PRÁCTICA 1 HISTORIA, EVOLUCIÓN Y CARACTERÍSTICAS DE LENG. DE PROGRAMACIÓN

EJERCICIO 1

1951 - 1955: Lenguajes tipo Assembly

- **Características nuevas:**
    - **Programación simbólica:** Los lenguajes de ensamblador reemplazaron las instrucciones numéricas directas (código máquina) con mnemónicos legibles por humanos (por ejemplo, MOV en lugar de un código binario).
    - **Abstracción de bajo nivel:** Permitían trabajar directamente con el hardware, pero con un nivel de abstracción ligeramente mayor que el código máquina.
    - **Dependencia del hardware:** Cada ensamblador estaba diseñado para una arquitectura específica, lo que facilitaba el control detallado de los recursos del sistema.
- **Lenguaje que lo incorpora:** Lenguajes tipo **Assembly** (no hay un lenguaje específico destacado, ya que eran específicos para cada máquina, como los ensambladores para computadoras como la UNIVAC o IBM 701).

1956 - 1960: FORTRAN, ALGOL 58, ALGOL 60, LISP

- **Características nuevas:**
    - **Lenguajes de alto nivel:** Introdujeron abstracciones que permitían escribir programas sin preocuparse directamente por el hardware.
    - **Estructuras de control avanzadas:** Incluyeron bucles, condicionales y subrutinas, facilitando la escritura de programas complejos.
    - **Compilación:** Uso de compiladores para traducir código de alto nivel a código máquina, mejorando la portabilidad.
    - **Programación científica:** Soporte para cálculos numéricos complejos (en **FORTRAN**).
    - **Programación estructurada (bases): ALGOL 58 y 60** introdujeron bloques de código, ámbito de variables y sintaxis estructurada.
    - **Programación funcional: LISP** introdujo el concepto de manipulación simbólica, listas como estructuras de datos fundamentales y funciones como objetos de primera clase.
- **Lenguajes que las incorporan:**
    - **FORTRAN (1957):** Primer lenguaje de alto nivel para cálculos científicos, con soporte para matrices y subrutinas.

- **ALGOL 58/60 (1958-1960):** Sintaxis estructurada, bloques de código y definición formal de gramáticas (BNF, Backus-Naur Form).
- **LISP (1958):** Programación funcional, gestión automática de memoria (recolección de basura) y estructuras dinámicas.

1961 - 1965: COBOL, ALGOL 60, SNOBOL, JOVIAL

- **Características nuevas:**
  - **Programación orientada a negocios:** Diseñada para procesar datos comerciales, con énfasis en la legibilidad para no programadores.
  - **Manipulación de cadenas:** Soporte avanzado para procesamiento de texto y patrones.
  - **Portabilidad mejorada:** Mayor independencia del hardware gracias a estándares más definidos.
  - **Programación estructurada consolidada:** Refinamiento de estructuras de control y modularidad.
- **Lenguajes que las incorporan:**
  - **COBOL (1960, consolidado en 1961):** Sintaxis en inglés para aplicaciones empresariales, manejo de registros y archivos.
  - **ALGOL 60 (1960, ampliamente usado):** Estandarización de estructuras de control (if-then-else, bucles for) y bloques anidados.
  - **SNOBOL (1962):** Procesamiento avanzado de cadenas y coincidencia de patrones, precursor de lenguajes para texto.
  - **JOVIAL (1960, usado en 1961):** Orientado a sistemas embebidos y aplicaciones militares, con énfasis en modularidad.

