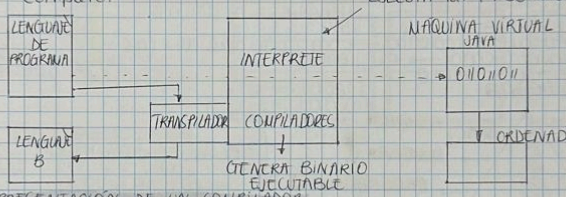
METODOLOGÍA DE LA PROGRAMACIÓN

EXÁMENES
PEI1: 12 Marzo → 15%
PEI2: 14 Mayo → 15%
PRÁCTICAS
EL1: Febrero final → 10%
EL2: Marzo " → 10%
EL3: Abril " → 10%
PEF: 14 Mayo → 40%

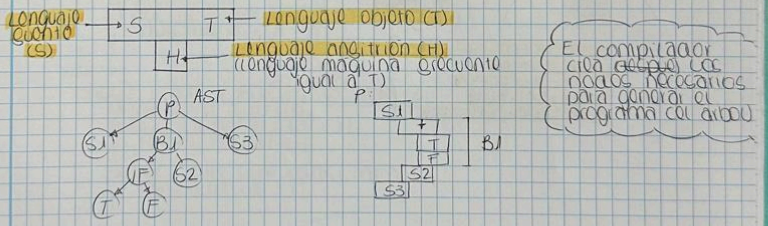
LENGUAJE FORMAL: Lenguaje cuyos símbolos primitivos y reglas para unir esos símbolos (cuaderno) están completamente especificados.
CONTEXTO: Ventana de trabajo para poder entenderse.
TIPOS DE LENGUAJE:
- L. Máquina: Lenguaje de más bajo nivel. Binario. (Los primeros)
- L. Ensamblador: Bajo nivel ya que están muy relacionados con la máquina versión simbólica de los lenguaje máquina (ADD). Son de segunda generación. Operaciones básicas. No apenas valor.
- L. Alto nivel: Estructuras de control, variables de tipo, reutilizadas. Primera lenguajes de programación. Fácil velocidad y flexibilidad a la hora de crear un programa. 3ª gener.
- L. Orientados a los problemas: describen la solución, no como conseguirlo 4ª generación.

NECESIDAD DE NUEVOS LENGUAJES ¿? Si los problemas evolucionan, es natural que los lenguaje para resolverlos también, e incluso aparezcan nuevos. Gran parte de los lenguajes no sobreviven mucho tiempo.

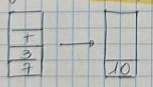
TRADUCTORES, INTERPRETES Y COMPILADORES
Traductor: Un traductor es un programa que lee un programa escrito en lenguaje ~~fuente~~ fuente de alto nivel, y lo traduce en un lenguaje objeto
Según las funciones puede ser dos tipos:
- Interprete: Ejecuta directamente lo que traduce, sin almacenar en disco la traducción realizada
- Compilador: Genera un fichero del lenguaje objeto ejecutable que puede posteriormente ser ~~ejecutado~~ ejecutado tantas veces se necesite sin volver a traducir. Como añadido adicional, el compilador informa de los errores del programa fuente ya que lo analiza al completo.



REPRESENTACIÓN DE UN COMPILADOR



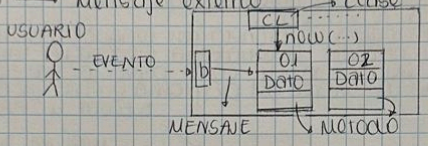
ENTORNOS DE EJECUCIÓN
- Según implementación
  - En metal (maquina física)
  - Maquina virtual
  - En maquina pila
  - En maquina de registros (Von Neumann) (más flexible para más cosas de implem)
- Según tipo de ejecución



