Arrays y colecciones

Arrays

[F.01] - Números locos

Crear una aplicación de consola que cargue 20 números enteros (positivos y negativos) distintos de cero de forma aleatoria utilizando la clase Random.
1. Mostrar el vector tal como fue ingresado.
2. Luego mostrar los positivos ordenados en forma decreciente.
3. Por último, mostrar los negativos ordenados en forma creciente.

[F.02] - Vienen con sistema de auto-navegación

De una empresa de transporte de cargas se quiere guardar el nombre de los conductores y los kilómetros que conducen cada día de la semana.

Para guardar esta información, la empresa de transporte tendrá un array de conductores. De cada conductor se tendrá la siguiente información:

- Nombre del conductor.
- Kilómetros recorridos cada día de la semana. Por ejemplo:
    - Día 1: 200
    - Día 2: 599
    - Día 3: 899
    - Día 4: 0 (tiene franco)
    - Día 5: 256
    - Día 6: 0
    - Día 7: 0

Desarrollar los diagramas UML y luego codificar en Java. Toda la lógica de negocio deberá estar encapsulada en clases y dentro de un proyecto de tipo biblioteca de clases. Crear un proyecto de consola y en el método main cargar tres conductores a la empresa de transportes y mostrar:

- El conductor que hizo más km en esa semana.
- El conductor que hizo más km el día 3.
- El conductor que hizo más km el día 5.

[F.03] - Pateta y témperas para el pintor

Hay que modelar la clase Tempera que posee tres atributos privados, constructor y métodos sobrecargados.
Atributos:

- color (de tipo Color, que es un enumerado que posee cómo enumeraciones: *Rojo*, *Blanco*, *Azul*, *Verde* y *Negro*)
- marca (cadena de caracteres)
- cantidad (entero)

Constructor

- recibe tres parámetros.

Método (de instancia)

- mostrar(): String. Método privado que retorna en formato de cadena los valores de cada atributo del objeto.
- getCantidad(): int, retorna el valor del atributo cantidad.

Métodos (de clase)

- mostrar(Tempera): String. Método público que recibe un parámetro de tipo Tempera y retorna en formato de cadena los valores de cada atributo del objeto.
- sonIguales(Tempera, Tempera): Boolean. Recibe dos parámetros de tipo Tempera y retorna ***true***, si el valor del color y de la marca son iguales en ambos objetos.
- sonDistintos(Tempera, Tempera): Boolean. Recibe dos parámetros de tipo Tempera y retorna ***false***, si el valor del color y de la marca son iguales en ambos objetos.
- add(Tempera, double): Tempera, método estático que suma la cantidad que recibe en el segundo parámetro al objeto de tipo Tempera que recibe cómo primer parámetro, retornado dicha instancia.

La siguiente clase que hay que modelar es la clase Paleta, la cual posee dos atributos privados, constructor y métodos sobrecargados.
Atributos:

- colores (array de tipo Tempera)
- cantidadMaximaColores (valor de tipo entero)

Constructor

- público y que posea un parámetro que indica la cantidad máxima de colores de la paleta.

Método (de instancia)

- mostrar(): String. Método público que retorna en formato de cadena los valores de cada atributo del objeto, incluyendo el detalle de cada tempera.
- obtenerIndice(): int. Método privado que retorna el primer índice disponible en el array de tipo Tempera, si no hay lugares disponibles, retorna -1.
- obtenerIndice(Tempera): int. Método privado que retorna el índice dónde se encuentra la témpera que recibe como parámetro. Si la témpera no está en la paleta, retorna -1.

Métodos (de clase)

- sonIguales(Paleta, Tempera): Boolean. Recibe dos parámetros, uno de tipo Paleta y otro de tipo Tempera. Retorna ***true***, si el segundo parámetro se encuentra en el atributo colores del primer parámetro.
- sonDistintos(Paleta, Tempera): Boolean. Recibe dos parámetros, uno de tipo Paleta y otro de tipo Tempera. Retorna ***false***, si el segundo parámetro se encuentra en el atributo colores del primer parámetro.
- add(Paleta, Tempera): Paleta. Si la témpera (segundo parámetro) se encuentra en la paleta (primer parámetro), se incrementa el valor del atributo cantidad en una unidad.
  Si la témpera no se encuentra en la paleta, se agrega al primer lugar disponible de la misma.
- add(Paleta, Paleta): Paleta. Retorna una paleta con la combinación de las paletas que recibe como parámetros. Si hay témperas iguales, se deberán sumar sus cantidades (no repetir colores).
- remove(Paleta, Tempera): Paleta. Si la témpera está en la paleta, se decrementa la cantidad de la misma. Si la cantidad resultante es menor o igual a cero, se elimina la témpera de la paleta (null).

