesarios.

Ahora crea una interfaz gráfica que permita:

- Jugar a la porra

10. **<u>Problema VI: Ruleta rusa</u>**

    Como muchos sabéis, se trata de un número de jugadores que con un revolver con una sola bala en el tambor se dispara en la cabeza.

    Las clases a hacer son:

    - **Revolver**:
        - Atributos:
            - posición actual (posición del tambor donde se dispara, puede que esté la bala o no)
            - posición bala (la posición del tambor donde se encuentra la bala)
            - Estas dos posiciones se generarán aleatoriamente.
        - Funciones:
            - disparar(): devuelve true si la bala coincide con la posición actual
            - siguienteBala(): cambia a la siguiente posición del tambor
            - toString(): muestra información del revolver (posición actual y donde está la bala)
    - **Jugador**:
        - Atributos
            - id (representa el número del jugador, empieza en 1)
            - nombre (Empezara con Jugador más su ID, “Jugador 1” por ejemplo)
            - vivo (indica si está vivo o no el jugador)
        - Funciones:
            - disparar(Revolver r): el jugador se apunta y se dispara, si la bala se dispara, el jugador muere.
    - **Juego**:
        - Atributos:

        - Jugadores (conjunto de Jugadores)
        - Revolver
    - Funciones:
        - finJuego(): cuando un jugador muere, devuelve true
        - ronda(): cada jugador se apunta y se dispara, se informara del estado de la partida (El jugador se dispara, no ha muerto en esa ronda, etc.)

El número de jugadores será decidido por el usuario, pero debe ser entre 1 y 6. Si no está en este rango, por defecto será 6.

En cada turno uno de los jugadores, dispara el revólver, si este tiene la bala el jugador muere y el juego termina.

Ahora crea una interfaz gráfica que permita:

- Crear jugadores
- Jugar a la ruleta

11. Problema VII: Gestión de empleados

Nos piden hacer un programa que gestione empleados.

- Los empleados se definen por tener:
    - Nombre
    - Edad
    - Salario
- También tendremos una constante llamada PLUS, que tendrá un valor de 300€
- Tenemos dos tipos de empleados: repartidor y comercial.
- El comercial, aparte de los atributos anteriores, tiene uno más llamado comisión (double)

- El repartidor, aparte de los atributos de empleado, tiene otro llamado zona (String).
- Crea sus constructores, getters and setters y toString (piensa como aprovechar la herencia).
- No se podrán crear objetos del tipo Empleado (la clase padre) pero si de sus hijas.
- Las clases tendrán un método llamado plus, que según en cada clase tendrá una implementación distinta. Este plus básicamente aumenta el salario del empleado:
    - En comercial, si tiene más de 30 años y cobra una comisión de más de 200 euros, se le aplicara el plus.
    - En repartidor, si tiene menos de 25 y reparte en la "zona 3", este recibirá el plus.
- Puedes hacer que devuelva un booleano o que no devuelva nada, lo dejo a tu elección.

Ahora crea una interfaz gráfica que permita:

- Crear varios empleados
- Aplicarles el plus

12. Problema VIII: Gestión de productos

Nos piden hacer que gestionemos una serie de productos.

- Los productos tienen los siguientes atributos:
    - Nombre
    - Precio

- Tenemos dos tipos de productos:
    - Perecedero: tiene un atributo llamado días a caducar
    - No perecedero: tiene un atributo llamado tipo

- Crea sus constructores, getters, setters y toString.
- Tendremos una función llamada calcular, que según cada clase hará una cosa u otra, a esta función le pasaremos un numero siendo la cantidad de productos:
    - En Producto, simplemente seria multiplicar el precio por la cantidad de productos pasados.
    - En Perecedero, aparte de lo que hace producto, el precio se reducirá según los días a caducar:
        - Si le queda 1 día para caducar, se reducirá 4 veces el precio final.
        - Si le quedan 2 días para caducar, se reducirá 3 veces el precio final.
        - Si le quedan 3 días para caducar, se reducirá a la mitad de su precio final.
    - En NoPerecedero, hace lo mismo que en producto

Ahora crea una interfaz gráfica que permita:

- Crear los productos
- Vender los productos

13. Problema IX: Gestión del almacén

- En un almacén se guardan un conjunto de bebidas.
- Estos productos son bebidas como agua mineral y bebidas azucaradas (coca-cola, fanta, etc).
- De los productos nos interesa saber su identificador (cada uno tiene uno distinto), cantidad de litros, precio y marca:
    - Si es agua mineral nos interesa saber también el origen (manantial tal sitio o donde sea).

- Si es una bebida azucarada queremos saber el porcentaje que tiene de azúcar y si tiene o no alguna promoción (si la tiene tendrá un descuento del 10% en el precio).

- En el almacén iremos almacenado estas bebidas por estanterías (que son las columnas de la matriz).
- Las operaciones del almacén son las siguientes:
  - Calcular precio de todas las bebidas: calcula el precio total de todos los productos del almacén.
  - Calcular el precio total de una marca de bebida: dada una marca, calcular el precio total de esas bebidas.
  - Calcular el precio total de una estantería: dada una estantería (columna) calcular el precio total de esas bebidas.
  - Agregar producto: agrega un producto en la primera posición libre, si el identificador esta repetido en alguno de las bebidas, no se agregará esa bebida.
  - Eliminar un producto: dado un ID, eliminar el producto del almacén.
  - Mostrar información: mostramos para cada bebida toda su información.

