Estudiante: Esteban Chinchilla Brenes

ID: 116760448

Correo: esteban.chinchilla.brenes@est.una.ac.cr

Grupo: 2-6:00 P.M

---

**Ejercicio de autoestudio#1**

**Enunciado:**

Inventario de Conceptos Clave

1. Diagrama de Venn L-C-P (L: Leguaje, P:Paradigma, C:compilador/traductor)

2. Paradigma en filosofía de la ciencia (Thomas Kuhn)

3. Realidad Objetiva versus Subjetiva. Positivismo

4. Paradigma de Programación

5. OOP: Sustantivos primero. Verbo supeditado a sustantivo (conducta del sustantivo)

6. FP: Verbo primero. Pueden existir sin sustantivo

7. LP: Relaciones primero. Pueden existir sin sustantivo

Ejercicios

No recurra a la IA sin intentarlo sin esta (recuerde en los exámenes no hay IA, solo NA (natural intelligence)

En general, a menos que se diga lo contario los ejercios de autoestudio no son para entregar. Es posible que se pregunten cosas no vistas aún.

Haga una colección para su propio estudio

0) Esté en capacidad de explicar cada concepto clave del inventario. Haga un glosario

1) "La boda": Suponga que se está planificando una gran boda y hay distintos tipos de personas involucradas: novios, padres, padrinos, invitados, fotógrafos, músicos, cocineros y potencialmente otros más. Y hay tambien muchas funciones o roles: quién baila el primer vals, quién da el discurso, quién corta el pastel, quién firma como testigo, etc. hay relaciones estáticas (se conocen en tiempo de compilación).La relación entre personas y

roles no es estática (es dinámica) y no es funcional en ninguna dirección. Su solución tiene casos de prueba separados del modelo y compila y corre desde una consola. Ningún rol es completamente abstracto.

a) Modele en OOP en Java.

b) Modele en JS pero sin usar class ni ninguna librería externa.

2) En modelamiento algebraico de datos (abstract data types, ADTs) hay datos puros que no tiene métodos de lógica especial (a lo más getters y formas de serialización como hileras, alis toString). Son inmutables (no se puede cambiar su estado, solo inicializarlo durante creación). En Java a ese tipo de dato se le dice un POJO (plain old data object). Su solución tiene casos de prueba separados del modelo y compila y corre desde una consola. Restricción: no usar 'instanceOf'

a) Modele los números naturales como ADTs.

b) Modele la suma de naturales sin usar la suma primitiva durante la operación. Añada un toString que permita ver al nat como número en decimal

3)[Opcional, Reto] ¿Cómo se denomina el patrón en el que se utiliza un atributo interno para distinguir subtipos de un objeto sin recurrir a herencia? Por ejemplo, al agregar un campo género (con un enum con valore Male, Female) al objeto Persona para tipificarlo. ¿Se considera esta práctica un antipatrón, y bajo qué circunstancias podría volverse problemática?

---

**SOLUCION**

0- **Glosario de conceptos**
    a. **Diagrama de Venn L-C-P (L: Leguaje, P:Paradigma, C:compilador/traductor)**

**Lenguaje:** Sistema estructurado compuesto por un conjunto de símbolos (caracteres) y reglas sintácticas y semánticas, utilizado para expresar instrucciones o facilitar la comunicación entre humanos y computadoras. En informática, los lenguajes de programación permiten escribir algoritmos que una máquina puede interpretar o ejecutar.

**Paradigma:** Modelo o enfoque que define una manera específica de resolver problemas computacionales, basado en principios teóricos y estructuras lógicas. Ejemplos comunes

incluyen la programación orientada a objetos, funcional y procedimental. Cada paradigma ofrece herramientas y metodologías distintas para abordar la construcción de software.

**Compilador/traductor:** Programa que convierte código fuente escrito en un lenguaje de alto nivel (como C# o Java) en código máquina o en otro lenguaje intermedio comprensible por la computadora. El compilador realiza esta conversión antes de la ejecución del programa, mientras que un intérprete lo traduce en tiempo real.