1966 - 1970: APL, FORTRAN 66, BASIC, PL/I, SIMULA 67, ALGOL-W

- **Características nuevas:**
  - **Programación interactiva:** Interfaces más amigables para usuarios no expertos.
  - **Programación orientada a objetos (inicios):** Introducción de conceptos como clases y objetos.
  - **Lenguajes multipropósito:** Diseños que combinaban capacidades científicas, comerciales y de sistemas.
  - **Notación matemática avanzada:** Uso de operadores especiales para cálculos complejos.
  - **Simplicidad para principiantes:** Diseños accesibles para enseñar programación.
- **Lenguajes que las incorporan:**

- **APL (1966):** Notación matemática concisa para operaciones con arreglos multidimensionales.
- **FORTRAN 66 (1966):** Estandarización y mejoras en subrutinas y entrada/salida.
- **BASIC (1964, popular en 1966):** Simplicidad para enseñar programación a principiantes, con sintaxis minimalista.
- **PL/I (1966):** Lenguaje multipropósito que combinaba características de FORTRAN, COBOL y ALGOL.
- **SIMULA 67 (1967):** Primer lenguaje orientado a objetos, con clases, herencia y objetos para simulación.
- **ALGOL-W (1966):** Mejoras a ALGOL 60, con énfasis en estructuras de datos y claridad sintáctica.

1971 - 1975: Pascal, C, Scheme, Prolog

- **Características nuevas:**
  - **Programación estructurada formal:** Énfasis en modularidad, tipos de datos fuertes y claridad.
  - **Programación de sistemas:** Control cercano al hardware con abstracciones de alto nivel.
  - **Programación lógica:** Resolución de problemas basada en reglas y hechos.
  - **Programación funcional refinada:** Mayor énfasis en funciones puras y evaluación perezosa.
- **Lenguajes que las incorporan:**
  - **Pascal (1970, usado en 1971):** Tipado fuerte, estructuras de datos definidas y enseñanza de programación estructurada.
  - **C (1972):** Programación de sistemas con acceso a bajo nivel y alta eficiencia, base de sistemas operativos como UNIX.
  - **Scheme (1975):** Dialecto minimalista de LISP, con énfasis en funciones de primera clase y simplicidad.
  - **Prolog (1972):** Programación lógica para inteligencia artificial, con inferencia automática basada en hechos y reglas.

1976 - 1980: Smalltalk, Ada, FORTRAN 77, ML

- **Características nuevas:**
  - **Programación orientada a objetos completa:** Entornos integrados con objetos, herencia y polimorfismo.
  - **Seguridad y fiabilidad:** Diseños para sistemas críticos con verificación estricta.

    - **Tipado estático avanzado:** Sistemas de tipos para prevenir errores en tiempo de compilación.
    - **Metaprogramación inicial:** Capacidades para manipular el propio código.

- **Lenguajes que las incorporan:**
    - **Smalltalk (1980, desarrollado desde 1976):** Programación orientada a objetos pura, con entornos gráficos y mensajes entre objetos.
    - **Ada (1980):** Tipado fuerte, modularidad y diseño para sistemas críticos (aviación, defensa).
    - **FORTRAN 77 (1978):** Mejoras en estructuras de control y portabilidad para aplicaciones científicas.
    - **ML (1973, consolidado en 1976):** Tipado estático, inferencia de tipos y programación funcional.

1981 - 1985: Smalltalk 80, Turbo Pascal, Postscript

- **Características nuevas:**
    - **Entornos gráficos integrados:** Programación visual y orientada a interfaces.
    - **Compilación rápida:** Entornos de desarrollo optimizados para productividad.
    - **Lenguajes para gráficos:** Diseños para describir contenido visual de manera programática.
- **Lenguajes que las incorporan:**
    - **Smalltalk 80 (1980, estandarizado en 1981):** Entorno completo con GUI, influyó en interfaces modernas.
    - **Turbo Pascal (1983):** Compilador rápido y entorno integrado para desarrollo eficiente.
    - **Postscript (1982):** Lenguaje para describir gráficos vectoriales y documentos impresos.

1986 - 1990: FORTRAN 90, C++, SML

- **Características nuevas:**
    - **Programación orientada a objetos en lenguajes de sistemas:** Combinación de abstracciones de alto nivel con control de bajo nivel.
    - **Paralelismo inicial:** Soporte para computación distribuida y matricial.
    - **Programación funcional estandarizada:** Refinamiento de lenguajes funcionales con tipado avanzado.
- **Lenguajes que las incorporan:**
    - **FORTRAN 90 (1990):** Soporte para arreglos dinámicos, recursión y programación modular.

