TAREA1. FACTORIA DE COCHES.

Construir una fábrica de coches. Para ello se deben codificar las siguientes clases.

1. Clase Rueda
    a. double radio
    b. 3 constantes estáticas tipo int:
        i. SECO = 0
        ii. HUMEDO = 1
        iii. NEVADO = 2
    c. int tipo: Se le asignará un tipo estático de los anteriores. Por defecto, SECO.
    d. Rueda(double radio, String tipo)
    e. Rueda (): Inicializa una rueda con el siguiente estado (radio = 1, tipo = SECO).
    f. Métodos get/set y toString (toString debe convertir el tipo en texto según la constante asignada actualmente a tipo).

Prueba:

```
Rueda ruedaHumedo = new Rueda(2, Rueda.HUMEDO);
Rueda ruedaDefault = new Rueda();

System.out.println(ruedaHumedo);
System.out.println(ruedaDefault);
```

```
Rueda [radio=2.0, tipo=HUMEDO]
Rueda [radio=1.0, tipo=SECO]
```

2. Clase Chasis.
    a. double peso
    b. 4 constantes estáticas tipo String:
        i. MATERIAL1 = “ALUMINIO”
        ii. MATERIAL2 = “ACERO”
        iii. MATERIAL3 = “HIERRO”
        iv. MATERIAL4 = “FIBRA”
    c. String material: Se le asignará un tipo estático de los anteriores. Por defecto, MATERIAL1.
    d. Chasis (double peso, String material)
    e. Chasis (): Inicializa un chasis con los siguientes valores por defecto (peso = 1000, material = MATERIAL1)
    f. Métodos get/set y toString

Prueba:

```
Chasis chasisCustom = new Chasis(2500, Chasis.MATERIAL2);
Chasis chasisDefault = new Chasis();

System.out.println(chasisCustom);
System.out.println(chasisDefault);
```

```
Chasis [peso=2500.0, material=ACERO]
Chasis [peso=1000.0, material=ALUMINIO]
```

3. Clase Coche
    a. 4 constantes estáticas tipo int:
        i. ROJO = 0
        ii. AZUL = 1
        iii. AMARILLO = 2
        iv. BLANCO = 3
    b. Rueda [] conjuntoRuedas: Arrays de 4 objetos Rueda
    c. Chasis chasis.
    d. int color: Se le asignará un tipo estático de los anteriores. Por defecto, BLANCO.
    e. **Coche (Rueda rueda, Chasis chasis, int color)**: Donde una copia de rueda se repetirá x4 para cada una de las ruedas del coche, inicializándose todas iguales.
    f. **toString()** . Imprime un String con la información del coche y sus componentes.
    g. **setColor(int color)** : modifica el color del coche.
    h. **setRueda(Rueda rueda, int pos)** : modifica una rueda en una posición determinada.
    i. **setChasis(Chasis chasis):** Modifica el chasis actual.
    j. **Getters** que consideres oportunos.

Prueba:

```
Coche coche1 = new Coche(ruedaHumedo, chasisCustom, Coche.AZUL);
System.out.println(coche1);
```

```
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO],
    Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO]],
    chasis=Chasis [peso=2500.0, material=ACERO], color=1]
```

4. Clase Fabrica
    a. String nombreFabrica
    b. double defaultRadio
    c. int defaultTipo
    d. double defaultPeso
    e. String defaultMaterial
    f. Int capacidadFabrica: Número de coches que puede albergar una fábrica.
    g. Coche[] conjuntoCochesFabricados: Conjunto total de coches fabricados
    h. **Constructor** indicando todos los parámetros representativos de la fábrica (nombre, radio, tipo, peso, material y capacidad)
    i. Métodos **get/set** para los parámetros por defecto de la fábrica.
    j. Método público **iniciarFabricacion(int numeroCoches) : boolean**. Se inicia la fabricación de coches en caso de que se quiera fabricar más coches de los que actualmente puede albergar dará un error antes de iniciar su fabricación cancelando el proceso. Para la fabricación se utilizará el método privado "fabricarCoche()" en caso de fallo se mostrará un mensaje por pantalla y se intentará nuevamente. Cada coche nuevo debe ocupar un espacio en el array de "cochesFabricados".
    k. **Coche fabricarCoche():** Método privado que fabrica un coche. Devolviendo un objeto de la clase Coche con los atributos por defecto de la fábrica y el color del coche se escoge de forma aleatoria entre los disponibles (num. Entre 0 y 3 ambos incluídos).
    l. Método público **retirarCoche(int numeroCoches) : boolean**.

