RESUMEN FINAL POO I

Type	RESUMEN
AUTOR	Juan Pablo Frascino
Reviewed	☐

INTRO POO

La programacion orientada a objetos nace como una forma de programar que busca facilitar la escritura, legibilidad e eficiencia del codigo. Es una forma de codificacion que busca acercarse a la realidad mediante la abstraccion de lo que se denomina objetos. En la poo todo es un objeto y los objetos se crean mediante planos/moldes que llamamos clases. Por eso a la hora de planificar una orientacion a objetos es importante una buena ingenieria de requisitos con el cliente, extraer todo lo quiere, armar un texto y de ahi sacar los sustantivos mas comunes que nos daran las clases que debemos crear, asi pudiendo comenzar a diagramar un diagrama de clases(nuestro plano de software).

CLASES-HERENCIA-INTERFACES

Las clases son como el molde de un objeto, ya que un objeto es una instacia de una clase.

El esqueleto de una clase se compone de 3 cosas fundamentales:

- El nombre
- Los atributos
- Los metodos

Y ademas cada uno de estos 3 pueden tener 3 tipos de acceso:

- +publicos
- -privados
- #privilegiados

Aunque por lo general y casi siempre los atributos son privados y los metodos son publicos(las clases tambien).

Los metodos de un objeto para ser utilizados deben ser llamados, esto se hace asi: objeto.llamarMetodo() ←esto es una llamada a metodo. Existen metodos static, que no necesitan ser llamados mediante un objeto.

Como dijimos que los atributos son siempre privados eso significa que no se pueden acceder fuera de su propia clase, entonces para cuando queremos acceder desde otro lado es que nace el ENCAPSULAMIENTO.

En el encapsulamiento se le agrega a una clase 2 metodos por atributo, uno set(para cambiar el valor del atributo) y uno get(para obtener el valor del atributo). Esto nos permite acceder y cambiar valores de los atributos por mas que sean privados.

Para crear un objeto a partir de una clase lo que utilizamos son constructores, y hay de dos tipos(puede haber uno o el otro, o ambos):

- POR DEFECTO: Nada entre parentesis, crea un objeto de estado vacio.
- SOBRECARGADO: Con datos entre parentesis(como parametros) que van a ser los datos en el estado del objeto, lo carga directamente sin necesidad de hacerlo con setters.

HERENCIA:

Las clases en java pueden heredar de otra clase(solo una), esto significa que una clase que hereda de otra(Extends) hereda todos los atributos y metodos de la superclase padre.

La clase hijo puede sobreescribir(crear un metodo con el mismo nombre de un metodo del padre) los metodos que hereda para definir su comportamiento propio, a esto lo llamamos overwriting(sobreescritura).

Debido a todo esto nace el polimorfismo, que es la capacidad que tienen los objetos de responder de diferente manera al mismo llamado. Para que ocurra esto debe haber sobreescritura(para que haya diferente comportamiento), herencia y upcasting anulado(esto lo que hace es que si el llamado es a la clase padre, el programa no transforme el objeto hijo en la padre sino que lo deje comportarse a su manera).

Las clases pueden ser:

- CONCRETAS: Es una clase instanciable que me permite crear objetos de ella
- ABSTRACTA: Es una clase no instanciable la cual no me deja crear objetos de ella, es una clase que debe heredar.

Asi como las clases puden heredar de otras clase tambien pude implementar interfaces, las interfaces se utilizan para heredar metodos o comportamientos. Y sus metodos son abstractos por lo que no tienen cuerpo y deben ser sobreescritos si o si en la clase que los implemente.

JAVA

Los programas en java corren en la ram, dentro de la ram se le asigna un espacio donde trabajar que se va a dividir en dos partes, una parte es la parte estatica o stack donde corre por ejemplo el main y luego esta la parte dinamica o heap que es donde se van creando los objetos y los metodos que se van agregando y eliminando durante su ejecucion.