**b. Paradigma en filosofía de la ciencia (Thomas Kuhn)**

No es solo una teoría o un modelo. Piensa en ello como un marco conceptual completo, un par de "gafas invisibles" que usamos sin darnos cuenta para ver el mundo. Estas gafas definen qué problemas son importantes, cómo se deben abordar, qué se considera una solución válida, e incluso qué datos son relevantes. Son tan fundamentales que, cuando estamos dentro de un paradigma, nos cuesta muchísimo ver las cosas de otra manera.

**c. Realidad Objetiva versus Subjetiva. Positivismo**

***Realidad Objetiva:***

Es la existencia de hechos o fenómenos que son independientes de las percepciones, creencias o emociones de las personas. Se considera que una realidad es objetiva cuando puede ser observada, medida o verificada por diferentes observadores de manera consistente. En ciencias de la computación, se asocia con datos empíricos, resultados reproducibles y comportamientos del sistema que no dependen de la interpretación individual.

***Realidad Subjetiva:***

Es la percepción individual y personal que una persona tiene sobre el mundo, influenciada por su experiencia, cultura, emociones o creencias. En tecnología o desarrollo de software, lo subjetivo puede verse reflejado en decisiones de diseño, experiencia de usuario (UX), o interpretación de requerimientos.

***Positivismo:***

Corriente filosófica que sostiene que el conocimiento verdadero se basa únicamente en hechos observables y comprobables empíricamente. En el ámbito científico y tecnológico, el positivismo influye en el método científico, promoviendo la recopilación de datos, pruebas objetivas y la formulación de leyes generales a partir de la observación. En

computación, este enfoque respalda prácticas como la validación de software, pruebas automatizadas y el uso de métricas cuantificables.

**d. Paradigma de Programación**

Es un enfoque o estilo fundamental para escribir y estructurar programas, que define cómo se organizan las instrucciones y cómo se resuelven los problemas mediante el código. Cada paradigma proporciona un conjunto de principios, modelos y estructuras para desarrollar software.

Los principales paradigmas de programación incluyen:

***Imperativo***: Se basa en instrucciones secuenciales que modifican el estado del programa. Ej.: C, Python (en su forma básica).

***Orientado a Objetos:*** Organiza el código en clases y objetos que encapsulan datos y comportamientos. Ej.: Java, C#.

***Funcional:*** Enfatiza el uso de funciones puras y evita el estado mutable. Ej.: Haskell, Elixir, partes de JavaScript.

***Lógico:*** Se basa en reglas y hechos para inferir soluciones mediante lógica formal. Ej.: Prolog.

**e. OOP: Sustantivos primero. Verbo supeditado a sustantivo (conducta del sustantivo)**

Paradigma de programación que organiza el software en torno a objetos, que representan entidades del mundo real o conceptual. Cada objeto combina datos (atributos) y comportamientos (métodos), y la lógica del programa gira en torno a la interacción entre estos objetos.

Una forma de entender OOP es considerar que los sustantivos vienen primero: se modelan los objetos relevantes del problema, y luego se les asignan verbos subordinados a esos

sustantivos, es decir, comportamientos que tienen sentido en el contexto del objeto (por ejemplo, un objeto Perro puede ejecutar ladrar() o comer()).

Principios fundamentales de la OOP:

***Encapsulamiento:*** Oculta la implementación interna y expone solo lo necesario.

***Abstracción:*** Simplifica la complejidad al enfocarse en lo esencial.

***Herencia:*** Permite crear nuevas clases a partir de otras existentes.

***Polimorfismo:*** Permite usar objetos de diferentes clases de forma intercambiable si comparten una interfaz común.

**f. FP: Verbo primero. Pueden existir sin sustantivo**

Programación Funcional (FP):

Paradigma de programación centrado en la definición y composición de funciones puras, donde la lógica del programa se construye a través de la aplicación de funciones a datos, sin depender de estados mutables ni estructuras orientadas a objetos.

En FP, los verbos vienen primero: las funciones son entidades de primera clase que pueden existir y operar de forma independiente, sin estar ligadas a un objeto. Es decir, una función puede existir sin un "sustantivo" (objeto) que la contenga, lo que permite un estilo más declarativo y matemáticamente riguroso.

Características clave:

***Funciones puras:*** Siempre devuelven el mismo resultado para los mismos argumentos y no tienen efectos secundarios.

***Inmutabilidad:*** Los datos no cambian; en su lugar, se crean nuevas estructuras.

***Composición de funciones:*** Se combinan funciones simples para formar otras más complejas.

***Transparencia referencial:*** Una expresión puede ser reemplazada por su valor sin cambiar el comportamiento del programa.

