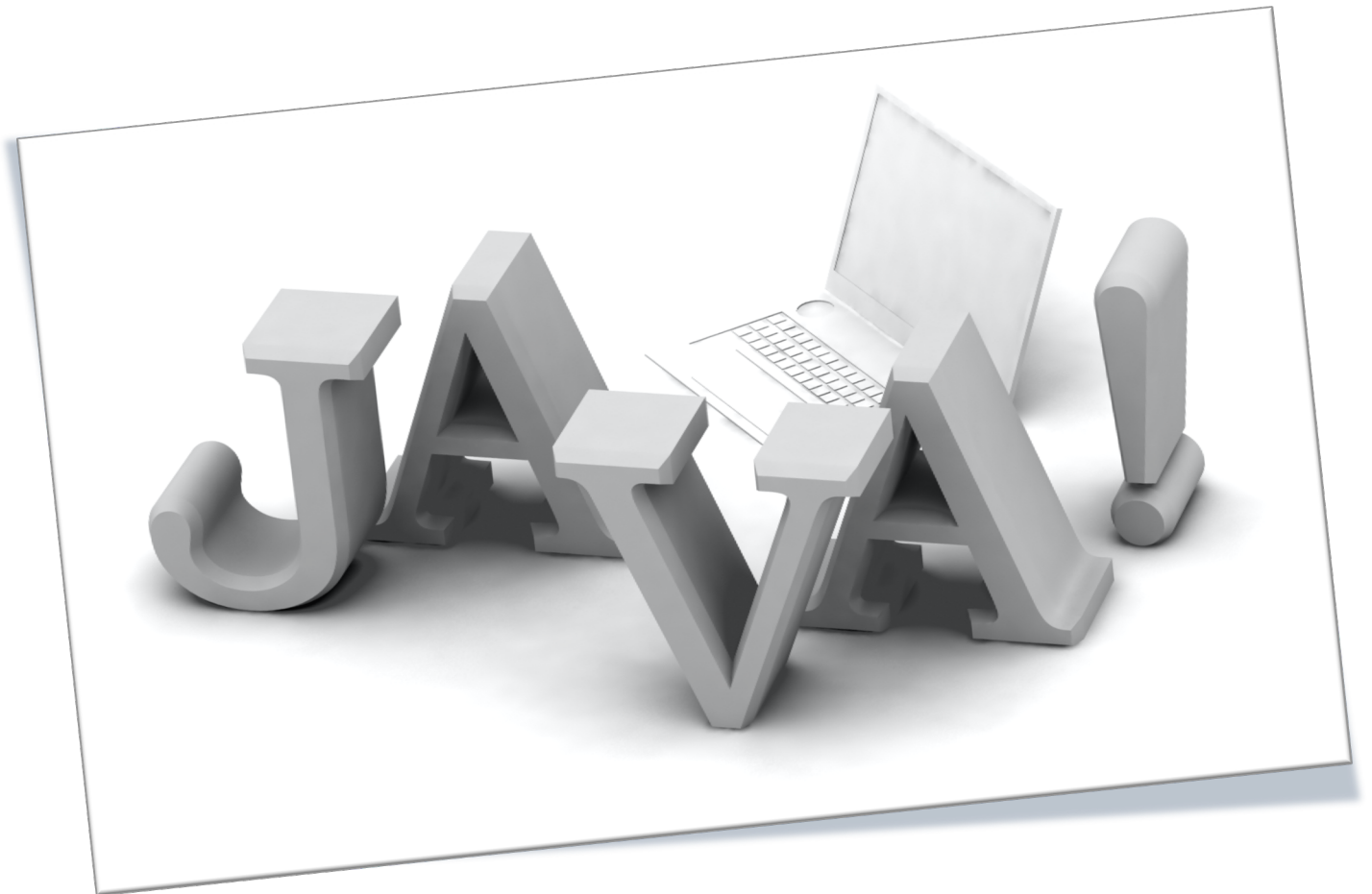

PRÁCTICAS

Capítulo 7: Colecciones



Unidad 7: Colecciones. Práctica 7.2

Crea un proyecto que se llame Capítulo7

El proyecto debe estar en el paquete `org.iesjaroso.daw.capitulo7`.

