



# Introducción

Esta es la implementación del Trabajo práctico integrador, en el cual se pedía diseñar un sistema para emular el álbum de figuritas del mundial para una empresa. Se aclaraba que hay 3 tipos de álbumes diferentes, con sus características y se daban una serie de especificaciones esperadas que ya se han entregado en la primera parte. En esta segunda parte veremos como se dio la implementación, cuales son los Irep que se respetaron, qué herramientas de java enseñadas en clase se utilizaron y analizaremos el orden de complejidad de algunos métodos.

# Antes de empezar

Previamente, como se había mencionado, se hizo entrega de un diseño para la primera parte del TP. En general, se mantuvo el mismo diseño de la primera parte, si embargo, apoyado en las correcciones del profesor, se hicieron ligeras modificaciones en el diseño.

Se reemplazo el diseño AlbumDelMundial→Album→Usuario.

por AlbumDelMundial→Usuario→Album.

*De esta manera, ahora el sistema contiene usuarios que contienen albumes.*

Por practicidad se cambió el tipo de dato HashCode de String a int.

# Desarrollo

Teniendo el diagrama del diseño en mano, se fue creando todas las clases necesarias, y con ellas sus métodos vacios, solo con la descripción, es decir se fue estructurando el código.

Luego, usando la **interface** pasada por el aula virtual,de manera que

public class IAlbumDelMundial implements InterfazPublicaAlbumDelMundial {

se fueron implementando los métodos solicitados, y junto con ellos , cada vez que se requería, se implementaban los métodos de las clases , métodos públicos y privados.Luego, los Test fueron muy útiles, a la hora de probar los métodos, junto con algunos checkPoint que se irían poniendo en el codigo para marcar donde se encuentra el problema lógico. En general, estos problemas de interpretación eran los que mas tiempo llevo resolver, dado que es un programa tan grande. Por ejemplo, un problema que llevo tiempo, en el public abstract class Album {cual el programa no se comportaba como se esperaba, se daba porque una variable booleana , que se creia inicializada en false, estaba inicializada en true.

El orden de desarrollo fue masomenos el siguiente:

Primero se desarrollo la clase public abstract class Album {, la cual es **abstracta** por su método

public abstract String darpremioFinal();

luego albumTradicional y Extendido, por ultimo albumWeb. La mayor parte de los métodos implementados se encuentran en la clase *Album*, ya que todos los albumes se comportan masomenos de la misma manera, y todos **heredan** de *Album*. A medida que se desarrollaban las clases, se hacia visible la necesidad de algunos métodos privados, auxiliares que proveyeran a los métodos principales.

Luego se paso a desarrollar la clase Usuario y Figurita , Esta parte no fue tarea difícil, ya que solo presentan atributos, métodos getters,y algún que otro método necesario.

Mas tarde, se movió parte del codigo que se encontraba en la clase IAlbumDelMundial, a la clase Fabrica, para que esa clase no este sobrecargada, porque hasta ese momento, IAlbumDelMundial se encargaba de crear todos los otros objetos.

Finalmente, luego de pasado todos los test,se probo que el codigo cliente funcione correctamente, con ello se desarrollo el método toString del sistema, y junto con la prueba del codigo cliente, se solucionaron algunos problemas invisibles del método completoElAlbum. Más tarde, se desarrolló la clase FiguritaTop10 junto con lo necesario para que el albumExtendido funcione correctamente. Uno de esos métodos necesarios fue el método private void verificar() { el cual ya se encontraba en la clase *Album,* sin embargo, dado que este album es extendido, fue necesario modificarlo ligeramente para que en lugar de chequear solo las 384 figuritas iniciales, tambien revise las otras 20 del top 10, para dar cuenta si el album esta lleno. Por lo tanto, dado que para el mismo mensaje cada clase muestra un comportamiento distinto, este método se considera **polimórfico**

Y por último se hicieron algunas correcciones solicitadas por el docente, y se formateo el código para que quedara más fácil de leer.