- **C++ (1985, estandarizado en 1986):** Orientación a objetos (clases, herencia) combinada con programación de sistemas.
- **SML (1987):** Estandarización de ML, con módulos y tipado polimórfico.

1991 - 1995: TCL, PERL, HTML

- **Características nuevas:**
  - **Programación de scripts:** Ejecución rápida para tareas de automatización y glue code.
  - **Procesamiento de texto avanzado:** Herramientas para manejar datos no estructurados.
  - **Lenguajes para la web:** Marcado para estructurar contenido en internet.
- **Lenguajes que las incorporan:**
  - **TCL (1990, popular en 1991):** Scripting para control de aplicaciones y automatización.
  - **PERL (1987, ampliamente usado en 1991):** Procesamiento de texto, expresiones regulares y scripts para servidores.
  - **HTML (1990, estandarizado en 1993):** Estructuración de contenido web con hipertexto.

1996 - 2000: Java, Javascript, XML

- **Características nuevas:**
  - **Portabilidad total:** Ejecución en múltiples plataformas sin recompilar.
  - **Programación web dinámica:** Interactividad en navegadores.
  - **Datos estructurados:** Formatos para intercambio de información entre sistemas.
  - **Manejo automático de memoria:** Simplificación del desarrollo con recolección de basura.
- **Lenguajes que las incorporan:**
  - **Java (1995, popular en 1996):** Máquina virtual (JVM) para portabilidad, orientación a objetos y robustez.
  - **Javascript (1995, estandarizado en 1996):** Programación del lado del cliente para páginas web dinámicas.
  - **XML (1998):** Formato para datos estructurados, independiente de plataformas.

EJERCICIO 2

Java es un lenguaje de programación desarrollado por Sun Microsystems (ahora parte de Oracle) y lanzado en 1995. Fue creado por un equipo liderado por James Gosling con el objetivo de ser un lenguaje portátil, robusto y fácil de usar. Su diseño se inspiró en C y C++, pero eliminó características complicadas como la gestión manual de memoria, introduciendo en su lugar un recolector de basura automático.

Una de las claves de su éxito es la Máquina Virtual de Java (JVM), que permite que el código compilado (bytecode) se ejecute en cualquier dispositivo con la JVM instalada, bajo el lema "escribe una vez, ejecuta en cualquier lugar". Inicialmente pensado para dispositivos electrónicos, Java ganó popularidad en aplicaciones empresariales, desarrollo web (con applets) y, más tarde, en móviles con Android. Hoy es uno de los lenguajes más utilizados globalmente gracias a su versatilidad y comunidad activa.

EJERCICIO 3

SIMPLICIDAD Y LEGIBILIDAD

- Por ejemplo: Python y Ruby
- Los lenguajes de programación deberían:
    - Poder producir programas fáciles de escribir y de leer.
    - Resultar fáciles a la hora de aprenderlo y enseñarlo
- Ejemplo de cuestiones que atentan contra esto:
    - Muchas componentes elementales
    - Conocer subconjuntos de componentes
    - El mismo concepto semántico – distinta sintaxis
    - Distintos conceptos semánticos - la misma notación sintáctica
    - Abuso de operadores sobrecargados

CLARIDAD EN LOS BINDINGS

- Por ejemplo: Rust, Java y TypeScript
- Los elementos de los lenguajes de programación pueden ligarse a sus atributos o propiedades en diferentes momentos:
    - Definición del lenguaje
    - Implementación del lenguaje
    - En escritura del programa
    - Compilación
    - Cargado del programa
    - En ejecución
- La ligadura en cualquier caso debe ser clara

CONFIABILIDAD

- Por ejemplo: Ada, Rust y Erlang
- La confiabilidad está relacionada con la seguridad
    - Chequeo de tipos

        - Cuanto antes se encuentren errores menos costoso resulta realizar los arreglos que se requieran.
    - Manejo de excepciones
        - La habilidad para interceptar errores en tiempo de ejecución, tomar medidas correctivas y continuar.

