Generics

Se pide resolver cada uno de los ejercicios utilizando diagramas de clases UML y el lenguaje de programación Java.

[J.01] - Torneo.

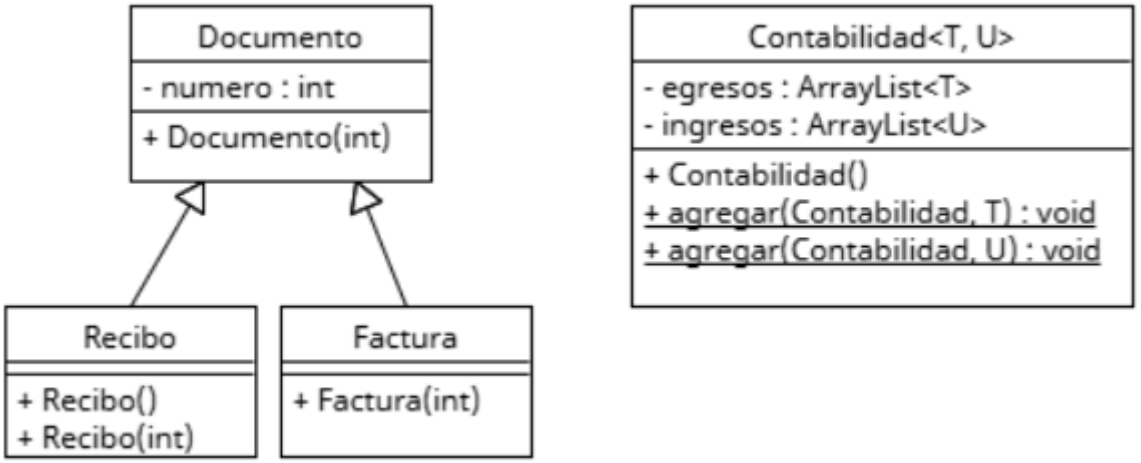
Crear un proyecto de biblioteca de clases con las clases Torneo (con un tipo genérico), Equipo, EquipoFutbol y EquipoBasquet.

1. Restringir el tipo genérico para que deba ser del tipo Equipo o sus derivados.
    a. Tendrá un atributo equipos de tipo ArrayList<T> y otro nombre de tipo String.
    b. Sobrescribir el ***equals*** para que controle si un equipo ya está inscrito al torneo.
    c. El método ***agregar***, para agregar un equipo a la lista, siempre y cuando este no se encuentre ya en el torneo.
    d. El método ***mostrar*** retornará los datos del torneo y de los equipos participantes.
    e. El método privado ***calcularPartido*** recibirá dos elementos del tipo **T**, que deberán ser del tipo Equipo o sus derivados, y calculará utilizando la clase Random un resultado del partido. Retornará el resultado como un string con el siguiente formato donde EQUIPOX es el nombre del equipo y RESULTADOX la cantidad de goles/puntos:
    [EQUIPO1][RESULTADO1] – [RESULTADO2][EQUIPO2]
    f. El método de instancia ***jugarPartido*** tomará dos equipos de la lista al azar y calculará el resultado del partido a través del método ***calcularPartido***.
2. Generar la clase Equipo abstracta.
    a. Agregar un atributo nombre de tipo string y otro fechaCreacion de tipo LocalDateTime.
    b. Dos equipos serán iguales si comparten el mismo nombre y fecha de creación.
    c. El método getFicha retornará el nombre del equipo y su fecha de creación con el siguiente formato:
    [EQUIPO] fundado el [FECHA]
3. Generar la clase EquipoFutbol que herede de Equipo.
4. Generar la clase EquipoBasquet que herede de Equipo.
5. Crear un proyecto de tipo consola.
    a. Generar dos torneos, uno de Futbol y otro de Basquet.
    b. Crear 3 equipos de cada tipo.
    c. Agregar los equipos en tantos torneos como se pueda.
    d. Llamar al método mostrar de Torneo e imprimir su retorno por pantalla.
6. Llamar al menos 3 veces a la propiedad jugarPartido de cada torneo e imprimir su respuesta por pantalla.

Una vez diseñado el diagrama de clases UML, crear, implementar y probar la jerarquía de clases en un proyecto de consola.

