1.- Ejercicio del reloj.

Realiza una clase reloj que sea capaz de almacenar la hora de un reloj (hora, minuto y segundo) y el modo en el que se mostrará la hora (versión 12 24 horas). Realiza los métodos habituales y alguno especial que permita recargar pila. Sobrecarga el método *toString* para mostrar la hora.

2.- Clase fracción

Crea una clase Racional que permita trabajar con números racionales (fracciones).
Incluye los siguientes métodos: *constructores* (por defecto y parametrizados), *toString*, *getters* y *setters*. Además los siguientes métodos:
leer(); → que la pide por teclado,
c=sumar(a,b); → que suma las fracciones a y b devolviendo la fracción **simplificada** c.
c=multiplicar(a,b); → que multiplica las fracciones a y b devolviendo la fracción simplificada c.
a = simplificar(b); → método que simplifica la fracción b devolviéndola en la a.

3.- El coche.

Realiza una clase coche que tenga los atributos necesarios que permitan reflejar su *marca*, *modelo*, *color* y *matrícula*. También debe almacenar información acerca de sus características de movimiento: *motor encendido* o *apagado*, *marchaActual*, *velocidadActual*, *subirMarcha*, *bajarMarcha* y aquellos que creas conveniente para manipular su información dinámica.
Crea los métodos habituales que creas conveniente y, además, los métodos necesarios que nos permitan simular lo siguiente:

a) El coche parte de una situación de reposo.
b) Arranca.
c) Acelera e irá subiendo marchas hasta llegar a una velocidad que se ha pedido por teclado al usuario.
d) Se mantiene esa velocidad un tiempo que se ha pedido al usuario por teclado.
e) Se va desacelerando y bajando marchas hasta que el coche se pare.
f) Punto muerto y paramos el motor.

Para hacer la simulación tendremos en cuenta:

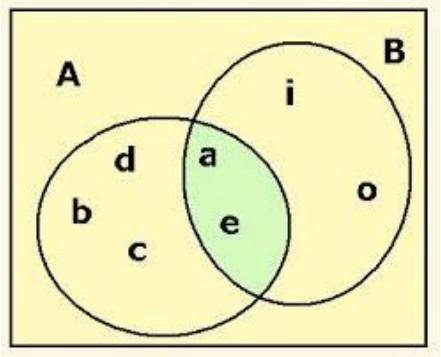
- Rangos de marcha: 1ª 0 – 30; 2ª 30 – 50; 3ª 50 – 70; 4ª 70 – 100; 5ª 100 →
- El 10% de las veces se nos cruza un gato. Para evitar pillarlo pegamos un frenazo que para el coche y cala el motor.

4.- Clase ordenador.

Crea la clase que almacén la información técnica de un ordenador.
Crea también la clase que permita almacenar la información de todos los ordenadores almacenados en un aula, así como el nombre del aula y el curso que se imparte en la misma.
Dejo a tu elección los atributos y métodos necesarios. El programa principal debe permitir gestionar dicha aula.

5.- Manejo de conjuntos.

Realizaremos una clase conjunto que permita crear conjuntos matemáticos. Estos conjuntos deben contener letras en un número indefinido de ellas y sin importar que se repitan. Crea los métodos habituales y los necesarios para añadir y borrar elementos.
Después se deben implementar las siguientes operaciones con los conjuntos:
C=Intersección(A,B) → Devuelve el conjunto formado por los elementos comunes en A y B.
C=Unión(A,B) → Devuelve el conjunto formado por todos los elementos de A y B.
N=Cardinalidad(A) → Devuelve la cantidad de elementos del conjunto A.



6.- Serpiente.