SOPORTE

- Por ejemplo: Java, Python y C#
- Debería ser accesible para cualquiera que quiera usarlo o instalarlo
    - Lo ideal sería que su compilador o intérprete sea de dominio público
- Debería poder ser implementado en diferentes plataformas
- Deberían existir diferentes medios para poder familiarizarse con el lenguaje: tutoriales, cursos textos, etc.

ABSTRACCIÓN

- Por ejemplo: Scala y Python
- Capacidad de definir y usar estructuras u operaciones complicadas de manera que sea posible ignorar muchos de los detalles.
    - Abstracción de procesos y de datos

ORTOGONALIDAD

- Por ejemplo: GO y Scheme
- Se refiere a un principio de diseño donde las construcciones o características del lenguaje son independientes entre sí y pueden combinarse libremente sin restricciones inesperadas o efectos secundarios. En un lenguaje ortogonal, cada elemento tiene un propósito claro y no está innecesariamente acoplado a otros, lo que permite una mayor flexibilidad y consistencia.

EFICIENCIA

- Por Ejemplo: C++ y Rust
- Tiempo y Espacio
    - Capacidad de un programa para realizar una tarea de manera correcta y consumiendo pocos recursos, de memoria, espacio en disco, tiempo de ejecución, tráfico de red, etc.
- Esfuerzo humano
    - que mejore el rendimiento del programador
- Optimizable
    - capacidad del lenguaje de programación de venir optimizado para tareas específicas

EJERCICIO 4

TIPOS DE EXPRESIONES QUE SE PUEDEN ESCRIBIR EN JAVA

- **Literales**
  - Descripción: Valores directos que representan constantes.
  - Ejemplos: 5 (entero), 3.14 (flotante), "Hola" (cadena), true (booleano), null (referencia nula).
- **Expresiones con variables**
  - Descripción: Uso de identificadores para acceder a valores almacenados.
  - Ejemplo: x (donde x es una variable declarada como int x = 5;).
- **Expresiones aritméticas**
  - Descripción: Operaciones matemáticas que combinan valores numéricos.
  - Ejemplos: x + 3, 5 * y, a / b - 2.
- **Expresiones relacionales y lógicas**
  - Descripción: Comparaciones y operaciones booleanas.
  - Ejemplos: x > 5, a == b, x < 10 && y > 0, !condicion.
- **Expresiones de asignación**
  - Descripción: Asignan un valor y devuelven ese valor como resultado.
  - Ejemplo: x = 10 (asigna 10 a x y evalúa a 10).
- **Expresiones de invocación de métodos**
  - Descripción: Llamadas a métodos que devuelven un valor.
  - Ejemplo: Math.sqrt(16) (devuelve 4.0), objeto.metodo().
- **Expresiones de acceso a campos o elementos**
  - Descripción: Acceso a propiedades de objetos o elementos de arreglos.
  - Ejemplos: objeto.campo, arreglo[2].
- **Expresiones de creación de objetos**
  - Descripción: Instanciación de clases o arreglos.
  - Ejemplos: new String("Hola"), new int[5].
- **Expresiones condicionales (operador ternario)**
  - Descripción: Forma compacta de una condición que devuelve un valor.
  - Ejemplo: x > 0 ? 1 : -1 (devuelve 1 si x > 0, -1 si no).
- **Expresiones lambda (desde Java 8)**
  - Descripción: Expresiones funcionales para interfaces con un solo método.

- Ejemplo: x -> x * 2 (una función que duplica un número).

Nota: En Java, estructuras como if, while o for son *statements*, no expresiones, porque no producen valores directamente

FACILIDADES PROVISTAS POR JAVA PARA LA ORGANIZACIÓN DEL PROGRAMA

1. Clases y Objetos

- **Descripción**: Java organiza el código en clases, que son plantillas para crear objetos. Todo el código debe residir dentro de una clase, promoviendo la encapsulación y la modularidad.

Ejemplo:

```java
public class Persona {
    String nombre;
    int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}
```

- **Beneficio**: Agrupa datos y comportamientos relacionados, facilitando la reutilización y el mantenimiento.

2. Paquetes (Packages)

- **Descripción**: Los paquetes permiten agrupar clases relacionadas en un espacio de nombres, evitando conflictos y organizando el código en módulos jerárquicos.