Organiza los ejercicios en paquetes si lo consideras necesario.

Para probar el ejercicio lo haremos una llamada desde el método principal de la clase `Capitulo7`

Ejercicio 1. BlackJack. Se desea programar el famoso juego de cartas BlackJack.





El blackjack, también llamado veintiuno, es un juego de cartas, propio de los casinos con una o más barajas inglesas de 52 cartas sin los comodines, que consiste en sumar un valor lo más próximo a 21 pero sin pasarse. En un casino cada jugador de la mesa juega únicamente contra el crupier, intentando conseguir una mejor jugada que este. El crupier está sujeto a reglas fijas que le impiden tomar decisiones sobre el juego. Por ejemplo, está obligado a pedir carta siempre que su puntuación sume 16 o menos, y obligado a plantarse si suma 17 o más. Las cartas numéricas suman su valor, las figuras suman 10 y el As vale 11 o 1, a elección del jugador. En el caso del crupier, los Ases valen 11 mientras no se pase de 21, y 1 en caso contrario. La mejor jugada es conseguir 21 con solo dos cartas, esto es con un As más carta de valor 10. Esta jugada se conoce como Blackjack o 21 natural. Un Blackjack gana sobre un 21 conseguido con más de dos cartas.

El crupier reparte dos cartas visibles a cada jugador. El valor del As es 11 o 1, las figuras valen 10, y las cartas numéricas su valor natural. El valor del As puede cambiarse según la necesidad de no pasarse de 21. Si al jugador le sale un As junto con una carta de valor 10, obtiene blackjack, ganando la apuesta salvo que el crupier obtenga también blackjack. Al terminar de repartir las dos primeras cartas a cada jugador, el crupier pondrá luego su primera carta boca arriba de manera que sea visible para el resto de jugadores, quienes podrán tomar sus decisiones en función de esa carta. Cada jugador compite únicamente contra el crupier, siendo indiferente las cartas que tengan el resto de jugadores.

Cada jugador tiene la posibilidad de plantarse y quedarse con cualquier puntuación, o de pedir más cartas hasta alcanzar los 21 puntos. Alcanzar los 21 puntos con más de una carta no se considera blackjack, siendo por tanto esa jugada inferior al blackjack con dos cartas. Si al pedir una nueva carta se pasa de 21, pierde automáticamente la apuesta y sus cartas y apuesta serán retiradas por el crupier. Cuando todos los jugadores hayan pedido sus cartas, el crupier mostrará su segunda carta y sacará más cartas si fuera necesario hasta sumar 17 o más puntos, momento en el que se plantará.

Para jugar al blackjack se necesita una baraja de naipes o cartas y jugadores.

Un naipe está representado por dos datos, el palo y la figura o número. La baraja está dividida en cuatro palos (en inglés: *suit*), dos de color rojo y dos de color negro:

-  → **Espadas** (conocidas como **picas**).
-  → **Corazones** (conocidos como **copas**).
-  → **Rombos** (conocidos como **diamantes**, **oros** o **cocos**).
-  → **Tréboles** (conocidos como **flores** o **bastos**).

Cada palo está formado por 13 cartas, de las cuales 9 cartas son numerales y 4 literales. Se ordenan de menor a mayor "rango" de la siguiente forma: **A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K**. Las cartas con letras, las figuras, se llaman *jack*, *queen*, *king* y *ace*.

Los nombres de las figuras provienen de personajes de la realeza y en inglés se llaman *court cards*. La carta J o *Jack* es conocida como jota o sota y representa a un sirviente. La Q o *Queen* es la reina y la K o *King* el rey.

Clase Naípe

Para implementar la clase Naípe se necesita:

- Un dato para almacenar el palo.
- Un dato para almacenar la figura.
- Se desea almacenar también el color.