Desarrollar los diagramas UML y luego codificar en Java. Toda la lógica de negocio deberá estar encapsulada en clases y dentro de un proyecto de tipo biblioteca de clases. Crear un proyecto de consola y en el método main, probar el correcto funcionamiento de los miembros de las clases modeladas.

Versión colecciones:

1. Reemplazar, en la clase Paleta, el array de tipo Tempera por un ArrayList<Tempera>.
2. Modificar la clase Tempera, quitar el atributo cantidad. En la clase Paleta, reemplazar la colección anterior por un HashMap<Tempera, double>.

Colecciones

[F.04] - Números locos 2

Crear una aplicación de consola que cargue en un ArrayList, Stack y Queue 20 números enteros (positivos y negativos) distintos de cero de forma aleatoria utilizando la clase Random.
1. Mostrar la colección tal como fue cargada.
2. Luego mostrar los positivos ordenados en forma decreciente.
3. Por último, mostrar los negativos ordenados en forma creciente.

[F.05] - La veterinaria

El dueño de una veterinaria nos contrató para que desarrollemos una aplicación donde pueda consultar la lista de clientes y sus mascotas.

A los clientes les interesa conocer el domicilio, el nombre y apellido y un teléfono. A un cliente se le pueden asociar una o más mascotas.

De las mascotas se necesita conocer su especie, su nombre, su fecha de nacimiento y su historial de vacunación. Los primeros tres datos son obligatorios para dar de alta una mascota, mientras que el último arrancará vacío y se podrá ir agregando vacunas.

De las vacunas sólo interesa conocer el nombre.

- Modelar en UML, luego codificar en Java, en una aplicación de tipo librería de clases.
- En un proyecto de tipo consola, crear:
  - Un cliente con un perro sin vacunar.
  - Un cliente con un gato con la vacuna "Triple Felina".
  - Un cliente con un gato sin vacunas y un perro con la vacuna contra la rabia.
- Mostrar por consola todos los datos de los clientes instanciados y sus mascotas.

[F.06] - Estadística deportiva

Modelar en UML las clases Jugador y Equipo y luego crear un proyecto de biblioteca de clases.

Clase Jugador
Atributos:

- dni (entero)
- nombre (cadena de caracteres)
- partidosJugados (entero)
- promedioGoles (flotante)
- totalGoles (entero)

Métodos:

- Jugador(), constructor sin parámetros y privado.
- Jugador(), constructor público con dos parámetros (dni y nombre)
- Jugador(), constructo público con cuatro parámetros (dni, nombre, total de goles, total de partidos).
- getPromedioGoles(), sin parámetros y retorna el promedio de goles del jugador.
- mostrarDatos(), sin parámetros y retorna una cadena de caracteres con todos los datos y estadísticas del jugador.
- sonIguales(), método estático que recibe dos parámetros de tipo Jugador y retorna un booleano que será ***true***, si ambos jugadores tienen el mismo DNI.
- sonDistintos(), método estático que recibe dos parámetros de tipo Jugador y retorna un booleano que será ***false***, si ambos jugadores tienen el mismo DNI.

Clase Equipo
Atributos:

- cantidadDeJugadores (entero)
- nombre (cadena de caracteres)
- jugadores (lista de jugadores)

Métodos:

- Equipo(), constructor privado y sin parámetros.
- Equipo(), constructor público que recibe la cantidad de jugadores y el nombre del equipo.
- add(), método estático que recibe dos parámetros, el primero de tipo Equipo y el segundo de tipo Jugador, retornando una instancia de Equipo. Este método agrega jugadores a la lista siempre y cuando no exista aún en el equipo y la cantidad de jugadores no supere el límite establecido por el atributo *cantidadDeJugadores*.

Nota: *La lista de jugadores se inicializará sólo en el constructor privado de Equipo.*

Crear un proyecto de consola y generar las invocaciones necesarias en el método main para probar el código.