# Requerimientos Tecnicos

* ~~- Grupos :El mismo grupo de la primera parte del TP. Si hay alguna modificación debe ser aprobada por sus docentes de la comisión.~~ Esto no se pudo cumplir.
* Stringbuilder, cuyo uso debe basarse en la necesidad de modificar el string.
* Iteradores y Foreach para recorrer las colecciones de Java

**Stringbuilder**

En varias partes del programa se utilizó esta herramienta, pero una sencilla es cuando se aplicó en el método crearStringRandom(int n) , el cual dado un número n que representa la cantidad de caracteres, crea un String completamente aleatorio, alfanumérico con ese numero de caracteres.

Cabe resaltar que el método es recursivo. Veámoslo:

private String crearStringRandom(int n) {

// elige un caracter al azar de la String

String AlphaNumericString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

+ "0123456789"

+ "abcdefghijklmnopqrstuvxyz";

StringBuilder sb = new StringBuilder(n);

// 0 y AlphaNumericString variable length

int index = (int) (AlphaNumericString.length()

\* Math.random());

sb.append(AlphaNumericString

.charAt(index));

if (n <= 1) {

return sb.toString();

} // caso base

else

return sb.append(crearStringRandom(n - 1)).toString();

}

**Iteradores**

Por su parte,a Iteradores se uso en mas de una ocasión, por ejemplo en el metodo VerificarYPegarFigus, que pega las figuritas que correspondan y ademas revisa si se completo el album.

Fue necesario ya que debia recorrer el ArrayList de figuritasSinpegar y quitar un elemento de ese arraylist al mismo tiempo que lo recorre, esto con un forEach normal hubiera causado problemas por los hilos de ejecución. Veamos la implementacion.

public ArrayList<String> verificarYpegarFigus() {// bueno aca use Iterator asi que no se pueden quejar

ArrayList<String> ret = new ArrayList<String>();// esta es la lista a retornar.

ListIterator<Figurita> algo = figuritasSinpegar.listIterator();// el iterador

// int checkPoint=0;

Figurita actual;// va almacenar el iterador actual.

if (this.AlbumCompleto == true) {// System.out.println("Album lleno, nada para agregar.");

return null;

} else {

while (algo.hasNext()) {

// checkPoint=checkPoint+1;

actual = algo.next();// uso esto porque si llamo a algo.next() mas de una vez pasa al siguiente en el

// mismo ciclo

if (actual.getNumeroQueIdentifica() < coleccion.length

&& coleccion[actual.getNumeroQueIdentifica()] == null

&& actual.getTipoDeFigu().equals(this.getTipoAlbum())) {// si la posicion es null, quiere decir

// que la figu no esta pegada.

// o la figu esta pegada mal o la figu no esta pegada. ademas las figuritas

// deben ser del mismo tipo

coleccion[actual.getNumeroQueIdentifica()] = actual;// agrego al array, espero no me de problema de

// ejecucion.

ret.add(actual.toString());// Agrego la figu a la lista de retorno

algo.remove();// y lo quito de la lista

} else if (!coleccion[actual.getNumeroQueIdentifica()].equals(actual)) {// si esta pegada pero es

// distinta.

System.out

.println("ups, lo sentimos algo salio mal y pegaron una figurita en el lugar equivocado.");

} // Esperemos que esto nunca pase.

// Si esta pegada pero es igual no se hace nada

}

}

// System.out.println("Cant veces entro en while = "+checkPoint);

verificar();// obviamente despues de pegar las figus, verificamos si completamos el album y

// cambiamos el estado

return ret;

}

**ForEach**

Con respecto al forEach se utilizo todas las veces que se pudo, debido a que es una herramienta que facilita muchísimo recorrer colecciones. En este caso, se uso parael metodo buscarRepetidas(), un metodo que devuelve el numeroQueIdentifica de figurita de la primer figurita que encuentre repetida. Como se puede observar, se recorre la colección de figuritasSinpegar.

public int buscarRepetidas() {

for (Figurita iter : figuritasSinpegar) {

if (this.contains(iter) || estaRepetidaEnsinPegar(iter)) {

return iter.getNumeroQueIdentifica();

}

}

return -1;

};