Crea dos implementaciones distintas una basada en atributos de tipo int o de tipo String para representar el palo y la figura y otra basada en un enum:

Por ejemplo para las figuras:

```
public enum Figura {
    AS(11),
    DOS(2),
    TRES(3),
    CUATRO(4),
    CINCO(5),
    SEIS(6),
    SIETE(7),
    OCHO(8),
    NUEVE(9),
    DIEZ(10),
    JACK(10),
    QUEEN(10),
    KING(10);

    private final int valor;

    private Figura(int valor) {
        this.valor = valor;
    }

    public Figura seleccionFigura() {
        Figura[] figuras = values();
        return figuras[(int) (Math.random() * figuras.length)];
    }

    public String nombreCorto(){
        String nombre = this.name();
        int pos = this.ordinal();
        if(pos==0 || pos>9) {
            return nombre.charAt(0)+" "; //Imprime el primer carácter
        }

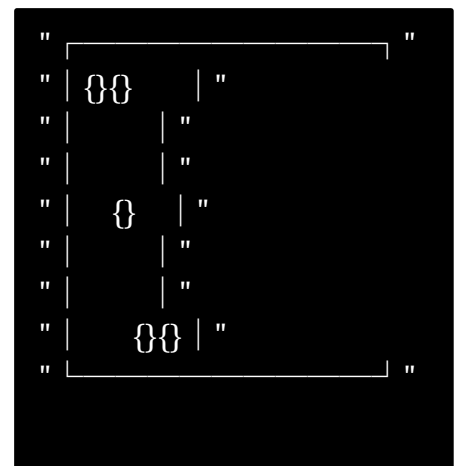
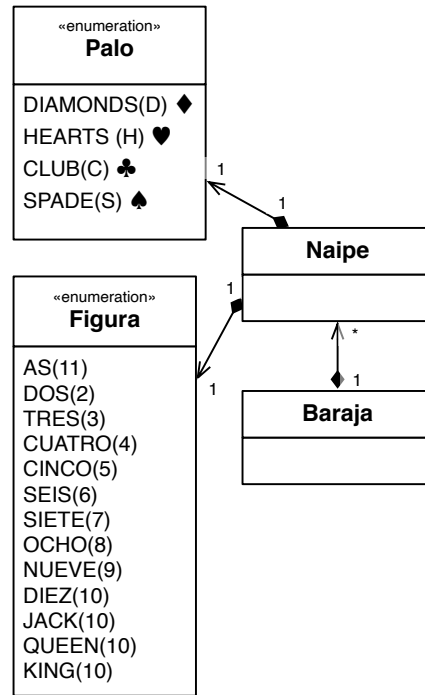
        return this.ordinal()+1+" "; //O el valor de las cartas numéricas
    }

    public int getvalor() {
        return valor;
    }
}
```

Algo similar para los palos...

Métodos:

- Constructor parametrizado
- getPalo(), getFigura(), getColor(), getValor(), getNaípe()
- equals(). 2 cartas son iguales si tienen el mismo palo y figura
- toString() muestra una carta con el formato de la derecha y usando el método replace para {}{} == A♥ y {}==♥



- String cartaReverso(). Muestra una carta del reverso:

[illegible]

```
for(String item: reverso) {
    System.out.println(item);
}
```

- `public String nombre()`. Muestra el nombre corto de una carta "2S" == "2♠", "JC" == "J♣"
- ¿Setters? En principio no

Clase Baraja.

Una Baraja es una colección de Naipes. Luego tiene los siguientes atributos y métodos.

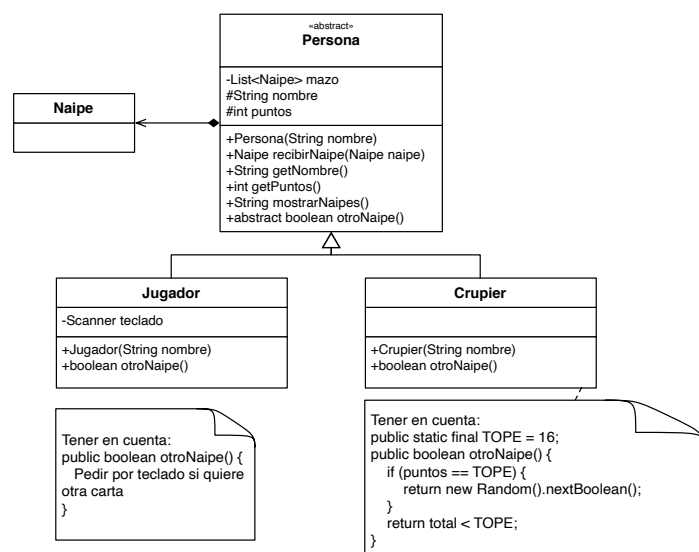
Atributos:

