PROGRAMACIÓN

Unidad 4. Programación Orientada a Objetos

1. OBJETOS

Definición: toda entidad relevante en un sistema software que tiene un estado y un comportamiento, con un ciclo de vida.

Estado: toda información relevante de un objeto, que encapsulamos dentro del objeto, conjunto de atributos para almacenar información.

persona
- nombre
- edad
- altura

identidad
información

Comportamiento: conjunto de acciones que puede hacer un objeto

funciones + procedimientos = métodos
- corre
- come
- estudia

Método constructor
Método destructor

Reglas:

- **Abstracción**
- **Ocultación**
- **Encapsulamiento**
- **Herencia:** todos los objetos heredan de algo un estado o un comportamiento.
    - **Simple:** un solo ancestro
    - **Múltiple:** varios ancestros.
- **Polimorfismo:** un objeto polimórfico, es un objeto que puede ser varias cosas a la vez.

(Los objetos se comunican a través de mensajes)

2. CLASES

Definición: mecanismo para definir la estructura de un objeto y el comportamiento del mismo. Se establece el ID del objeto

Propiedades
Atributos
Funciones
Procedimientos

3. MEMORIA

Funcionamiento de la memoria:

- **Stack:** memoria estática
- **Heap:** memoria dinámica

Destrucción de objetos: en ciertos lenguajes de programación se hace de forma automática, como en java con el recolector de objetos, pero hay otros lenguajes de programación donde le tendremos que crear nosotros.

4. PUBLIC & PRIVATE

En los diagramas de clases los métodos públicos se representan con un + y los métodos privados se representan con un –
Sí no queremos que aparezca una variable indicaremos que es private *(private int antigüedad).*
El estado de los objetos va a ser privados, en ese caso se accede a través de su comportamiento, las propiedades deben ser privadas salvo causa justificada-

5. GETTER & SETTER

Getter → da visibilidad.

Setter → permiten modificar.

.this → apunta a mí mismo (cuando tenemos conflicto de nombre).

Consejo: en IntelliJ para tardar menos podemos hacer los siguiente: code > generate > getter and setter

6. NULL

Definición: null es un valor, por lo tanto, no está vacío, pero no almacena un dato.

El null está ahí y nos lo vamos a comer así que podemos lanzar una excepción al ver el nulo y meterlo en un try cath, o la segunda opción es hacer un is donde imprimamos las cosas si no hay un null, y si lo hay tendremos que dar un mensaje de que no existe. A==null / a!=null.

En distintos lenguajes puedes elegir si permites almacenar o no un nulo, como kotlin.

7. MÉTODO CONSTRUCTOR

Método constructor → new Persona ();

reserva constructor

8. SOBRECARGA

Definición: método con el mismo nombre, pero con distintos parámetros. Se produce el problema cuando el mismo número de parámetros y el mismo tipo son iguales.

Se soluciona en el proceso de compilación, de forma estática.

Sobrecarga de constructores: se puede usar el constructor vacío, cuando no implementamos ningún otro constructor.

Recolector de basura en Java: System.gc(); Se ejecuta automáticamente, pero también lo podemos ejecutar nosotros.

9. IDÉNTICOS E IGUALES

Objeto idénticos != iguales

Equals, no se puede utilizar para comparar si dos objetos cumplen en igualdad.

10. COMO ENLAZAR POJO A LA CLASE

Eso se llama **navegabilidad** y se representa con flechas de una dirección entre clase y objeto, en una o dos direcciones.

Dependencia: Por ejemplo, para encontrar un objeto primero tendremos que saber a qué clase pertenece, como si buscas un animal, pero lo primero que tienes que encontrar es la especie (especie "clase perro" y entonces objeto "dálmata"). Dentro de la dependencia existen 2: dependerá de si existen o no al borrar la clase.

- **Agregación**: existen una relación entre la clase y el objeto, pero cada uno de los objetos pueden aparecer de manera independiente (un equipo de fútbol tiene distintos jugadores, pero estos pueden cambiar de equipo y si el quipo se deshace los jugadores siguen existiendo (se representa con un rombo blanco).
- **Composición**: no puede existir ningún objeto sin esa clase y si se borrara la clase borramos todos sus objetos (si quemo un colegio también se queman las clases).

11. CRUD

(Crear, leer, actualizar, borrar) (create, read, update, delete)

Crear

Tenemos que hacer un for para que busque el primer null que tenga e insertar ahí el alumno, siempre buscaremos el alumno null para que no sobrescriban un alumno ya hecho, y ponemos con un if que si la posición es mayor o igual a 0 y menos a el largo máximo entonces lo creamos.

Para saber cuántos alumnos tengo creados iré añadiendo +1 a una propiedad que añadiremos que se llamará alumnosCreados.

Leer (Todos o uno)

Todos:

Para ello hacemos un por que valla imprimiendo cada uno de los alumnos y que si encuentra un null lo ignore, por lo que solo sacará los alumnos creados, aquí también comprobaremos si tenemos o no alumnos viendo la propiedad antes creada de número de alumnos.

Uno:

También hacemos una función que lea solo uno dando el número de alumno que queremos leer, y tendremos que hacer un if else para ver si el número está entre el largo del vector, y si no damos un mensaje y otro if que mire si la posición que estamos pidiendo está llena con un alumno o si está vacía y por lo tanto es null, si es null daremos un mensaje o una excepción para saber que no existe.

Si lo ponemos con excepciones, para que no nos casque el programa podemos hacer fuero un try cath que nos las coja todas y nos dé un mensaje de fallo.

Actualizar

Para ello hacemos una clase.

Borrar

Hacemos una función que dado un alumno que queremos borrar introduzcamos un null y con eso lo tenemos hecho.

12. COMPARE TO

____.CompareTo()____ = 0 son iguales
-1 es menor
+1 es mayor

13. MÉTODOS Y PROPIEDADES DE CLASE:

Propiedades y métodos que son gestionados por la clase:

- Se pueden utilizar sin instanciar al objeto.
- Solo pueden utilizar propiedades y métodos de clase.
- Java los llama Static
  Persona p = new Persona();
  p.caminar();
  Persona.X

14. PATRONES DE DISEÑO

- Sirve para cualquier lenguaje.
- Es un lenguaje común.
- Organización de los patrones:
  - **Creacionales**
    - **Singleton:** nos asegura que una clase solo tenga una instancia.
      Eliminamos el constructor
      LocalDataTime instante = LocalDataTime.now();
      private static Sala = null;
      private Sala () {}
      public static Sala getInstancia(){
      if (instancia == null){
      instancia = new Sala()
      }
      return instancia;
      }
    - **Builder:** creamos distintos tipos usando el mismo código de construcción.
  - **Estructurales**:
    - Facade (fachada): cuando una clase queremos que sea la de comunicación, punto de entrada.
  - **Comportamiento**