TIPOS DE PROGRAMACIÓN

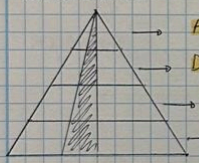
- Programación funcional
- Programación orientada a objetos
  - Abstracción: método de reutilización (utilizas listas, bibliotecas... ya hechas para la creación de tu programa sin necesitar los detalles de lo que reutilizas)
  - Encapsulamiento: No quedan expuestos los detalles de tu programa solo se muestran puntos específicos y controlados → PRINCIPIO DE OCULTACIÓN
  - Modularidad: Los objetos para funcionar deben colaborar entre sí. Dependiendo de lo que necesita en cada momento colaborarán distintas clases, distintos objetos, creando un módulo. Trabajamos como los distintos objetos colaboran entre sí.
  - Herencia: En el momento del diseño debemos como se relacionan las clases que voy construyendo (herramienta estática)
  - Polimorfismo: Dadas dos clases que se llaman igual y que aparentemente tienen la misma funcionalidad pero (¡ojo) tienen distintos comportamientos.
  - Recolección de basura: C

Clase → Implementación
Objetos → Implementación
Métodos → Funcionalidad de la clase
Mensaje → Invocación de un método
Evento → Mensaje externo



Atributo / Propiedad → Dato variable de clase
Identificador → Referencia
Estado interno

DIAGRAMAS



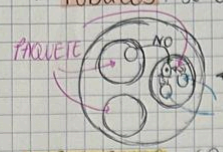
- ANÁLISIS
- DISEÑO
- PROGRAMACIÓN
- PRUEBAS

} Establecimiento de Registros

- D. de casos de uso
- D. de clases
- D. de secuencia
- D. de estados
- D. de componentes

} Diagramas básicos

- **PAQUETES**
  - Privados: se oculta el interior (atributo)
  - Protegidos: solo determinadas clases pueden acceder
  - Públicos: se oculta el exterior (atributo)



Para las clases de un mismo paquete el atributo es público pero para el resto de clases de otros paquetes es privado.

- **ANOTACIONES**: cómo quieres que se comporte el lenguaje
  @override
  public String to String
  ⊛ Las notaciones no las declaramos nosotros, vienen dadas. Sirve para comunicar al programa que estás utilizando un método que ya está definido. Estás sobrecargando un método o estás cambiando el comportamiento de un objeto)
- **HERENCIA**
  - Clases abstractas

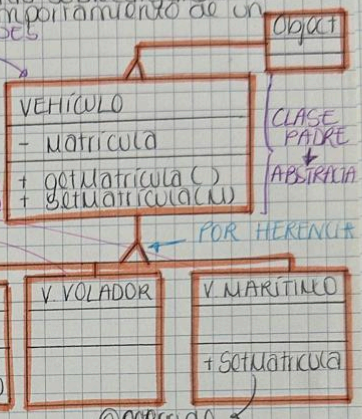
NOMBRE DE LA CLASE
- Dato 1
Dato 2
+ Dato 3
- Método 1
Método 2
+ Método 3

PRIVADO (-) PROTEGIDO (#) PUBLICO (+)

SOLO pueden acceder a los atrib (+) (#)

NUNCA PUEDES INSTANCIARLO (NEW)

Ej:

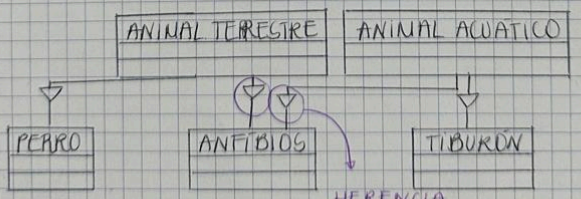


EN PROGRAMACIÓN:

ABSTRACT

```
pubuc class Vehículo{
    private String matrícula;
    pubuc String getMatrícula(){
      return matrícula;
    }
    pubuc void setMatrícula(String m){
      this.matrícula = m;
    }
}
```

```
pubuc class V.Ruedas extends Vehículo{
    private int n_ruedas;
    pubuc int getRuedas(){
      return n_ruedas;
    }
    pubuc void setRuedas(int n){
      this.n_ruedas = n;
    }
}
```



- **INTERFACES**

EN JAVA SE IMPLEMENTA:

```
interface terrestre{
    public getEdad();
}
interface acuatico{
    public get...();
}
```