-List<Naipes> baraja

Métodos:

- Baraja(). Crea la baraja entera. Se recomienda bucles for...
- +void barajar(). Desordena la baraja de cartas
- +boolean cortar(). Corta la baraja, colocando la mitad de los naipes al final.
- +int size(). número de cartas en la baraja
- +Naipе robar(). Extrae y devuelve un naipе del final de la baraja
- +Naipе robar(int pos). Extrae y devuelve un naipе del final de la baraja de una posición.
- +boolean poner(Naipе naipе). Coloca un naipе al final de la baraja
- +boolean poner(Naipе naipе, int posicion)
- +boolean isVacia(). Devuelve true si la baraja está vacía
- +String toString(). Muestra todas las cartas de la baraja
- +boolean isNaipе(Naipе naipе). Devuelve true si existe una carta en la baraja
- +static int valor(List<Naipе> naipes). Devuelve el valor entero de una mano de Naipes. Por ejemplo: "2S AH" == 2+11 = 13
- +static int valorMin(List<Naipе> naipes). Devuelve el valor entero de una mano de Naipes. Pero tiene en cuenta el valor mínimo del AS. Por ejemplo: "2S AH" == 2+1 = 3
- +void ordenar(). Establece el orden de cartas por figuras.

Clase Persona, Jugador y Crupier. De acuerdo con el siguiente diagrama de clases: Según las “reglas de negocio” nos dice que hay un jugador que hace de banca(Crupier) y otros jugadores. Obviamente son objetos parecidos, pero que tienen comportamientos diferentes, al menos en la parte de seguir jugando. Esto justifica este diagrama de clases.