[J.02] - Contabilidad

Crear un proyecto de biblioteca de clases y agregar las clases del siguiente diagrama:



- Crear en Contabilidad un constructor que no reciba parámetros e inicialice las listas.
- El constructor sin parámetros de Recibo asignará 0 como número de documento.
- Tanto el tipo genérico T como el U deberán ser del tipo Documento o uno de sus derivados.
- El método estático agregar recibe un objeto de tipo Contabilidad y otro de tipo T y agrega un elemento a la lista egresos.
- El método estático agregar recibe un objeto de tipo Contabilidad y otro de tipo U agrega un elemento a la lista ingresos.

Crear un proyecto de consola y generar el código necesario para probar dichas clases.

[J.03] - Depósitos

1.- Crear un proyecto *Class library* (*Entidades.Genericas*)

Agregar al proyecto las clases detalladas a continuación.

Auto:

- Atributos privados:
    - color (cadena de caracteres)
    - marca (cadena de caracteres)
- Constructor
    - recibe dos parámetros de tipo cadena de caracteres.

- Getters
    - agregar un getter por cada atributo.
- Polimorfismo:
    - equals. Retorna ***true***, si el objeto a comparar es del tipo auto y tienen la misma marca y color.
    - toString. Retorna una cadena conteniendo la información del auto (marca y color).

Cocina:

- Atributos privados:
    - codigo (entero)
    - esIndustrial (booleano)
    - precio (flotante)
- Constructor
    - recibe tres parámetros, para inicializar cada uno de sus atributos.
- Getters
    - agregar un getter por cada atributo.
- Polimorfismo:
    - equals. Retorna ***true***, si el objeto a comparar es del tipo cocina y tienen el mismo código.
    - toString. Retorna una cadena conteniendo la información de la cocina (código, precio y si es industrial o no).

Deposito<T>: Es una clase diseñada para poder almacenar (en una lista genérica de tipo T) una cierta cantidad de objetos T. Dicha clase tiene la funcionalidad de agregar, remover y listar objetos T.

- Atributos privados:
    - capacidadMaxima(entero)
    - lista(lista genérica de T)
- Constructor
    - recibe un parámetro de tipo entero.

- Métodos
    - getIndice(T). Privado y de instancia. Retorna el valor del índice en el cual se encuentra el objeto T pasado como parámetro. Se debe recorrer la lista genérica y retornar el índice de la primera ocurrencia, **-1** si no se encuentra en la lista.
    - agregar(T). Público y de instancia. Retorna true, si pudo agregar el objeto T al depósito de T, false, caso contrario. Para poder agregar un objeto T a la lista genérica hay que tener en cuenta que la capacidad máxima del depósito no puede ser superada y que el objeto no se encuentre en la lista.
    - remover(T). Público y de instancia. Retorna true, si pudo remover el objeto T del depósito de T, false, caso contrario. Para poder remover un objeto T a la lista genérica, el objeto debe estar en la lista. Remover por índice.
- Polimorfismo:
    - toString. Retorna una cadena conteniendo la información del depósito T (capacidad y todo el detalle de los objetos T que contiene).

2. Diseñar el diagrama de clases UML, crear, implementar y probar la jerarquía de clases en un proyecto de consola (TestEntidadesGenericas) colocando las siguientes líneas de código:

```
public static void main(String[] args) {

    Cocina c1 = new Cocina(111, 12300, false);
    Cocina c2 = new Cocina(112, 15000, true);
    Cocina c3 = new Cocina(113, 5600, false);

    Auto a1 = new Auto("Rojo", "Ferrar");
    Auto a2 = new Auto("Amarillo" ,"Porche");
    Auto a3 = new Auto("Negro", "BMW");
    Auto a4 = new Auto("Verde", "Ford");

    Deposito<Cocina> dc = new Deposito<Cocina>(2);
    Deposito<Auto> da = new Deposito<Auto>(3);

    dc.agregar(c1);
    dc.agregar(c2);
    if (!(dc.agregar(c3)))
    {
        System.out.println("No se pudo agregar el item!!!");
    }
```

```
        if ((da.agregar(a1)))
        {
            System.out.println("Se ha agregado el item!!!");
        }

        da.agregar(a2);
        da.agregar(a3);
        if (!da.agregar(a4))
        {
            System.out.println("No se pudo agregar el item!!!");
        }

        System.out.println(dc);
        System.out.println(da);

        dc.remover(c2);
        if (!(dc.remover(c2)))
        {
           System.out.println("No se pudo remover el item!!!");
        }
        da.remover(a2);
        if (!(da.remover(a4)))
        {
            System.out.println("No se pudo remover el item!!!");
        }

        System.out.println(dc);
        System.out.println(da);
}
```