¿Que es la programación Orientada a Objetos?

La **Programación Orientada a Objetos** es un paradigma de la programación, en el que se basa en el concepto de **clases y objetos**. Esto quiere decir que hay que hacer un diseño **Modular**

Modularidad y criterios

Modularidad

Consiste en separar y estructurar el código en fragmentos más simples y reutilizables como lo son las clases y nos permite instanciar en objetos.

Criterios

Los criterios son utilidades para que en el código se...

- facilite la **descomposición de módulos**
- facilite la **combinación de módulos**
- facilite la **comprensibilidad de los módulos** sin conocimiento sobre los otros módulos
- continuo y facilite que cambios pequeños tengan pequeñas consecuencias
- **proteja** y dificulte que los errores de un módulo afecten a otros módulos

Modularidad y reglas

Reglas

A partir de lo anterior mencionado se generan unas reglas a seguir:

- **Correspondencia directa** para que haya un único concepto de módulo
- **Pocas interfaces** para que un módulo se comunique con pocos módulos
- **Interfaces pequeñas** para que la comunicación entre módulos sea mínima
- **Interfaces explícitos** para que la comunicación sea obvia
- **Ocultación de información** puesto que solo se mostrará al público los justo

Principios de la modularidad

El resultado de juntar los criterios y las reglas es que nacen los principios, en este caso de la modularidad:

Principios

- Los módulos deben ser **unidades ligüística**
- El módulo tiene que estar **auto-documentado**
- El módulo tiene un **acceso uniforme** es decir que sus servicios no distinguen almacenamiento de calculo
- El módulo ser **cerrado** para su uso, pero **abierto** para su modificación
- Si hay un conjunto de alternativas, solo un módulo conocerá la lista completa

PERO, ¿QUÉ ES UN OBJETO?

¿Qué es un objeto?

Antes de poder descifrar esta pregunta primero tenemos que definir qué es una clase, porque el objeto es una instancia de esta

Tipo de Dato Abstracto

Abstracción de un concepto que contiene tantos datos como comportamientos

Clase

Una clase tiene la intención de tener un comportamiento. Por ello es un Tipo de Dato Abstracto (TAD - Type of Abstract object). También cabe decir que las clases se organizan en directorios que son paquetes o también llamados package

¿Qué es un objeto?

Finalmente podemos decir que es un objeto:

Objeto

Un objeto es una instancia de una clase, es decir, es una variable a la que apunta a la clase pero definida con unos valores.

Quizás no quede claro...

Creacion de una clase

```
public class Ventana{
  private boolean abierta;
  private double aperturaVentana;
  //Esto seria el constructor de la clase
  public Ventana(boolean abierta, double apertura){
    this.abierta = abierta;
    this.aperturaVentana = apertura;
  }
  public void abrirVentana(double grados){
    if (grados > 0){
    abierta = true;
    aperturaVentana = grados;
  }
  else{
    abierta = false;
    aperturaVentana = grados;
  }}}
```

Ahora vamos a usar la clase Ventana en nuestro main:

```
public static void main(String[]args){
            //Llamada al contructor
            Ventana casa = new Ventana();

            if(casa.getAbierta()){
                  casa.abrirVentana(0);
            }
      }
```

Constructor

Ventana casa = new Ventana();

Esta es la instancia, aquí es donde se genera la magia del objeto, a partir de aquí la variable casa es un objeto. Por tanto un objeto es una variable que puede usar y/o acceder a todas las características de una clase. Todas menos los Main que nos conocemos

Protecciones a tomar

protecciones

Para que no le pase nada a nuestra la clase, tenemos que declarar nuestros atributos de la clase con p̈rivate d̈elante y nos aseguraremos que los datos que se introduzcan sean validos, es decir, no consideraremos que los vayan a implementar bien. Aunque esto es de más adelante...

Todo esto esta muy bien, pero como vamos a saber que es lo que tiene una clase, que puede hacer que métodos tiene. Para ello usaremos javadoc, espera ¿java que?