Java es un LENGUAJE COMPILADO ya que a diferencia de los lenguajes INTERPRETADOS donde puedo elegir que linea quiero ejecutar y cual no, este ejecuta todas las lineas juntas. Luego pasa todo este codigo a un archivo intermedio llamado BYTECODE que nos permite ejecutar nuestro codigo en cualquier plataforma ya que es leido por la JVM(java virtual machine) que se encargara de traducir nuestro byte code a la cpu.

PROGRAMA→COMPILADOR→BYTECODE→JVM→EJECUCION

Java ademas es un lenguaje TIPADO osea que hay que definir el tipo de cada variable, esto lo hace mas seguro que uno NO TIPADO pero el no tipado es mas veloz.

Para que un codigo sea lo mas eficaz debe tener alta coesion(que cada clase se encargue de una tarea especifica) y bajo acoplamiento(menor vinculacion entre clases para no romper el codigo)

En java existe un tipo de dato llamado ENUM que permite guardar valores enumerados para luego ser reutilizados con objetos del mismo tipo enum.(es un array de datos)

Java no es un lenguaje puramente de objetos(a diferencia de por ej python) ya que permite el manejo tanto de tipo de variables objetos como de tipo de variables primitivas(int, float, double).

Java posee un GARBAGE COLLECTOR que va recorriendo el espacio de memoria del programa y lo optimiza eliminando variables/metodos no utilizados o que ocupan espacio inecesariamente.

Si un metodo es boolean se le suele llamar comenzando con “is”

UML

El UML(UNIFIED MODELING LANGUAGE) es un lenguaje de modelacion de diagrama de clases. Fue creado con el objetivo de estandarizar los planos de software en la programacion orientada a objetos. En el se representan las clases de un programa(con sus nombres, atributos y metodos) y sus relaciones entre ellas.

TIPOS DE RELACIONES EN EL UML:

- HERENCIA: Cuando una clase hereda de otra(es una flecha con la punta blanca)
- INTERFACE: Cuando una clase implementa una interface(es una flecha puntuada con punta blanca)
- ASOCIACION: no la usamos
- DEPENDENCIA: Cuando un objeto de otra clase aparece en los parametros de un metodo de otra clase, entonces la clase que lo tiene depende de la que tiene en los parametros.(flecha puntuada con la punta en forma de > apuntando a la cual depende)

- AGREGACION(PARTE DE): Cuando un objeto de una clase contiene a un objeto de otra clase en sus atributos. Tiene dos tipos AGRUPACION(rombo blanco) si no es esencial o COMPOSICION(rombo negro) si es esencial(la parte no pude existir fuera de la composicion)

Las clases abstractas tienen en nombre en italica, y las interfaces se les pone interface.

COLLECTIONS

De la clase collections heredan 4:

LIST

ARRAYLIST: Me permite guardar una cantidad ilimitada de elementos pero consume mas memoria que un array normal. Aqui los elementos se guardan uno al lado del otro por lo que borrar o agregar un dato en el medio es complicado ya que implica mover a todos los demas.

LINKEDLIST: Es igual que la arraylist pero se diferencia en la manera en que se guardan en memoria, a diferencia de la arraylist esta no necesita tener los datos uno al lado de otro sino que cada dato lleva consigo la direccion de memoria donde se encuentra el siguiente dato(si no hay siguiente datos es null). En java especifiacamente se encuentra doblemente encadenada osea apunta al proximo y al anterior, esto permite un eliminacion o adicion mas sencilla pero es mas lenta de recorrer a comparacion a la arraylist.

PILA/STACK: Conjunto de elementos del mismo tipo que puede crecer(push) o decrecer(pop) solo por uno de sus extremos(tope). Tiene una estructura LIFO(Last in, First out). Sirven para evaliar expresiones aritmeticas, analizar sintaxis. simular procesos recursivos, pasar recorridos binarios. Solo es visible el tope de la pila(peek).

QUEUE

COLAS/QUEUE: Es un conjunto de elementos en donde se ingresan(offer) todos los elementos por un extremo fondo y todos salen(poll) por el otro

extremo frente. Tiene una estructura FIFO(First in, First out). Solo podemos ver el frente(peek).