Ejemplo:

```java
package com.ejemplo.modelos;

public class Empleado {
// Código de la clase
}
```

- **Uso**: import com.ejemplo.modelos.Empleado; para acceder desde otro archivo.
- **Beneficio**: Estructura proyectos grandes (ej. java.util, java.io) y mejora la legibilidad.

3. Modificadores de acceso

- **Descripción**: Java ofrece modificadores como public, private, protected y el acceso por defecto (package-private) para controlar la visibilidad de campos, métodos y clases.

Ejemplo:

```java
public class Cuenta {
    private double saldo; // Solo accesible dentro de la clase

    public double getSaldo() {
        return saldo;
    }
}
```

- **Beneficio**: Fomenta la encapsulación y protege la implementación interna.

4. Herencia

- **Descripción**: Permite que una clase herede campos y métodos de otra, organizando jerarquías de código reutilizable.

Ejemplo:

```java
public class Animal {
    String nombre;
}

public class Perro extends Animal {
    void ladrar() {
        System.out.println("Guau");
    }
}
```

- **Beneficio**: Reduce duplicación y organiza relaciones entre entidades.

5. Interfaces

- **Descripción**: Define contratos que las clases deben implementar, separando la especificación de la implementación.

Ejemplo:

```java
public interface Volador {
    void volar();
}

public class Pajaro implements Volador {
    public void volar() {
        System.out.println("Volando...");
    }
}
```

- **Beneficio**: Promueve el diseño modular y la interoperabilidad.

6. Clases Abstractas

- **Descripción**: Clases que no se pueden instanciar directamente, usadas como base para otras clases con métodos abstractos o concretos.

Ejemplo:

```
public abstract class Vehiculo {
    abstract void mover();
}

public class Auto extends Vehiculo {
    void mover() {
        System.out.println("El auto se mueve");
    }
}
```

- **Beneficio**: Organiza jerarquías con comportamiento compartido.

7. Métodos y constructores

- **Descripción**: Los métodos organizan la lógica en bloques reutilizables, y los constructores inicializan objetos de manera estructurada.

Ejemplo:

```
public class Libro {
    String titulo;

    public Libro(String titulo) {
        this.titulo = titulo;
    }

    public String getTitulo() {
        return titulo;
    }
}
```

- **Beneficio**: Divide el código en unidades funcionales claras.

8. Excepciones

- **Descripción**: El manejo de excepciones con try, catch, throw y throws organiza la gestión de errores.

Ejemplo:

```
public void dividir(int a, int b) throws ArithmeticException {
    if (b == 0) throw new ArithmeticException("División por cero");
    System.out.println(a / b);
}
```

- **Beneficio**: Separar la lógica normal del manejo de casos excepcionales.

9. Enumeraciones (Enums)

- **Descripción**: Define conjuntos de constantes con un tipo seguro, organizando valores predefinidos.

Ejemplo:

```
public enum Dia {
    LUNES, MARTES, MIERCOLES
}
```

- **Beneficio**: Mejora la claridad y evita errores con valores mágicos.

10. Anotaciones (Annotations)

- **Descripción**: Metadatos que organizan y configuran el comportamiento del código (ej. @Override, @Deprecated).

Ejemplo:

```
@Override
public String toString() {
  return "Objeto personalizado";
}
```

- **Beneficio**: Facilita la configuración y el mantenimiento.

11. Estructura de archivos y convención de nombres

- **Descripción**: Java exige que el nombre de la clase pública coincida con el nombre del archivo (ej. Clase.java) y sigue convenciones como CamelCase.
- **Ejemplo**: Archivo Persona.java contiene public class Persona.
- **Beneficio**: Estandariza la organización física del proyecto.

ATRIBUTOS QUE POSEE Y NO POSEE JAVA

1. SIMPLICIDAD

- **Posee**: Parcialmente
- **Justificación**: Java tiene una sintaxis estructurada y explícita, pero no es tan simple como lenguajes como Python o Lua. Requiere declaraciones de tipos, modificadores de acceso y una estructura de clases obligatoria, lo que añade complejidad inicial (ej. public class Hola { public static void main(String[] args) { System.out.println("Hola"); } }). Aunque es más simple que C++ al eliminar punteros manuales y gestión de memoria, su verbosidad lo aleja de la simplicidad máxima.