Javadoc

Antes hemos hablado de que uno de los principios debía ser que estubieran autodocumentados, ¿no?

javadoc

Javadoc es una herramienta de java, para ayudarnos a documentar los usos que tiene cada método que forman las clases o que parámetros hacen falta para invocar a una función, esto facilita nuestro trabajo y el de nuestros compañeros.

Un ejemplo con la clase Ventana sería:

```
/**
 * Contructor de la clase Ventana
 *
 * @param abierta, true si esta abierta false lo contr
 * @param apertura, grados de apertura si esta abierta
 **/
public Ventana(boolean abierta, double apertura){
  this.abierta = abierta;
  this.aperturaVentana = apertura;
}
```

```
/**
 * Abre la ventana tantos grados y si es 0 se cierra
 *
 * @param grados, grados que se va a abrir la ventana
 **/
public void abrirVentana(double grados){
  if (grados > 0){
  abierta = true;
  aperturaVentana = grados;
}
else{
  abierta = false;
  aperturaVentana = grados;
}}}
```

¿Sois observadores?

Antes de empezar con los ejericios básicos, os pregunto ¿como de observadores sois? ¿Veis lo que no está?

Exacto lo veis, veis que no está en las declaraciones el **static**, esa palabra que ponias en FProg. Bien ahora sabreis porque la poniais y para que se usa.

Static

En java de normal usais public **static** void metodo(){}

Static

El static es un "comando" para hacer una declaración estática, es decir que no va a poder variarse en diferentes cosas si no que solo tendrá un único valor y si este es modificado se modificaría en todos.

ejemplo

El numero de banquetas que tenemos en las aulas, son siempre 6, aunque sean de diferentes aulas, si modificásemos esto, y dijésemos que solo haya 5 banquetas, de repente todas las banquetas de las aulas perderían 1 asiento, y alguno se llevaría un culetazo.

Ejercicios

1º Ejercicio

Generar una clase pistola, con los siguientes atributos:

- modelo o tipo
- balas
- recamara

Y hacer los siguientes métodos:

- Dar el tipo
- Dar cuantas balas hay o quedan
- Meter bala en la recamara
- Disparar arma

Por si acaso el arma no puede disparar sin tener una bala en la recamara

Ejercicios

2º Ejercicio

Hacer que todas las armas sean el mismo **modelo o tipo** siempre y hacer el javadoc del ejercicio anterior

Hasta aquí

Bueno hasta aquí hemos llegado con la presentación sobre programación orientada a objetos

¿O no? Antes de nada, ¿que pasaría si al arma le meto -5 balas? O si diparase balas en negativo?, ¿curarían? Por desgracia, no. Pero aquí seguimos con la siguiente parte

Los TEST

Pues sí aquí también hay test, pero no para examinaros a vosotros, sino para examinar vuestro código...

Pruebas de Programas

Son pruebas para comprobar el funcionamiento del programa, buscar errores en ellos, forzarlos y comprobar el buen funcionamiento del mismo

Errores

Bueno, ahora sabemos que hay que hacer "exámenes" a nuestro código, pero a diferencia de a nosotros, no les vamos a suspender, solo vamos a corregirlos y mejorarlos. Pero para ello, tendremos que saber sobre los errores que nos podemos encontrar o al menos los más habituales.

Errores hábituales

IllegalArgumentException $\rightarrow$ Error en el argumento
IllegalStateException $\rightarrow$ Error de estado
NullPointerException $\rightarrow$ Error por uso de un puntero a null.

Clase throwable

Bueno habeis visto que son poquitos los errores que vamos a tratar nosotros comparados con todos los del mercado, no? Antes de seguir tenemos que hablar de lo que encabeza el árbol de los tipos de errores.

Clase Throwable

Clase base que representa todo lo que se puede "lanzar´´ en java.

- Contiene una instantánea del estado de la pila en el momento que se creo el objeto.
- Almacena un mensaje que se puede usar para dar detalles del error que se ha producido.