# Invariante de Representación IREP

Los Invariantes de Representacion son los siguientes:

para la clase Usuario

0<dni<100.000.000 (Enteros) (Es el rango de dnis validos en Argentina,no queremos dnis negativos, ni tampoco flotantes)

albumpropio extends Album (El tipo de dato de atributo albumPropio solo puede ser alguna instancia de album, es decir no queremos por ejemplo albumPropio = bicicleta.)

para la clase Figurita

nombre (Los nombres de las figuritas deben ser algunos de los nombres de los que cree la clase fabrica.)

tipoDeFigu (Solo puede ser Tradicional, Web o Extendido)

valorBase (Solo pueden ser valores positivos)

rankingValorBase (Esto solo puede un valor del ranking que genere la clase fabrica)

numeroQueIdentifica (este valor debe ser único para cada figurita que tenga distinto nombre, es decir no puede ser que figuritas con nombres iguales, pais iguales, todo el resto de atributos iguales, tengan numeroQueIdentifica diferente.)

para la clase FiguritaTop10

sedeMundialYanio ( solo pueden ser años pasados, los años deben estar a distancia de múltiplo de 4 entre si y países donde se jugo el mundial)

balonde (Solo puede ser Oro y plata)

para la clase *Album*

tipoAlbum (Solo puede ser Extendido,Tradicional o Web)

cantPaises( solo puede ser 32)

cantJugadores(solo puede ser 12, por enunciado)

codigoUnico(No pueden haber dos códigos únicos iguales amenos que sea el mismo album, de la misma persona.)

dniDueno( mismos parámetros que dni de la clase usuario, si en usuario no se admite -2, aca tampoco)

coleccion (solo puede contener tantas figuritas como cantidad de países por cantidad de jugadores (384), no puede tener figuritas repetidas ni de las del top10, solo puede tener figuritas del mismo tipo que el album)

figuritasSinpegar (puede tener todo tipo de figuritas, puede tener repetidas)

paisesCompletos( solo puede tener tantas entradas como cantPaises (32), si un país es marcado como completo debe haber 12 figuritas de ese pais en coleccion, las keys solo pueden ser países de la lista de países participantes de factory, no pueden haber entradas repetidas)

albumCompleto( solo puede estar en true si se completo el album)

para clase AlbumExtendido

los mismos que album ya que hereda de *Album,* pero además.

cantExtend=20 (cantidad de figuritas adicionales)

coleccion20 ( solo puede tener igual de figuritas como cantExtend,20. )

para clase Fabrica

cantPaises=32 (por enunciado)

paisesParticipantes (la entrada debe ser igual a cantPaises, y no se pueden repetir entradas.)

lugaresPorPais(solo puede ser 12, como la cantidad de jugadores, dado que cada pais solo puede tener 12 lugares en el album

ranking(no se pueden repetir las entradas , es decir paises, ni tampoco las salidas(posicion en el ranking), la saida no puede ser un valor negativo)

listadoDeMundialesTop10 ( solo puede contener paises que albergaron el mundial y fechas de mundiales)

balonYpaisPorMundial (solo puede tener como entrada los datos de listadoDeMundialesTop10 y como salida paises que esten en el ranking)

nombres (no puede tener menos de 20 entradas)

apellidos(no puede tener menos de 20 entradas, esto es para que de diversidad de nombres y apellidos de jugadores y que no haya figuritas con mismos nombres y apellidos, si bien esto puede pasar en la realidad y no es un verdadero condicionante, decidí que no estaria bueno que por ejemplo haya 384 leonel messi, uno de cada pais y con un numero distinto. De esta forma garantizo que hay al menos 400 combinaciones de nombres distintas)

coleccion (debe tener tantas figuritas como cantPaises por lugaresPorPais)

coleccion20(debe tener 20 figuritasTop10)

para IAlbumDelMundial

dniToHash (como entrada debe tener dnis, no puede tener mas entradas que dnis posibles,no puede tener ni entradas repetidas ni salidas repetidas.)

participantesConAlbumes (como entrada tiene los hash de salida de dniToHash, no puede tener una entrada que no se encuentre en los valores de salida de dniTohash, debe tener igual de entradas que dniToHash, debe tener igual de entradas que salidas, como salida debe tener el participante correspondiente al hash de la entrada. No puede tener salidas repetidas.)

codigoWeb (debe tener igual cantidad de entradas y salidas que participantes con album web en ParticipantesConAlbum, una vez que los valores booleanos de la salida cambian no debe ser posible modificarlos.)