2. LEGIBILIDAD

- **Posee**: Sí
- **Justificación**: La sintaxis de Java es clara y estructurada, con nombres descriptivos y convenciones como CamelCase que facilitan la lectura (ej. getNombre()). Los bloques delimitados por llaves {} y la obligatoriedad de clases hacen que el código sea predecible y fácil de seguir, aunque puede ser más verboso que lenguajes como Ruby.

3. CLARIDAD EN LOS *BINDINGS*

- **Posee**: Sí
- **Justificación**: Java usa tipado estático, lo que significa que los *bindings* (asociaciones entre variables y tipos) se definen en tiempo de compilación y son explícitos (ej. int x = 5;). Esto evita ambigüedades y errores en tiempo de ejecución relacionados con tipos, ofreciendo claridad y seguridad en cómo se vinculan los datos.

4. CONFIABILIDAD

- **Posee**: Sí
- **Justificación**: Java está diseñado para ser confiable, con características como manejo de excepciones robusto (try-catch), tipado estático, y una máquina virtual (JVM) que previene errores comunes como desbordamientos de memoria o punteros nulos no controlados. Es ampliamente usado en sistemas críticos (ej. bancarios), lo que demuestra su fiabilidad.

5. SOPORTE

- **Posee**: Sí
- **Justificación**: Java cuenta con un soporte excepcional: respaldado por Oracle, tiene una comunidad masiva, documentación extensa y un ecosistema de bibliotecas (ej. Spring, Hibernate). Además, su compatibilidad multiplataforma y actualizaciones regulares aseguran soporte continuo para desarrolladores.

6. ABSTRACCIÓN

- **Posee**: Sí
- **Justificación**: Java ofrece potentes mecanismos de abstracción a través de clases, interfaces y clases abstractas (ej. interface Comparable { int compareTo(T o); }). La programación orientada a objetos y las características modernas como lambdas (desde Java 8) permiten modelar problemas complejos sin exponer detalles de bajo nivel.

7. ORTOGONALIDAD

- **Posee**: Parcialmente
- **Justificación**: Java no es completamente ortogonal debido a restricciones en su diseño. Por ejemplo, no todos los tipos pueden usarse uniformemente (los primitivos como int no son objetos, a diferencia de Integer), y estructuras como switch no aceptan todos los tipos (solo enteros, cadenas desde Java 7). Aunque es consistente en muchas áreas, estas excepciones limitan su ortogonalidad frente a lenguajes como Scheme.

8. EFICIENCIA

- **Posee**: Sí, pero con matices
- **Justificación**: Java es eficiente gracias a la JVM y el compilador JIT (Just-In-Time), que optimizan el código en tiempo de ejecución, alcanzando un rendimiento cercano a lenguajes compilados

como C++. Sin embargo, no iguala la eficiencia de bajo nivel de C++ o Rust debido a la sobrecarga de la JVM y el *garbage collector*, lo que lo hace menos ideal para aplicaciones de tiempo real extremadamente críticas.

LENGUAJES - ADA

EJERCICIO 5

1. TIPOS DE DATOS

Ada es un lenguaje fuertemente tipado. Esto significa que no permite conversiones implícitas entre tipos distintos, lo cual mejora la seguridad y confiabilidad del código. Los tipos de datos se dividen en varias categorías:

- **Tipos escalares**: enteros, reales, booleanos y caracteres.
- **Tipos derivados**: se pueden crear nuevos tipos a partir de otros ya existentes, con sus propias restricciones.
- **Subtipos**: permiten definir rangos específicos dentro de un tipo base.
- **Tipos enumerados**: definidos por el usuario, útiles para representar conjuntos finitos de valores.

Gracias a este sistema de tipos, Ada detecta muchos errores durante la compilación, ayudando a evitar fallos en tiempo de ejecución.

