Nombre:	**Nota**
Curso:	

Tiempo estimado	Tipo de actividad
3 hora	**Evaluación**

Unión Europea
Fondo Europeo de Desarrollo Regional
Una manera de hacer Europa

GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

IES Mare Nostrum

Normas

- **El código entregado tiene que compilar sin problemas**
- **De cada ejercicio tendrá su propia nota, se especificará la nota mínima. Si no se alcanza esa nota mínima, la nota del examen será 4 como máximo.**
- **De cada ejercicio se tiene que proporcionar el main que has usado para ver el correcto funcionamiento. Si este main es escaso y se deja porciones de algoritmos sin cubrir, bajará la nota. Si es demasiado escasa, la nota máxima del examen será de 4.**
- **La estructura de cada fichero será el siguiente (si no, no se corregirá el ejercicio)**
  - **enums? delegados?**
  - **Una clase**
    - **Clases nuevas dentro de la clase**
    - **Atributos**
    - **Properties**
    - **Constructores**
    - **Destructores**
    - **Getters y Setters**
    - **Otros métodos**
- **Si cometes alguno de los errores comentados en clase (que son mayoritariamente 3) que no se pueden permitir, la nota máxima del examen será de 4.**
- **No puedes hacer trampas**

(1 punto) (Nota mínima 7) Implementa la siguiente función: Implementa una función que, dado un número “n”, devuelve una lista con los n primeros números de la serie de fibonacci.

(2 puntos) (Nota mínima 7) Implementa la siguiente función.
Una función llamada Merge, que reciba dos arrays de enteros que están ordenados y devuelva un array ordenado con el contenido de los dos arrays de entrada. Prohibido usar algoritmos de ordenación, ni de C#, ni tuyos. El algoritmo se explicará en la pizarra.

(2 puntos) (Nota mínima 4) Implementa la siguiente clase: Es una clase llamada User que tiene que gestionar una lista de claves (una clave es un string). Cuando pruebes esta clase, en el main pon el código necesario para capturar las excepciones.

- **Atributos**
  - Tendrás que definirlos tú mismo/a.
- **Properties**
  - Name (El getter devolverá el nombre en mayúsculas)
  - Code: Se obtiene sumando todos los caracteres del nombre más la edad multiplicado por 33
  - Age (sólo get)
  - KeyCount
- **Constructores**
  - Un único constructor que tenga el número máximo de claves que puede almacenar y la edad. Si la edad es menor o igual que cero, hay que lanzar una excepción. Si el número de claves máximo es menor que 1 hay que lanzar una excepción
- **Métodos**
  - Los que necesites
  - AddKey(string) Toma la decisión que creas conveniente aquí
  - ContainsKey(string)
  - RemoveKeys(delegado)
  - ClearKeys

(2 puntos) (nota mínima 3) Implementa la siguiente clase, pon los atributos, constructores y métodos que veas necesarios. La clase que hay que implementar es un pool de objetos que implementa una interfaz, es decir hay que hacer una interfaz y la clase. Entre el código a desarrollar que tienes que hacer: ToArray, Clone y Count. Es posible que tengas que usar “where T : class”. El constructor de la clase será **privado, si no, no se corregirá el ejercicio**, y en la interfaz habrá una función de clase (estática) llamada CreatePool que creará un pool.

(3 puntos) (nota mínima 3) Se desea realizar una aplicación que es un simulador del juego de la oca con alguna restricción especial.

- **Prohibido usar funciones que añadan elementos a un array, tanto tuyas como del sistema.**
- **Prohibido hacer castings.**
- El juego (la clase Juego) constará de una lista de casillas jugadores y un **array de casillas**. Tendrá al menos los siguientes métodos
    - Añadir jugador: le paso un jugador
    - Borrar jugadores: Dejará el juego sin jugadores
    - Crear tablero. Creará el tablero con el que jugar. Esta función ocupará como máximo 20 líneas.
        - *Pista: Cada vez que crees un tablero, empieza creando un array con fichas normales*
    - Ordenar jugadores (privada): Antes de empezar la partida se debe determinar el orden de los jugadores. Cada jugador lanza los dados, y empiezan en orden de ese resultado.
        - En caso de empate en el dado de salida, desempatarán tirando dados de nuevo hasta que uno de ellos consiga más que el otro.
        - Cada jugador comienza en la casilla 1
    - Simular: devolverá el jugador que ha ganado. Antes de empezar se debe determinar el orden de salida.
- Las casillas heredarán de una clase abstracta, y sólo podrán tener un único atributo que es su número (empezando por 1) que se le pasará por el constructor. Las casillas son:
    - 63: Casilla ganadora. Si un jugador cae aquí, gana la partida. Si se pasa, satará a la casilla 55.
    - 6, 12, 18, 24, 30, 36, 42, 48, 54 y 60: De oca en oca. Si un jugador cae aquí, saltará a la siguiente casilla de este tipo y vuelve a tirar.
    - 8 y 14: De puente en puente. Si un jugador cae en una de estas casillas, pasará a la otra casilla y vuelve a tirar.
    - 13, 26, 39 y 52: Casillas de castigo. Si un jugador cae aquí esperará 1, 2, 3 o 4 turnos sin jugar.
    - 27 y 53: De dado a dado: Si un jugador cae en la 26, pasará a la 53 y le vuelve a tocar
    - 58: La muerte. Si un jugador cae aquí va a la casilla de salida.
- Los jugadores. Hay que implementar varios tipos de jugadores que heredarán todos de la clase Jugador.
    - Un jugador tiene:
        - Nombre que no puede cambiar.
        - Otros aspectos que consideres
    - Tipos de jugadores
        - Jugador normal. Cada vez que le toque lanzará un dado de 6 caras que será el número de casillas a avanzar.
        - Jugador rápido. Cada vez que le toque, lanzará entre 1 y 3 dados de 6 caras y se quedará con el más alto. El número de dados que lanzará se le pasará por el constructor.
        - Jugador tramposo. Lanzará un dado de 8 caras para avanzar.