Diseñaremos un programa orientado a objetos que nos permita simular la vida de una serpiente. Nuestra serpiente será un poco especial y tendrá las siguientes características:

- Es una serpiente compuesta por anillas de un color diferente cada una. Estos colores podrán alternar entre (r ,v,a). Por ejemplo: rrvva → sería una serpiente de 5 anillas.
- Cuando nace tiene un color asignado al azar.
- Se simulará su vida hasta que esté muerta (en nuestro caso se quede sin cuerpo) de forma que cada año (segundo):
    - Si es joven (menos de 10 años) hará dos cosas:
        - Crecerá el 80% de las veces: añadiendo una nueva anilla a su cuerpo de un color asignado aleatoriamente.
        - Mudará la piel el 20% de las veces. En este caso se cambia el cuerpo de toda la serpiente respetando su número de anillas.
    - Si ya va siendo mayorcilla (más de 10 años):
        - Decrecerá el 90% de las veces: quitando una anilla a su cuerpo (la última) y respetando el resto.
        - Mudará la piel el 10% de las veces.
- Aleatoriamente, el 10% de las veces, durante el año de esa vida puede sufrir el ataque de una mangosta que se la zampa y por lo tanto la serpiente muere y se para la simulación.

Cuando esté realizado, generaliza el ejercicio para un nido de serpientes que puede albergar, como máximo, hasta 20 serpientes.

En este caso la simulación se ajusta a lo siguiente:

- Se simulará la vida del nido durante 5 minutos.
- Cada cinco segundos nacen (si pueden) entre 1 y 3 serpientes, ocupando su lugar en el nido.
- Cada segundo (año de la vida de una serpiente) pasará la vida para cada serpiente del nido según se especificó en el apartado anterior (menos el ataque de la mangosta).
- Cada 10 segundos una mangosta aparece por el nido zampándose entre 0 y 4 serpientes el 20% de las veces.
- Se debe mostrar el estado del nido cada segundo y describir lo que va ocurriendo.

7.- La mosca orientada a objetos.

Vamos a realizar la mosca que ya hicimos con matrices en el tema anterior pero orientado a objetos.

Una vez que tengamos resuelto el problema vamos a ampliar el juego añadiendo a la mosca un *tipo* (corriente, verde, negra, coj…) y *vidas*. Las vidas de la mosca indican la cantidad de golpes que tiene que recibir para matarla. Cuando golpeamos una mosca y le quedan vidas, le restamos una vida y revolotea.

8.- Buscaminas orientado a objetos.

Realizaremos el ejercicio del buscaminas pero orientado a objetos. En el programa principal se desarrolla el juego con los métodos proporcionados por la clase correspondiente.

9.- Cuenta bancaria.

Gestionaremos una cuenta bancaria. Dicha cuenta constará de un número de cuenta, un saldo y tres titulares (como máximo, al menos uno). Para cada titular tenemos que almacenar su DNI, nombre, apellidos y teléfono.

Crea los métodos necesarios que permitan realizar las operaciones habituales con la pasta: ingresar y retirar. Y añadir/borrar titulares a la misma. Cuando se crea una cuenta debe tener al menos un titular.

El programa principal debe tener menús que permitan gestionar todos los aspectos de la cuenta.

10.- Conquistadores de Catán.

Vamos a implementar un juego conocido como **Conquistadores de Catán.** Nuestro juego será mucho más sencillo que el original y sólo permitirá a dos jugadores: un **humano** y el **ordenador**.

El **funcionamiento** es el siguiente:

El juego consiste en un **mapa de 3x4 casillas**, en cada **casilla** habrá un **recurso** (*trigo, madera y carbón*); un jugador que será el **dueño** de esa casilla y un **valor** numérico entre 1 y 6 (valores de un *dado*).