2. TIPOS ABSTRACTOS DE DATOS – PAQUETES

Ada utiliza **paquetes (packages)** como mecanismo principal para definir tipos abstractos de datos. Un paquete encapsula tanto la declaración de tipos y operaciones (interfaz pública) como su implementación interna (parte privada).

Esta separación permite:

- Ocultar los detalles de implementación.
- Reutilizar código de manera estructurada.
- Mejorar el mantenimiento y la legibilidad.

Los paquetes promueven una programación modular y organizada, facilitando la construcción de aplicaciones grandes y complejas.

3. ESTRUCTURAS DE DATOS

Ada ofrece varios mecanismos para la construcción de estructuras de datos complejas:

- **Arrays**: permiten el almacenamiento de colecciones indexadas de elementos.
- **Registros**: agrupan varios campos de distintos tipos bajo un mismo nombre, similar a estructuras en otros lenguajes.

- **Tipos de acceso (punteros)**: permiten la creación y manipulación de estructuras dinámicas como listas o árboles.
- **Tipos definidos por el usuario**: permiten construir tipos personalizados adaptados a las necesidades del programa.

Estas herramientas proporcionan flexibilidad para implementar estructuras clásicas como pilas, colas, listas enlazadas, entre otras.

4. MANEJO DE EXCEPCIONES

Ada incluye un sistema robusto para el manejo de errores mediante **excepciones**. Este mecanismo permite capturar y controlar errores durante la ejecución del programa sin interrumpir su flujo de manera abrupta.

El lenguaje ofrece excepciones predefinidas para errores comunes (como violaciones de rango o errores de tipo), y también permite que el programador defina sus propias excepciones personalizadas.

El manejo de excepciones en Ada mejora la confiabilidad y estabilidad del software, especialmente en sistemas críticos.

5. MANEJO DE CONCURRENCIA

Ada se destaca por su soporte nativo para la **programación concurrente** mediante el uso de **tasks**. Cada task puede ejecutarse de forma independiente y simultánea con otras tareas.

Para coordinar la interacción entre tareas, Ada ofrece mecanismos como:

- **Entries y rendezvous**: puntos de sincronización entre tareas.
- **Objetos protegidos (protected objects)**: permiten el acceso seguro a recursos compartidos entre múltiples tareas, evitando condiciones de carrera.

Gracias a estas características, Ada es ideal para sistemas embebidos, tiempo real y aplicaciones que requieren ejecución paralela y sincronización precisa.

LENGUAJES - JAVA

EJERCICIO 6

¿PARA QUÉ FUE CREADO JAVA?

Java fue creado originalmente por Sun Microsystems, con el objetivo de ser un lenguaje de programación **portátil, seguro, y orientado a objetos**. Su propósito principal era permitir el desarrollo de software que pudiera ejecutarse en distintos dispositivos electrónicos (como electrodomésticos inteligentes), sin necesidad de ser reescrito para cada uno. Sin embargo, debido a su versatilidad y potencia, Java rápidamente se adaptó al desarrollo de aplicaciones de escritorio, empresariales y web.

¿QUÉ CAMBIOS INTRODUJO JAVA EN LA WEB?

Java tuvo un impacto importante en la evolución de la Web al permitir:

- **Interactividad y dinamismo** mediante el uso de **applets** (pequeños programas Java que se ejecutaban en navegadores).
- Desarrollo de **aplicaciones web complejas** y dinámicas del lado del servidor, gracias a tecnologías como **Servlets**, **JSP (JavaServer Pages)** y más adelante frameworks como **Spring**.
- Promoción del modelo de programación **cliente-servidor**, facilitando la creación de sistemas distribuidos y aplicaciones empresariales basadas en la web.

Aunque los applets quedaron obsoletos con el tiempo, Java consolidó su lugar en el backend de aplicaciones web y sistemas empresariales.

¿JAVA ES UN LENGUAJE DEPENDIENTE DE LA PLATAFORMA EN DONDE SE EJECUTA

No, Java **no es dependiente de la plataforma**. Esta es una de sus principales características.

