1. **Momentos de Verificación (Static y Dynamic):**
    - **Verificación Estática:** Este término se refiere al proceso de revisar y evaluar el código de un programa sin ejecutarlo realmente. Se realiza antes de que el programa se ejecute y se centra en comprobar la estructura y las propiedades del código fuente. Los lenguajes de programación estáticos, como C++ o Java, utilizan este enfoque. Los errores se detectan antes de la ejecución, lo que puede ayudar a prevenir muchos problemas comunes.
    - **Verificación Dinámica:** Este término se refiere al proceso de evaluar el comportamiento de un programa mientras se ejecuta. Implica probar el programa con entradas específicas para observar su comportamiento en tiempo de ejecución. Los errores dinámicos, como excepciones o problemas de lógica, se detectan durante la ejecución del programa. Lenguajes como Python o JavaScript utilizan este enfoque.
2. **Declaratividad de Tipos (Explicito e Implicito):**
    - **Declaración Explícita de Tipos:** En este enfoque, el programador debe especificar el tipo de datos de una variable de manera clara y directa al declararla. Por ejemplo, en muchos lenguajes estáticos como C++ o Java, debes declarar el tipo de una variable antes de usarla. Esto ayuda a prevenir errores relacionados con tipos de datos, pero puede requerir más código.
    - **Declaración Implícita de Tipos:** En este enfoque, el lenguaje de programación infiere automáticamente el tipo de datos de una variable según el contexto y el valor con el que se inicializa. Lenguajes como Python o JavaScript a menudo utilizan esta técnica, lo que puede hacer que el código sea más conciso y legible, pero también puede llevar a problemas si no se comprende correctamente el tipo inferido.
3. **Identificación de Tipos (Nominal y Estructural):**
    - **Identificación Nominal de Tipos:** En este enfoque, la comparación de tipos se basa en la declaración nominal o el nombre del tipo. Dos tipos son considerados iguales si tienen el mismo nombre de tipo, independientemente de su estructura interna. Esto significa que incluso si dos tipos tienen la misma estructura, si tienen nombres diferentes, se consideran diferentes. Este enfoque se usa en muchos lenguajes estáticos.
    - **Identificación Estructural de Tipos:** En este enfoque, la comparación de tipos se basa en la estructura interna de los tipos, es decir, en sus campos y métodos. Dos tipos se consideran iguales si tienen la misma estructura, aunque sus nombres sean diferentes. Este enfoque se utiliza en algunos lenguajes de

> programación, como TypeScript o algunos sistemas de tipado en lenguajes funcionales.

Clase 3

Memoria, que es?

- Es hardaware
- Programa
- Datos del programa

CPU es quien se encarga, busca en memoria, decodifica y ejecuta.

Modele de von Neumann (Computadoras de hogar se usa este modelo.

CPU <-> Memoria de programa y datos

Modelo de Harvard

Memoria de programa <-> CPU <-> Memoria de datos

La memoria se organiza en celdas. Los datos ocupan una celda. Tambien instrucciones.

Quien se encarga de administrar la memoria es el sistema operativo. Este organiza la celda en bloques o sectores de memoria.

Cuando se crea el proceso

- Carga el programa en memoria RAM.
- Datos del programa (Chica y fija) – Stack es algo fijo.

El sistema operativo le da mas memoria, cuando el programa la pide. Por defecto siempre da una cantidad de memoria predetermina, dependiendo el sistema operativo. Le puede pedir mucha manera y esta se puede ir extendiendo a lo largo de su ejecución. La heap es algo que se puede ir extendiendo.

Donde se guardan las cosas en la programación oriendta a objetos?

Var firulais := new Perro()

Los objetos viven en la Heap.

Las variables locales (referencias), como por ejemplo virulais viven en el stack.

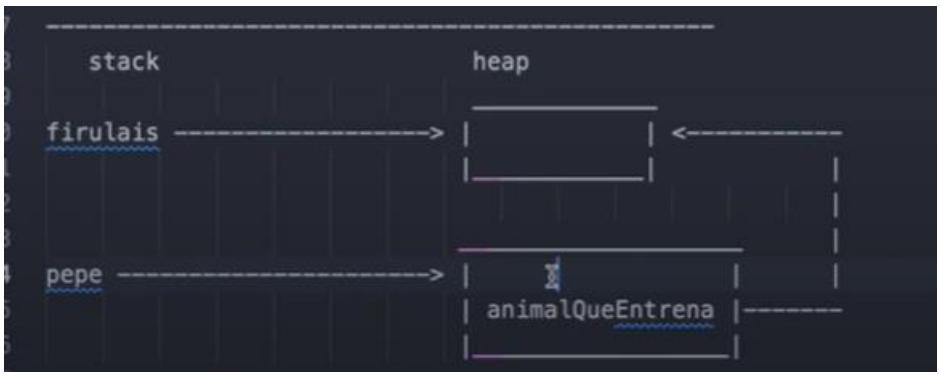
Var firulais := new Perro()

Var pepe := new Entrenador()

Pepe.animalQueEntrena = firulais

Stack: firulais y pepe

Head: Perro y Entrenador (animal que entrena).



Un puntero es un numerito que representa una porción de memoria.

Que pasa si hacemos pepe = null.?

El sistema operativo no borra el objeto guardado en stack.

Para eliminar un objeto, que ya nos referenciado hay que hacer free(), para que libere memoria.

Heap Overflow: es cuando le pedis mucha memoria, y el sistema operativo ya no tiene para darte.

Algunos lenguajes tienen constructos y destructo, un claro ejemplo es C++.

Algunos lenguajes como java tiene recolectores de basura, que hacen esta tarea de manera automática.