Al principio se **inicializará** el tablero colocando en cada casilla un recurso (de forma aleatoria) y asignándole un valor (también de forma aleatoria entre 1 y 6).
Al comienzo los dos jugadores tienen sus **almacenes** de madera, carbón y trigo a *cero*.
Después le preguntamos al usuario que casilla quiere ocupar y la convertimos en propiedad del mismo, luego el ordenador elige una casilla vacía al azar y la ocupa; posteriormente le volvemos a preguntar al usuario, luego el ordenador… y así hasta que estén las 12 casillas ocupadas (tengan propietario).
Una vez completado el tablero, **comienza el juego**:
Tiran los jugadores por turnos. El valor del dado se comparará con todos los valores de las casillas del tablero y con aquellos que coincidan se incrementará la cantidad de recursos que esas casillas indiquen a los propietarios de dichas casillas.
Ganará el primero que consiga llegar a 20 en todos los recursos.
Se valora:

- **La elección de clases y la forma en que interactúan para implementar la solución.**
- **Claridad y ajuste de las clases a lo pedido en el enunciado.**
- **Se sigan los principios de abstracción y encapsulación de los objetos.**

Ejemplo:

Valores aleatorios iniciales:

Trigo 4	Carbón 1	Trigo 5	Carbón 5
Carbón 5	Trigo 2	Carbón 3	Trigo 1
Madera 3	Madera 6	Carbón 2	Madera 4

Elección del humano:

Trigo 4	Carbón 1	Trigo 5	Carbón 5
Carbón 5	Trigo 2	Carbón Humano 3	Trigo 1
Madera 3	Madera 6	Carbón 2	Madera 4

Elección del ordenador:

Trigo 4	Carbón 1	Trigo 5	Carbón 5
Carbón 5	Trigo 2	Carbón Humano 3	Trigo 1
Madera 3	Madera Ordenador 6	Carbón 2	Madera 4

Y así sucesivamente hasta que estén todas repartidas:

Trigo Humano 4	Carbón Humano 1	Trigo Ordenador 5	Carbón Humano 5
Carbón Ordenador 5	Trigo Ordenador 2	Carbón Humano 3	Trigo Humano 1
Madera Ordenador 3	Madera Ordenador 6	Carbón Ordenador 2	Madera Humano 4

Juego:

Ha salido el 5.

El humano: 1 ítem de carbón.

La máquina: 1 ítem de Carbón, 1 ítem de Trigo.

Ha salido el 6.

El humano: 0 ítems.

La máquina: 1 ítem de Madera.

11º Age of Empires.

Trabajamos en una empresa de programación de juegos. Nos encargan programar una nueva versión del ***Age of Empires***. Más concretamente el funcionamiento de una **mina de recursos.**
Nuestro juego consta de unos **aldeanos** que pertenecen a una **civilización** (Españoles, Ingleses, Bizantinos, etc...) y están **gobernados por un rey** (Alejandro, Isabel, Constantino, etc...); además cada aldeano consta de un **indicador de salud**.

La **mina** es explotada por los aldeanos y pueden ser de ORO o de PIEDRA. **No existe límite en el número de aldeanos que pueden trabajar en la mina. Tampoco tienen que ser todos de la misma civilización.**
Las minas por defecto son de "ORO" y tienen una cantera de 500 *ítems*. Debemos definir también un constructor que parametrice todo, es decir que desde la construcción se pueda cambiar el tipo de mina y los *ítems* que tiene.

Vamos a tener para **nuestro juego,** dos tipos de aldeanos: Españoles, gobernados por Isabel I y con una salud inicial de 200; y Bizantinos, gobernados por Constantino II y con una salud inicial de 250.

El sistema se simula **durante 1 minuto**, de forma que:

- Cada **segundo** todos los aldeanos extraen un ítem de la mina y lo suman al almacén de su civilización.
- Cada **2 segundos** se añade a la mina un español (al 40%) o un bizantino (al 20%); el resto de las veces no se añade a nadie.
- Cada **5 segundos** sufrimos el ataque de un cura bizantino. Estos curas tienen la capacidad de convertir a otros aldeanos a su bando. En dicho ataque sólo se convierte a un aldeano. Este aldeano convertido cambiará de bando y trabajará para los bizantinos desde ese momento.

Crea los métodos que creas conveniente para simular todo esto y guardar toda esta información. Así como los constructores más útiles.