Ahora crea una interfaz gráfica que permita:

- Añadir productos
- Calcular precios
- Mostrar información

14. Problema X: Agenda telefónica

Nos piden realizar una agenda telefónica de contactos.

- Un contacto está definido por un nombre y un teléfono (No es necesario de validar). Un contacto es igual a otro cuando sus nombres son iguales.

- Una agenda de contactos está formada por un conjunto de contactos (Piensa en que tipo puede ser)
- Se podrá crear de dos formas, indicándoles nosotros el tamaño o con un tamaño por defecto (10)
- Los métodos de la agenda serán los siguientes:
    - anhadirContacto(Contacto c): Añade un contacto a la agenda, sino se pueden meter más a la agenda se indicara por pantalla. No se pueden meter contactos que existan, es decir, no podemos duplicar nombres, aunque tengan distinto teléfono.
    - existeContacto(Conctacto c): indica si el contacto pasado existe o no.
    - listarContactos(): Lista toda la agenda
    - buscaContacto(String nombre): busca un contacto por su nombre y muestra su teléfono.
    - eliminarContacto(Contacto c): elimina el contacto de la agenda, indica si se ha eliminado o no por pantalla
    - agendaLlena(): indica si la agenda está llena.
    - huecosLibres(): indica cuantos contactos más podemos meter.

Ahora crea una interfaz gráfica que permita:

- Implementa todas las operaciones en una interfaz

15. Problema XI: Mejora de la baraja

Vamos a hacer unas mejoras a la clase Baraja del problema VI.

- Lo primero que haremos es que nuestra clase Baraja será la clase padre y será abstracta.
- Le añadiremos el número de cartas en total y el número de cartas por palo.

- El método crearBaraja() será abstracto.
- La clase Carta tendrá un atributo genérico que será el palo de nuestra versión anterior.
- Creamos dos Enum:
    - PalosBarEspañola:
        - OROS
        - COPAS
        - ESPADAS
        - BASTOS
    - PalosBarFrancesa:
        - DIAMANTES
        - PICAS
        - CORAZONES
        - TREBOLES
- Creamos dos clases hijas:
    - BarajaEspañola: tendrá un atributo boolean para indicar si queremos jugar con las cartas 8 y 9 (total 48 cartas) o no (total 40 cartas).
    - BarajaFrancesa: no tendrá atributos, el total de cartas es 52 y el número de cartas por palo es de 13. Tendrá dos métodos llamados:
        - cartaRoja(Carta<PalosBarFrancesa> c): si el palo es de corazones y diamantes.
        - cartaNegra(Carta<PalosBarFrancesa> c): si el palo es de tréboles y picas.
- De la carta modificaremos el método toString()
- Si el palo es de tipo PalosBarFrancesa:
    - La carta número 11 será Jota
    - La carta numero 12 será Reina
    - La carta numero 13 será Rey

        - La carta numero 1 será As
    - Si el palo es de tipo PalosBarEspañola:
        - La carta numero 10 será Sota
        - La carta numero 12 será Caballo
        - La carta numero 13 será Rey
        - La carta numero 1 será As

Ahora crea una interfaz gráfica que permita:

- Realizar cada una de las operaciones para una baraja

16. Problema XII: Gestión de una academia

Una academia nos pide hacer un programa para hacer un pequeño test a sus alumnos.

Estas preguntas, para facilitar la inclusión, estarán escritas en un txt (incluido en el ejercicio).

- Una opción se compone de:
    - El texto de la opción (digamos la respuesta)
    - Es correcto o no
- Una pregunta consta de:
    - Pregunta (tendrá delante dos puntos y coma ;P;)
    - Opciones de la pregunta (entre 2 y 4)
    - Opción correcta (tendrá delante dos puntos y coma ;R;)
    - Puntos
- La pregunta no será válida en los siguientes casos:
    - Las opciones no están entre 2 y 4.
    - La opción correcta esta entre el número de opciones y es un número.
    - Los puntos es un número entero.
- Sus métodos son:

  - mostrarPregunta(): muestra la pregunta con sus opciones.
  - comprobarRespuesta(int respuestaUsuario): comprueba la respuesta del usuario si es correcta o no.
  - Getter de los atributos.

- Un test está formado por un conjunto preguntas y los puntos acumulados. Piensa que debemos saber por cual pregunta vamos.
- Sus métodos son:
  - cargarPreguntas(String fichero): carga todas las preguntas del fichero
  - siguientePregunta(): devuelve la siguiente pregunta
  - reiniciarTest(): nos permite reiniciar el test.
  - realizarTest(): empieza el test y empieza a formular las preguntas
- El fichero de preguntas tiene el siguiente formato:

;P;Pregunta 1

Opción 1 pregunta 1

Opción 2 pregunta 1

Opción 3 pregunta 1

Opción 4 pregunta 1

;R;Numero opción correcta

Puntos pregunta 1

;P;Pregunta 2

Opción 1 pregunta 2

Opción 2 pregunta 2

…

…

Ahora crea una interfaz gráfica que permita:

- Realizar test
- Comprobar resultados