Clase Blackjack

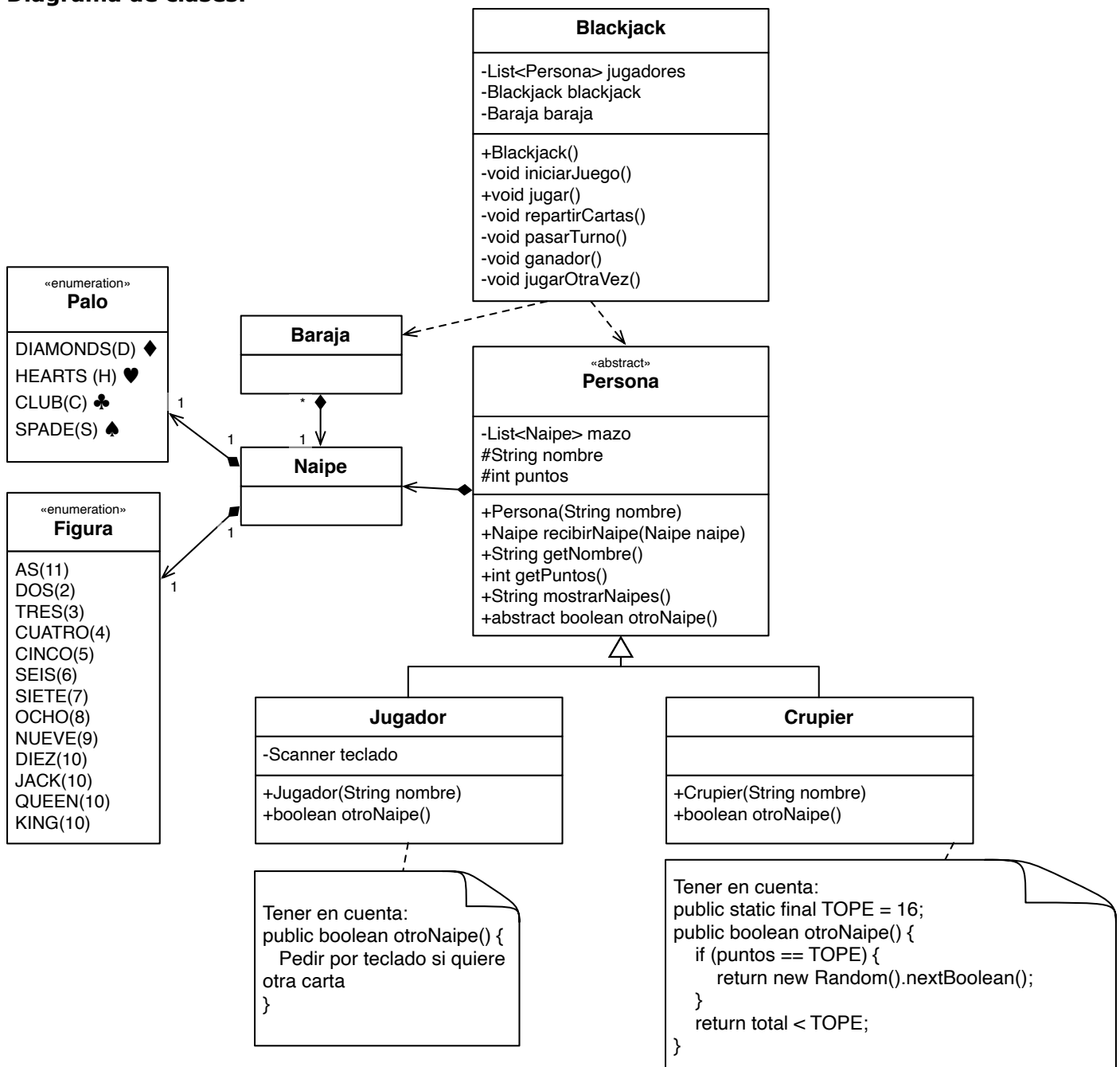
En esta clase es dónde está la clase principal y es la que hace y controla el flujo principal del programa.

Tenemos los métodos:

- +void jugar():

```
public void jugar() {
    repartirNaipes();
    cambiarTurno();
    ganador();
}
```

Blackjack
-List<Persona> jugadores -Blackjack blackjack -Baraja baraja
+Blackjack() -void iniciarJuego() +void jugar() -void repartirCartas() -void pasarTurno() -void ganador() -void jugarOtraVez()

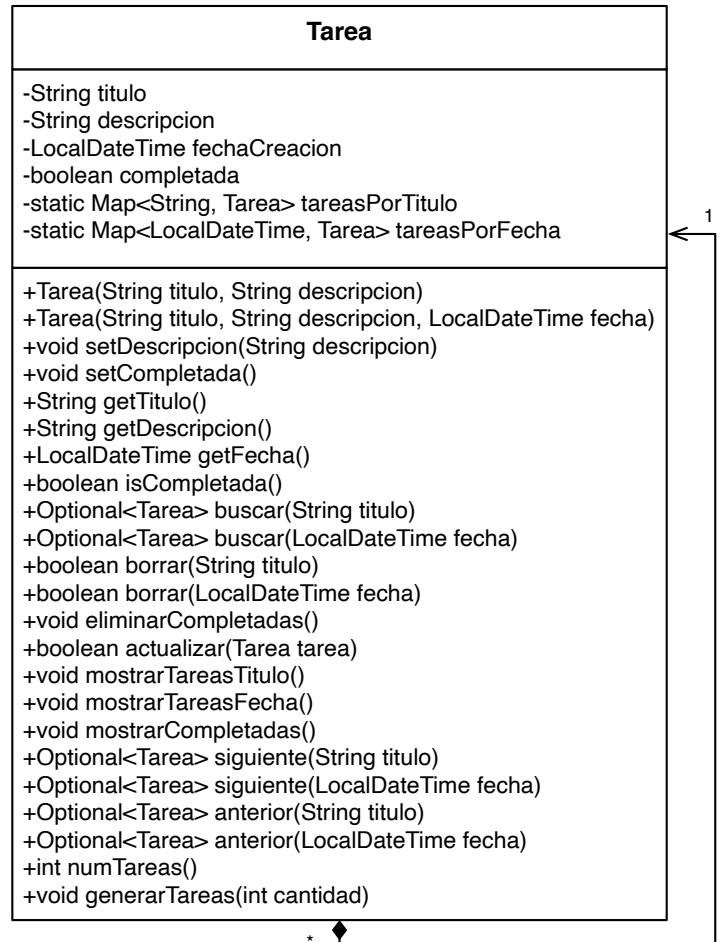
Diagrama de clases.