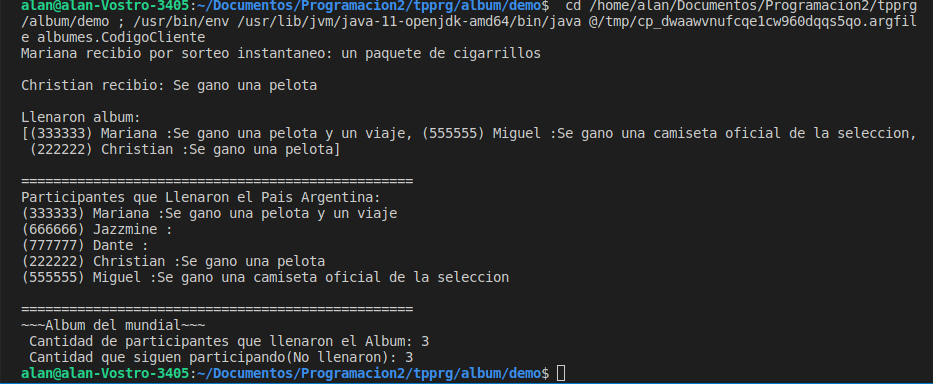
sorteoDado (la entrada solo puede ser hash contenidos en la entrada de participantesConAlbumes, no puede tener mas entradas que ParticipantesConAlbumes, una vez modificado el booleano de la salida no debe ser posible cambiarlo,si un participante recibió el sorteo, entonces debe aparecer su hash entre estas entradas.)

ganadores (no puede tener nodos que entradas en DniToHash,solo pueden aparecer participantes que hayan completado el album, no se puede borrar ningún nodo.)

# Consideraciones Importantes

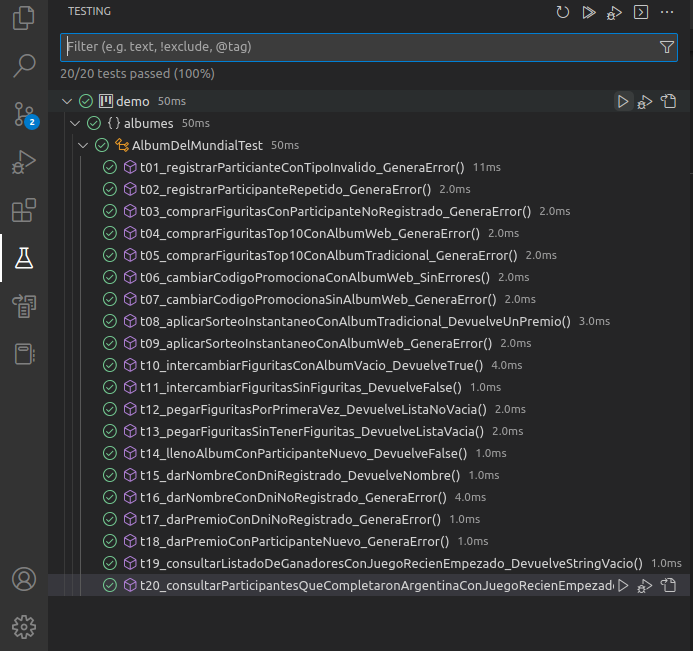
Se puede señalar que el Trabajo cumple con los siguientes puntos.

**☑**Deberá correr satisfactoriamente con el código cliente entregado:

Correr, lo corre sin errores. 

**☑**Deberá pasar satisfactoriamente el test junit proporcionado.

Pasa el test.