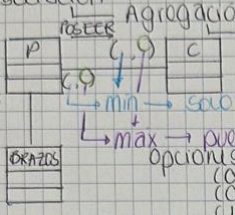
```
class anfibio implements terrestre, acuatico{
    public getEdad(){
    ==
    ==
    }
}
```

terrestre t1 = new Anfibio → Las interfaces pueden declararse para una variable

interface: pichard

MULTIPLICIDAD Y CARDINALIDAD

Asociación



Agregación — composición

"0": Representa la ausencia de relación

→ min → solo puede ser "0" "1"

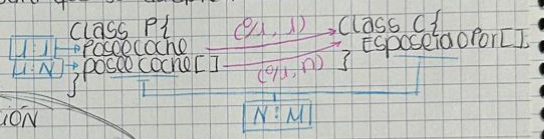
→ max → puede tener valores "1", "n"

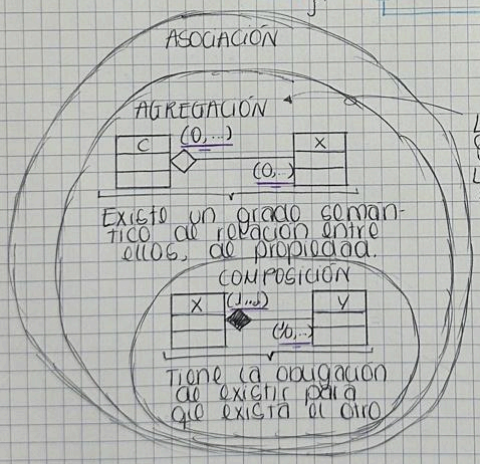
} Estas estableciendo las condiciones de tu programa

opciones: (con el de coches)
(0,1) ⟼ min 0 coches max 1
(0,n) ⟼ min 0 coches max n
(1,1) ⟼ min 1 coche max 1 → solo 1 coche
(1,n) ⟼ min 1 coche max n

↳ Dependiendo del programa que estes realizando y para que lo vas a implementar deberas utilizar unos u otros para que se adapten a tus condiciones.

Categorías:
- 0:0
- 1:N
- N:M





Existe un grado semántico de relación entre ellos. de propiedad.

Tiene la obligación de existir para que exista el otro

La clase padre posee gran parte de las características hijo La diferencia es semántica

MODELO EXAMEN

Mirar diferencia entre int e integer. Y entre float y double.
Mirar las preguntas sueltas de los diagramas UML

PREGUNTA 4: Genera el código

```
Abstract class ventana{
} (protected) void mover(){}
}

(Public) class ventanuco extends ventana{
Integer  get dimension;
    @Override
    void mover(){}
} void cerrar(){}

Class Manija{
    integer tamano;
    void isCerrable(){}
}

Class Visagra{
    integer num_tornillos;
}
```

Manija compone = new Manija();
Lista<Visagra> sujeta = new Lista<Visagra>();

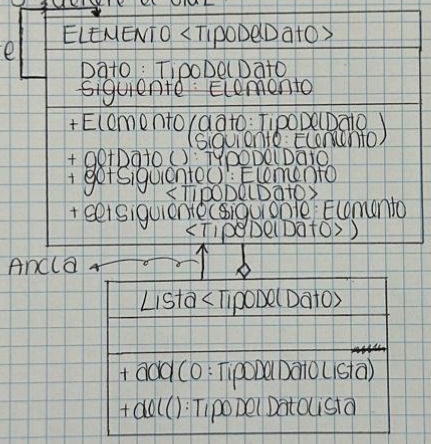
Si te pidieran hacerlo bien sería:

```
Class Manija{
private integer tamano
public boolean isCerrable(){}
public Manija(integer tamano){
  this.tamano = tamano
}
public Manija(){
  this.tamano = 1
}
```

¡TENER CUIDADO CON LOS NOMBRES DE LAS CLASES PORQUE PUEDEN NO SER LO QUE SU NOMBRE INDICA!

¡UN PROBABLE QUE ENTRE EN EL EXAMEN

PREGUNTA 5: Genera el UML:



¡Mirar ejercicios donde se pregunta la causa de un programa y cómo explicar pf!