Ejercicio 2. Maps. Clase Tarea.

Se desea implementar una clase Tarea con los siguiente miembros:

Atributos miembros:

- String titulo: Título de la tarea. Por ejemplo "Hacer compra" No se permiten modificaciones.
- String descripcion: Descripción detallada de la tarea a realizar. Por ejemplo: "Comprar: pan, leche, huevos, agua"
- LocalDateTime fechaCreacion: Fecha de cuando se creación de la tarea, normalmente la fecha y hora del sistema, aunque desde el constructor se puede establecer.
No se puede cambiar una vez establecida
- boolean completada. Si es true quiere decir que la tarea ha sido completada.
- static Map<String, Tarea> tareasPorTitulo: en esta estructura se almacenan todas las tareas. OJO. Cuando se crea un objeto Tarea (llamada al constructor) SE DEBE añadir a este mapa. El String del mapa es el título de la tarea, y Tarea la tarea que se está instanciando. Este mapa usará la implementación LinkedHashMap.
- static Map<LocalDateTime, Tarea> tareasPorFecha: en esta estructura se almacenan todas las tareas. OJO. Cuando se crea un objeto Tarea (llamada al constructor) SE DEBE añadir a este mapa. El LocalDateTime del mapa es la fecha de la tarea, y Tarea la tarea que se está instanciando. Si la tarea no usa el constructor Tarea(String titulo, String descripcion, LocalDateTime fecha), se usará como fecha la del sistema: LocalDateTime.now(). Este mapa usará la implementación TreeMap



NOTA: Todas las tareas quedan registradas dos veces (tareasPorTitulo , tareasPorFecha) en la propia clase. Tened en cuenta que estas dos variables son static, luego están compartidas por todos los objetos de la clase.

Métodos miembros:

+Tarea(String titulo, String descripcion). Constructor. Por defecto fechaCreacion se establece con la fecha y hora del sistema y completada se establece a false. **IMPORTANTE.** Se debe añadir a los mapas tareasPorTitulo y tareasPorFecha ESTA (this) la actual tarea, previa ajuste de todos sus atributos

+Tarea(String titulo, String descripcion, LocalDateTime fecha). Ídem al anterior pero nos permite establecer una fecha. **REUTILIZA CÓDIGO!!**

+void setDescripcion(String descripcion): permite cambiar la descripción de una tarea

+void setCompletada(). Cambia una tarea al estado de completada. Una vez cambiado este estado no se puede cambiar.

+String getTitulo(). Obtiene el título de la tarea.

+String getDescripcion(). Obtiene la descripción

+LocalDateTime getFecha(). Obtiene la fecha y hora de creación

+boolean isCompletada(). Nos dice si una tarea está completada

+Optional<Tarea> buscar(String titulo). Busca una Tarea por título. Obviamente, esta tarea se debe buscar en el mapa tareasPorTitulo. Devuelve un Optional<Tarea> para evitar nulos.

+Optional<Tarea> buscar(LocalDateTime fecha). Busca una Tarea por fecha. Obviamente, esta tarea se debe buscar en el mapa tareasPorFecha. Devuelve un Optional<Tarea> para evitar nulos.

+boolean borrar(String titulo). Borra una tarea en los mapas tareasPorTitulo y tareasPorFecha. Antes de eliminar se debe buscar, si no se encuentra no se puede eliminar (return false). **CUIDADO** se debe borrar en ambos mapas para mantenerlos sincronizados.