**g. LP: Relaciones primero. Pueden existir sin sustantivo**

Paradigma de programación basado en la lógica formal, donde los programas se describen como un conjunto de hechos, reglas y consultas. En lugar de indicar cómo realizar una tarea paso a paso, se especifica qué condiciones deben cumplirse, y el sistema se encarga de buscar las soluciones mediante inferencia lógica.

En LP, las relaciones vienen primero: se definen conexiones lógicas entre elementos, y no es necesario tener objetos o estructuras concretas que las contengan. El foco está en declarar relaciones entre conceptos y permitir que el motor lógico (como Prolog) resuelva preguntas basándose en esas relaciones.

Características clave:

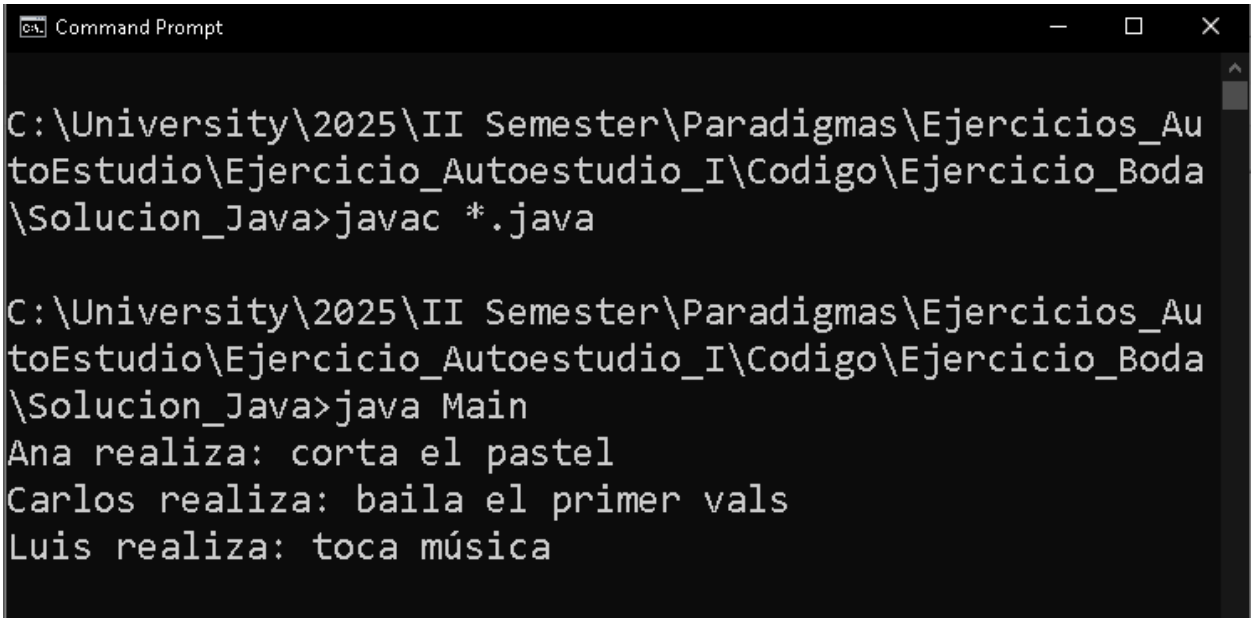
***Hechos:*** Información base del sistema (ej.: es_padre(juan, maria).)