SET(conjuntos)

Implementan la interfaz set

Es una serie de elementos que respetan la TEORIA DE CONJUNTOS ya que no permite que existan elementos duplicados y no tiene orden entre sus elementos. Hay 3 tipos:

- TreeSet: Conjunto definido por la estructura arbol
- HashSet: Conjunto definido por la estructura hash
- LinkedHashSet: Conjunto definido por listad doblemente enlazadas

Aunque las 3 utilizan un tabla hash: codigoHexadecima | palabra

MAP(diccionarios)

Implementan la interfaz map

Parten del concepto indice al regirse y moverse con la relacion “CLAVE/VALOR”(Para una clave tenemos solamente un valor y viceversa). Hay 3 tipos:

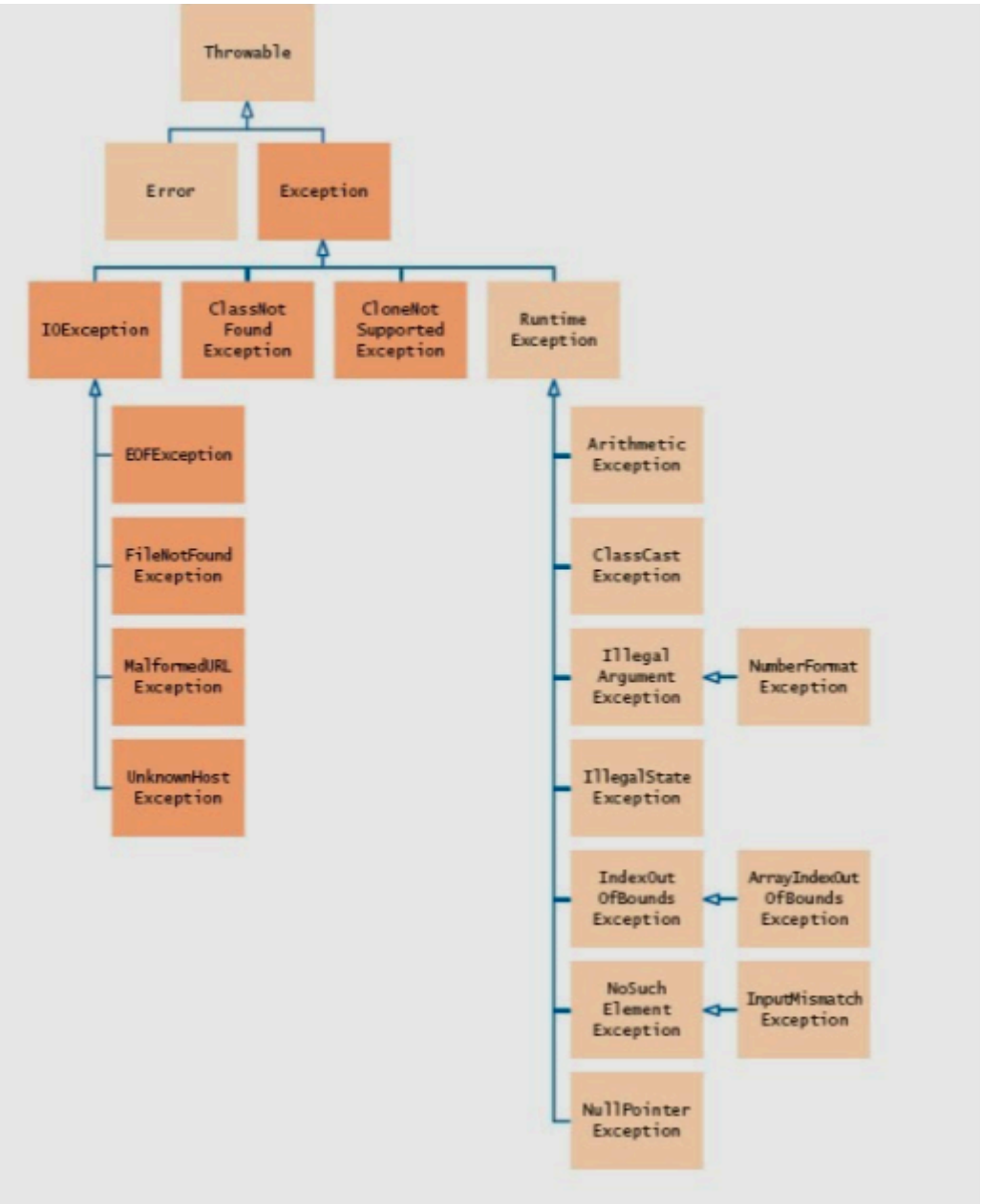
- TreeMap: Usa la estructura arbol de orden natural.
- HashMap: Usa la estrucura hash.
- LinkedHashMAP: Usa la estructura doblemente encadena permitiendo recorrer de abajo arriba o arriba abajo.