Ahora sí seguimos con los errores y como tratarlos

Trato de Errores

Lo más normal es dejar tratada la excepción que pueden ocurrir y añadirlo también al javadoc, es decir:

Tratamiento de Errores

Nosotros principalmente usaremos la sentencia Throw que se utiliza para lanzar objetos de tipo Throwable:
throw new Exception (“Mensaje de error”);

A partir de aquí pueden ocurrir 4 casos que realmente vienen a ser 1, como la santísima trinidad.

Tratamiento de errores

Casos

Bueno los casos que van a ir en orden y el primero siempre se va a cumplir:

1. Se sale inmediatamente del bloque de código natural
2. Si el bloque tiene asociada una clausula **catch**{} adecuada para el tipo, entonces se ejecuta el catch
3. Sino sale del bloque y busca un catch adecuado
4. Si no encuentra nada, el proceso de ejecución finaliza con un error. El error que nosotros hayamos puesto.

ErROes

Muy bien ahora sabemos que errores se pueden producir y como afrontarlos por lo menos los que nos vamos a encontrar en esta asignatura, pero, ¿cómo y cuándo van a aparecer?

Es hora de empezar a realizar pruebas para encontrar los fallos de nuestro programa... Mmmmm curiosa palabra, creo que es la primera vez que la mencionamos por aquí

Fallos y Errores

Fallos

Un Fallo es cualquier situación inesperada que se produce durante la ejecución de un programa

error

Un error es cualquier elemento de software que es incorrecto desde un punto de vista sintáctico o semántico

Es decir un fallo viene de un error, pero aún así mantendremos que buscamos fallos, puesto que si hay un fallo, hay un **ERROR**

Clases de Equivalencias

Una cosa clara, las pruebas no sirven para quitar los errores, solo para demostrar que están ahí y que somos conscientes de ellos, y que son defectos del software. Ahora:

Clases de Equivalencia

Es un conjunto de datos de entrada para el mismo de los que podemos esperar un comportamiento homogéneo, estás pueden ser válidas o no válidas.

Clases de equivalencias

Ahora ¿cómo diferenciamos las clases?

División de Clases

La división en clases de equivalencias se realiza en función de las condiciones de entrada de los datos que nosotros le metamos.

Es decir, en un programa que nos dé la hora, que tengamos 3 entradas, horas, minutos y segundos, no serán los mismos datos que metamos en las horas que en los minutos ni que en los segundos.

Clases de equivalencias

Es decir que tenemos de condiciones de entrada:

Condiciones de entrada

- Un rango de valores $\rightarrow$ Un valor por encima y otro por debajo
- Una cantidad de valores $\rightarrow$ Un valor valido y 2 que no lo sean
- Un valor entre un conjunto de valores $\rightarrow$ Tantas clases válidas como valores posibles y una no valida
- Un dato debe ser $\rightarrow$ Un dato válido y otro no (También entra que si tiene que ser nulo ;-)

Tabla de clases de equivalencia

Es decir se nos queda con una tabla con Condición de Entrada (CE), otra con Clases Válidas (CV) y otra con Clases No Válidas (CNV)

La tabla de las clases de equivalencias

CE	CV	CVN
Condición 1	Clase 1.1	Clase 1.2 Clase 1.3
Condición 2	Clase 2.1	Clase 2.2
Condición 3	Clase 3.1	Clase 3.2

Bueno ahora que tenemos algo para clasificar las pruebas, podemos realizarlas:

Tabla de Casos de Prueba

Se harán de 3 columnas, en la primera indicaremos los Casos de prueba que haremos, en la segunda daremos las clases de equivalencia cubiertas y por último el resultado esperado

Tabla de Casos de Prueba

Caso de prueba	Clases de equivalencia cubiertas	Resultado esperado

Ejericicio 3

3º ejercicio

Hacer la clase de equivalencia de la pistola

Qué pensabas que iba a ser original y mostraros un código para que me digáis la clase de equivalencias, sí claro, y ya a parte os pongo mi propio trabajo para que me lo digáis y me lo hagáis. XD

AUNQUE PUEDE, nada era broma hasta aquí el taller, **DUDAS**, no? Si no mi @ de telegram es @cadore33 y si no mi hogar es el GUI esa salita (ErZuloMiAIrma) que hay nada más subir las escaleras.