+boolean borrar(LocalDateTime fecha). Borra una tarea en los mapas tareasPorTitulo y tareasPorFecha. Antes de eliminar se debe buscar, si no se encuentra no se puede eliminar (return false). CUIDADO se debe borrar en ambos mapas para mantenerlos sincronizados.

+void eliminarCompletadas(). Elimina en ambos mapas todas las tareas completadas. Se recomienda usar entrySet() y keySet() de los mapas para recorrerlos.

+boolean actualizar(Tarea tarea). Actualiza una tarea en ambos mapas. Una tarea dispone de titulo y fecha...Si un mapa usa la misma clave, actualiza el valor.

+void mostrarTareasTitulo(). Muestra todas las tareas por título

+void mostrarTareasFecha(). Muestra todas las tareas por fecha

+void mostrarCompletadas(). Muestra las tareas completadas

+Optional<Tarea> siguiente(String titulo). Devuelve la tarea siguiente según el titulo

+Optional<Tarea> siguiente(LocalDateTime fecha). Devuelve la tarea siguiente según la fecha

+Optional<Tarea> anterior(String titulo) Devuelve la tarea anterior según el titulo

+Optional<Tarea> anterior(LocalDateTime fecha)). Devuelve la tarea anterior según la fecha

+int numTareas(). Número de tareas disponibles. Como los mapas están sincronizados da igual cuál se consulte.

+void generarTareas(int cantidad). Genera una cantidad aleatoria de tareas. Usar un String [] para los datos.

Sobrescribe los métodos:

+toString(): Devuelve una cadena <Titulo> <Descripcion> <Fecha 20 MARCH 2020 – 13:40:25> <No completada>

+equals(): Dos tareas son iguales si tienen el mismo título o la misma fecha y hora (completa)

Implementa el orden por defecto usando fechaCreacion

Ejercicio 3. Programación Funcional. Streams.

Utiliza de la relación anterior (Colecciones. Práctica 7.1) de los ejercicios 2 y 3 las Clases Persona y Grupo. Escribe en el método main, usando programación funcional, Streams y Funciones Lambda, el código necesario que permita resolver las cuestiones:

- a) Genera un grupo con 1000 personas ordenadas por fecha de nacimiento DESCENDENTEMENTE y cuya fecha de nacimiento sea entre 1-1-1970 y 31-12-2010 y almacena el resultado en una lista llamada grupoA

A partir de grupoA genera un Stream por cada consulta. Fíjate en la clase de ejemplo de los Streams:

- b) Cuenta el número de personas cuyo apellido empiece por "M".
- c) Cuenta el número de personas cuya edad este entre los 18-30 años.
- d) Muestra las personas que su nombre comience por una vocal y su apellido1 sea "Moreno" o "Castro" y sean mayores de edad. Se ordenará por nombre, apellido1 y apellido2.
- e) Cuántas personas han nacido en el año 2000.
- f) Cuántas personas que son del signo del zodiaco Sagitario.
- g) Muestra las personas cuyo nombre comienzan por "A" y que ocupan la posición de la 3-7 (recuerda que tienes el método skip).
- h) Muestra todas las personas que su nombre completo (nombre+apellido1+apellido2) la suma de sus vocales sean mayor o igual que 5.
- i) Muestra las personas que cumpla hoy años.
- j) Muestra el nombre completo más largo.
- k) Muestra el nombre completo más corto.
- l) Muestra todos los Familiares (si existen) de las 5 primeras personas.
- m) Muestra todos los hermanos.
- n) Muestra todas las personas nacidas en años bisiestos.
- o) Muestra los 3 primeros nombres completos del revés (dando la vuelta a los caracteres).
- p) Muestra las siglas del nombre, apellido1 y apellido2 de las 10 primeras personas.
- q) Agrupa por nombre calculando el porcentaje de cada nombre que hay
- r) Agrupa por edad contabilizando cuántas personas hay de una determinada edad.
- s) Agrupa las personas por apellido1.
- t) Borra todas las personas que tengan el apellido2 terminado en "z".
- u) Borra todas las personas que sean menores de 5 años.