Comprobación:

- Primero es necesario comprobar si hay suficientes coches, sino lo hay se devolverá simplemente un false y no se hará nada.

Actuación en caso de poder realizar la retirada:

- Utilizar el método privado "sacarCoche" tantas veces como el número de coches a retirar y finalmente devolverá true.
  Mediante este método se retira siempre los coches por orden de fabricación el que está en primera posición del conjunto de cochesFabricados por cada coche a retirar – llamando a sacarCoche(0).

m. Método público **retirarCoche(int numeroCoches, int color) : boolean.**
Comprobaciones:

- Si hay suficientes coches, sino lo hay se devolverá simplemente un false y no se hará nada.
- Entre los coches disponibles hay suficientes coches del color indicado, sino lo hay se devolverá simplemente un false y no se hará nada.

Actuación en caso de poder realizar la retirada:

- Utilizar el método privado "sacarCoche" tantas veces como el número de coches a retirar y finalmente devolverá true.
  Mediante este método se retira siempre los coches por orden de fabricación el que está en las primeras posiciones y que tenga el color de coche deseado del conjunto de cochesFabricados por cada coche a retirar – llamando a sacarCoche(posicion). Siendo posición la posición del primer coche detectado del color deseado.

n. Método privado **sacarCoche(int posición)** : Retira el objeto Coche del conjunto de cochesFabricados situado en la posición indicada y actualiza el conjunto desplazando las posiciones de los coches en fabrica una posición menos desde la posición retirada.

TIP: Cada vez que retiremos un coche recoloca la colección de coches para que no queden huecos vacíos llevando los coches. Es decir, si tenemos [coche1, coche2, coche3, null] si quitamos el coche1 arrastramos coche2 a la posición de coche1 y coche3 a la posición de coche2. Resultado: [coche2, coche3, null,null].

o. **toString().** Imprime todos los coches fabricados hasta ese momento y disponibles en la fábrica y sus características.

Prueba:

```
Fabrica fabricaPepe = new Fabrica("PEPE", 2, Rueda.HUMEDO, 2500, Chasis.MATERIAL2, 10);
fabricaPepe.iniciarFabricacion(3);
System.out.println(fabricaPepe);

Fabrica fabricaJuan = new Fabrica("JUAN", 2, Rueda.NEVADO, 2000, Chasis.MATERIAL4, 5);
fabricaJuan.iniciarFabricacion(2);
System.out.println(fabricaJuan);
```

```
########################################
Fabrica: PEPE
Numero actual de fabricados: 3/10
******************************
Coche1 color AZUL:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO]], chasis=Chasis [peso=2500.0, material=ACERO], color=AZUL]:
******************************
Coche2 color ROJO:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO]], chasis=Chasis [peso=2500.0, material=ACERO], color=ROJO]:
******************************
Coche3 color BLANCO:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO]], chasis=Chasis [peso=2500.0, material=ACERO], color=BLANCO]:

########################################
Fabrica: JUAN
Numero actual de fabricados: 2/5
******************************
Coche1 color BLANCO:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO]], chasis=Chasis [peso=2000.0, material=FIBRA], color=BLANCO]:
******************************
Coche2 color ROJO:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO]], chasis=Chasis [peso=2000.0, material=FIBRA], color=ROJO]:
```

Crea una clase auxiliar con el método main para testear las clases. Se debe instanciar dos fábricas con 100 y 50 de capacidad máxima de coches y habrá que testear tanto el proceso de iniciar nuevas fabricaciones como la retirada de los coches ya sea cogiendo coches independientemente del color como precisando el color deseado.

```
fabricaPepe.retirarCoche(10); // Retirar 10 cuyo color sea de aleatorio.
fabricaPepe.retirarCoche(10, Coche.AZUL); // Retirar 10 coches azules.
```

ENTREGA:

Adjuntar antes de la fecha de finalización un fichero comprimido con los siguientes ficheros en su interior:

1. Crear diagrama UML de clases. Se puede facilitar el diagrama en formato .dia (software DIA) o directamente una imagen con suficiente calidad para identificar cada una de las partes.
2. Codificar cada una de las clases y testear salida. Proporcionando los ficheros .java correspondiente a cada una de las clases codificadas.
3. Codificar main y testear cada una de los métodos de fabricación y retirada. Proporcionar el/los fichero/s .java donde se han realizado las pruebas.

EVALUACIÓN:

- Codificación de las relaciones entre las clases.
- Codificación de los métodos solicitados
- Uso de las estructuras correcta de forma optima y sin problemas en la ejecución y depuración.
- Entrevista: Una vez finalizado el plazo de entrega y entre las entregas el docente concretará una entrevista con el alumno.