**☑**Deberá aprovechar correctamente las estructuras de datos elegidas.

**☑**El código debe tener implementado el método toString del Sistema de Album.

public String toString() {

StringBuilder ret = new StringBuilder();

ret.append("~~~Album del mundial~~~ \n Cantidad de participantes que llenaron el Album: " + ganadores.size()

+ "\n Cantidad que siguen participando(No llenaron): " + (DniToHash.size() - ganadores.size()));

return ret.toString();

}

**☑**Se debe responder si un Participante cualquiera, dado su dni, completó su álbum en O(1).

Para ello se implemento una variable Booleana que diga si esta completo en la clase album, y lo único que hace el metodo desde el sistema es hacer un get de esa misma variable.

(Tambien hace get del album del participante, y get del dni, pero como todas estas operaciones estan en O(1), y sobre un map hacer un get, tambien es O(1), porque ya sabe la posición de memoria, y no recorre, entonces todos los O(1) sumados quedan en O(1).

public boolean llenoAlbum(int dni) {

if (DniToHash.get(dni) == null) {

throw new RuntimeException("El participante no se encuentra registrado");

}

return participantesConAlbumes.get(DniToHash.get(dni)).completeElAlbum();

}

* **Explicar por escrito, la complejidad del método int buscarFiguritaRepetida(int dni):**

**public int buscarFiguritaRepetida(int dni){**

**if(!DniToHash.containsKey(dni)){throw new RuntimeException("El participante no se encuentra registrado");}**

**else{ return participantesConAlbumes.get(DniToHash.get(dni)).getAlbumpropio().buscarRepetidas();**

**}}**

Este es el metodo en la clase IAlbumDelMundial:

Comienza haciendo 2 gets a el diccionario O(1)+O(1), luego hace un get de albumPropio(Devuelve el album del participante) +O(1)

Y recien en ese momento, llama a buscarRepetidas(); que se encuentra en la clase Album.

Veamos la clase Album:

public int buscarRepetidas(){

for (Figurita iter : figuritasSinpegar) {

if(this.contains(iter)||estaRepetidaEnsinPegar(iter)){

return iter.getNumeroQueIdentifica();

}

}

return -1;

};

Lo que observamos, es que Hacemos un forEach O(n) y dentro de este, llamamos a las funciones contains() y estaRepetidasinpegar().

public boolean contains(Figurita figu){

if(coleccion[figu.getNumeroQueIdentifica()]!=null){//si la posicion no es null, esta ocupada por una carta, que deberia ser la correcta.

return true;

}

Contains, revisa el album y accede por posicion, es decir O(1), por lo que contains, no significa un problema, mientras que…

private boolean estaRepetidaEnsinPegar(Figurita figu){//revisa si esta repetida entre las sin pegar

int cont=0;

for (Figurita iter : figuritasSinpegar) {

if(figu.equals(iter)){

cont=cont+1;

}

if(cont>1){return true;}

}

return false;}

estaRepetidaEnsinpegar usa otro ciclo forEach para comparar la figurita pasada por parametro con las que estan repetidas, y si aparece mas de una vez, devuelve True.

estaRepetidaEnsin pegar es o(n).

Volviendo al metodo **int buscarFiguritaRepetida,** Notamos que hace un ciclo for each y contains, dentro de otro ciclo forEach. por lo tanto tenemos O(n\*\*2), es decir es de orden de complejidad cuadratica.

# Conclusiones

En general fue muy grato hacer el trabajo, sobre todo en la segunda parte de implementacion. Costo un poco amoldar el diseño a la interface solicitada, se tuvo que hace un click, cambiar un poco el punto de vista. Pero después de todo sirvio para reforzar los conocimientos y fundamentos de programación aprendidos en clase, se intento descargar lo mayor posible de lo aprendido en el codigo. En su momento salieron a la luz ciertos problemas de lógica y sintaxis , por la falta de practica con el lenguaje, pero con el paso del tiempo se pudo recuperar el ritmo y llevar este trabajo a lo que se ideo desde un principio. Quedaron algunos detalles que hubieran estado bueno implementar, de ideas que llegaron a la cabeza al momento del primer diseño, como por ejemplo cargar imágenes, abrir archivos y mostrar el album en primera persona, de forma interfaz cliente servidor, pero se llego con el tiempo. Se finalizo el trabajo de muy buena manera, gracias por la oportunidad de extensión.

Fin.