La diferencai entre HASH SET y HASH MAP, es que el SET solo permite almacenar de un tipo/clase y el MAP permite de cualquier tipo/clase, y la DIFERENCIA de ellos dos con sus compañeros tree y linkedHash es que en HASH no importa el orden de insercion o clave

EXCEPTIONS

Las excepciones son Objetos, y esos objetos derivan de la clase Throwable y luego de 2 sub clases Error y Exception, a eso lo llamamos jerarquia de

excepciones.



Entonces cuando la JVM encuentra un error durante la ejecucion de su codigo crea un respectivo Objeto exception dependiendo el error y lo lanza. En java cuando una excepcion no es manejada se comienza a lanzar hacia arriba(throw) hasta que alguien la pueda catchear y manejar, de ser caso de que nadie la maneja esa excepcion aborta el programa y muestra el stack trace(camino que hizo la exception y su error).(Tambien las podemos lanzar manualmente con throw en body y throws en header, y hasta relanzarlas)

Para evitar estas excepciones sin manejar podemos prevenirlas con una estrutura asi:

```
try{
    //codigo donde sucede la exception
}catch(ObjetoExceptionACapturar e) {
    //lo que queremos qeu haga si sucede la exception
}finally {
    //este bloque es opcional, lo que este aqui se va a ejecutar si se catcheo la
}
//ademas puede haber varios catch, pero durante la ejecucion del try cuando
```

Ademas existen excepciones checked y unchecked:

- UNCHECKED: Estamos obligados a tratarlas, son todas las exception excepto las runtime
- CHECKED: No estamos obligados a tratarlas, son las de las clases Error y runtime

Java tambien nos permite crear nuestras propias exceptions, al crear una nueva clase siempre y cuando herede de Exception(podemos agregar un super a su constructor con un mensaje).

ARCHIVOS

Java nos permite manejar Archivos con lo que se llama Streams. Los stremas son secuencia de datos y existen de dos tipos:

- ENTRADA: Que representan un origen de datos
- SALIDA: Que representa un destino de datos(si el archivo de salida no existe a la hora de definir el path, el programa lo crea)

Podemos manejar 2 tipos de archivos en java;

DE **TEXTO**:

- BYTE: (cuando llegan al final del archivo arrojan un -1)

Byte streams

Los *byte streams* están enfocados en E/S de bytes de 8 bits

Representan E/S de bajo nivel, las más primitivas de todas y sobre las cuales se basan E/S más complejas

Todas las clases de *byte streams* heredan de `InputStream` y `OutputStream`

Para E/S de archivos, utilizaremos:

`FileInputStream`	`int read() throws IOException`
`FileOutputStream`	`void write(int) throws IOException`

- CHAR: (cuando llegan al final del archivo arrojan un null)

Character streams

Los *character streams* están enfocados en E/S de chars de 16 bits

Utilizan a los *byte streams* para realizar la E/S física, para luego traducir los bytes a caracteres

Todas las clases de *character streams* heredan de `Reader` y `Writer`

Para E/S de archivos, utilizaremos:

`FileReader`	`int read() throws IOException`
`FileWriter`	`void write(int) throws IOException`

De estos ultimos 2 tambien existen sus versiones BUFFERED que utilizan el buiffer para optimizar estas operaciones de entrada y salida que al no estar buffereadas se le delegan al s.o haciendolas costosas y poco eficientes, al bufferearlas mejoramos su eficiencia y permitimos realizar entradas y salidas de a 8192 bytes a la vez.(Para realizar esto cambiamos la palabra file por la palabra buffered)(las buffered implementan sus versiones no buffered y heredan de la principal).