***Reglas:*** Definen relaciones lógicas entre hechos (ej.: es_abuelo(X, Y) :- es_padre(X, Z), es_padre(Z, Y).)

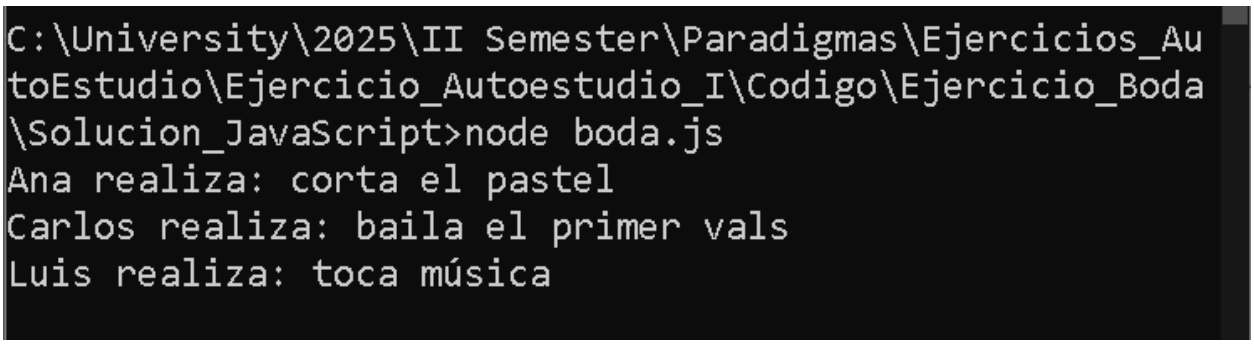
***Consultas:*** Preguntas que el sistema intenta responder a partir de los hechos y reglas.

**1- Ejercicio la boda**

Java:




JavaScript:



2- **ADTs**

a. **Modelar numeros naturales como ADTs**

```java
public class Nat {
    private final int valor;

    public Nat(int valor) {
        if (valor < 0) {
            throw new IllegalArgumentException("Natural debe ser >= 0");
        }
        this.valor = valor;
    }

    public int getValor() {
        return valor;
    }

    @Override
    public String toString() {
        return Integer.toString(valor);
    }
}
```

b. **Modelar suma de naturales sin usar la suma primitiva durante la operacion. Annadir un toString que permite ver al nat como un numero en decimal.**

```
C:\University\2025\II Semester\Paradigmas\Ejercicios_Au
toEstudio\Ejercicio_Autoestudio_I\Codigo\Ejercicio_ADT>
java Main
Cero: 0
Uno: 1
Dos: 2
Tres: 3
2 + 3 = 5
0 + 0 = 0
```

```java
public class NatOps {

    public static Nat suma(Nat a, Nat b) {
        return new Nat(sumar(a.getValor(), b.getValor()));
    }

    private static int sumar(int x, int y) {
        if (y == 0) {
            return x;
        } else {
            return sumar(incrementar(x), decrementar(y));
        }
    }

    private static int incrementar(int x) {
        return x - (-1); // suma 1 sin usar '+'
    }

    private static int decrementar(int x) {
        return x + (-1); // resta 1
    }
}
```

**3- Patron en el que se utiliza un atributo interno para distinguir subtipos de un objeto sin recurrir a herencia.**

Type Code Pattern (o "Type Field Pattern")

Este patrón ocurre cuando una clase usa un atributo interno para distinguir subtipos, en lugar de usar herencia o composición. Por ejemplo:

```java
public class Persona {
    enum Genero { MASCULINO, FEMENINO }
    private String nombre;
    private Genero genero;

    // Lógica condicional basada en el género
    public void saludar() {
        if (genero == Genero.MASCULINO) {
            System.out.println("Hola señor " + nombre);
        } else {
            System.out.println("Hola señora " + nombre);
        }
    }
}
```

Sí, puede considerarse un antipatrón si:

Se usa en lugar de herencia o composición.

Lleva a múltiples if/switch en muchas partes del código.

Hace que una clase crezca innecesariamente con lógica para todos los "tipos".

Este patrón rompe el Principio de Abierto/Cerrado (OCP) del diseño orientado a objetos: si quieres agregar un nuevo subtipo, debes modificar la clase base.

Puede ser aceptable si:

El tipo es puramente informativo y no cambia la lógica interna del objeto.

El dominio es muy estable y no justifica crear jerarquías innecesarias.

Se busca mantener una estructura de datos simple y se prefiere evitar polimorfismo.

```java
abstract class Persona {
    protected String nombre;
    public Persona(String nombre) { this.nombre = nombre; }
    public abstract void saludar();
}

class Hombre extends Persona {
    public Hombre(String nombre) { super(nombre); }
    public void saludar() { System.out.println("Hola señor " + nombre); }
}

class Mujer extends Persona {
    public Mujer(String nombre) { super(nombre); }
    public void saludar() { System.out.println("Hola señora " + nombre); }
}
```