- CSV: Al utilizar los buffered podemos comenzar a hacer entradas y salidas por lineas, y si las hacemos con comas(",") separando las palabras/oraciones entonces tenemos archivos csv(comma separated value).(luego podemos usar un .split(",") y conseguir las palabras separadas)

O **BINARIOS**:

- BINARIO PURO:

Data streams

Los *data streams* soportan E/S binaria de datos primitivos y Strings

Todos implementan las interfaces `DataInput` y `DataOutput`

Para E/S binaria, utilizaremos:

`DataInputStream`	`Métodos específicos por cada tipo de dato primitivo y Strings`
`DataOutputStream`	`Métodos específicos por cada tipo de dato primitivo y Strings`

Tener en cuenta que:

- sólo pueden ser creadas como *wrappers* para un stream ya existente
- detectan el final de archivo capturando la excepción `EOFException` en lugar de retornar un valor inválido
- cada *read* debe tener su *write* correspondiente

- OBJETOS:

Object streams

Los *object streams* soportan E/S binaria de objetos

Todos implementan las interfaces `ObjectInput` y `ObjectOutput`, que son subinterfaces de `DataInput` y `DataOutput` respectivamente

Para E/S binaria, utilizaremos:

`ObjectInputStream`	`Object readObject() throws IOException, ClassNotFoundException`
`ObjectOutputStream`	`void writeObject(Object o) throws IOException`

Tener en cuenta que:

- para objetos complejos (compuestos por ejemplo) se procesan completos, dado que hay que reconstruir el objeto original en su totalidad
- un *stream* sólo puede contener una copia de un objeto, pero sí múltiples referencias a él

IMPORTANTE tener en cuenta que TODOS los objetos que esten o vayan a estar en ObjectStream tienen que implementar la **interface serializable**, esto indica que una clase puede ser serializada, osea convertida en una secuencia binaria de bytes que es la que permitira grabarlos en los archivos.

Ademas todos los archivos se pueden añadir en modo append, eso quiere decir que se continua agregando cosas desde el final del archivo, ya que si no lo abrimos de esta manera, se sobreescribiria lo nuevo escrito sobre lo que ya estaba, perdiendo todo lo anteriormente guardado en el archivo, esto lo

hacemos al agregar un “,true” luego del path del archivo cuando creamos el objeto stream.

GENERICS

Hay cierto tipo de clase que contiene datos GENERICOS, el tipo de dato generico es aquel que va a ser aceptado por la clase al ser definido entre <> al momento de su creacion, por lo tanto puede ser cualquiera que el programador desee.

Ejemplo de clase con dato generico:

```
public class clase<T/*Este es el tipo de dato generico*/> {
    private T/*atributo de tipo generico*/ t;

    public T get() {
    return t;
    }

    public void set(T t1) {
    this.t = t1;
    }
//los ultimos 2 son metodos genericos
}
```

Luego en main declarariamos algo asi:

```
//clase<TipoDeDato> tipoDeDatoClase = new clase();
//EJ:
clase<String> stringClase = new clase();
//en este caso todas las T de la clase anterior pasarian a ser un String
```

Aunque tambien se puden declarar mas de un tipo de dato generico hacienco class<T1, T2, ... , Tn>

ACLARACIONES:

- No es posible reemplazar un parámetro de tipo por un tipo primitivo.
- No es posible lanzar ni capturar objetos de clases genéricas.

- No se pueden crear objetos de tipos genéricos