Java fue diseñado para ser **independiente de la plataforma** gracias a su lema: **"Write once, run anywhere"** (escribe una vez, ejecuta en cualquier parte). Esto es posible porque el código Java se compila a un formato intermedio llamado **bytecode**, que no es específico de ningún sistema operativo.

Este bytecode es interpretado o ejecutado por la **Java Virtual Machine (JVM)**, que actúa como una capa intermedia entre el programa Java y el sistema operativo. Mientras exista una JVM para una plataforma determinada (Windows, Linux, macOS, etc.), el programa Java podrá ejecutarse sin modificaciones.

EJERCICIO 7

- **C**: Hereda gran parte de su **sintaxis**, como el uso de llaves {}, estructuras de control (if, for, while), operadores y estilo de declaración de variables.
- **C++**: Toma los conceptos principales de la **programación orientada a objetos**, como clases, objetos, herencia, encapsulamiento y polimorfismo. Sin embargo, Java evita complejidades de C++ como la herencia múltiple directa y los punteros explícitos.
- **Smalltalk**: Influye en el modelo puro de orientación a objetos. En Java, **todo el código debe estar dentro de una clase**, lo cual es similar a la filosofía de Smalltalk.
- **Otros lenguajes modernos**: Java también incorpora ideas de lenguajes más modernos para mejorar la seguridad, la simplicidad del código y la robustez del entorno de ejecución.

EJERCICIO 8

¿QUÉ SON LOS APPLETS?

Los **applets** eran pequeños programas escritos en Java que se ejecutaban dentro de un navegador web. Estaban diseñados para hacer las páginas web más **interactivas y dinámicas**, ya que permitían ejecutar código en el cliente (es decir, en la computadora del usuario).

Características principales:

- Se ejecutaban en el navegador mediante un **plugin de Java**.
- Requerían estar incrustados en una página HTML.
- Tenían acceso limitado al sistema del usuario por razones de seguridad.
- Están actualmente en **desuso** y ya no son soportados por los navegadores modernos.

Los applets fueron útiles en los primeros años de la web interactiva, pero fueron reemplazados por tecnologías más ligeras y seguras como JavaScript, HTML5 y CSS.

¿QUÉ SON LOS SERVLETS?

Los **servlets** son programas Java que se ejecutan en el **servidor web**. Están diseñados para **procesar peticiones de clientes**, como las que se realizan al enviar un formulario o acceder a una página dinámica.

Características principales:

- Se ejecutan dentro de un **contenedor de servlets** (como Apache Tomcat).
- Permiten generar contenido dinámico (por ejemplo, páginas HTML construidas en tiempo real).
- Se usan para manejar peticiones HTTP (GET, POST, etc.).
- Son parte fundamental del desarrollo web del lado del servidor en Java.
- Son la base de tecnologías más avanzadas como **JavaServer Pages (JSP)** y frameworks como **Spring**.

Mientras que los applets eran para el cliente (navegador), los servlets trabajan en el servidor y siguen siendo ampliamente usados en el desarrollo web Java.

LENGUAJES - C

EJERCICIO 9

¿Cómo es la estructura de un programa escrito en C?

Un programa en C está compuesto por distintas secciones que siguen una estructura típica. Las principales partes son:

1. **Directivas del preprocesador**
   Instrucciones que comienzan con #, como #include, para incluir bibliotecas estándar.
2. **Definiciones globales**
   Definición de constantes, macros, variables globales y prototipos de funciones.
3. **Función main()**
   Es el punto de entrada del programa. Desde acá comienza la ejecución.
4. **Funciones auxiliares**
   Otras funciones definidas por el programador para dividir el código en bloques reutilizables y organizados.

Estructura básica:

```c
#include <stdio.h>

// Prototipo de función
void saludar();

int main() {
    saludar();
    return 0;
}

// Definición de función
void saludar() {
    printf("Hola mundo\n");
}
```

¿EXISTE ANIDAMIENTO DE FUNCIONES EN C?

No, en C **no se permite el anidamiento de funciones**. Es decir, **no se pueden definir funciones dentro de otras funciones**. Todas las funciones deben estar definidas a nivel global, fuera de otras funciones.

Sin embargo, una función puede **llamar a otras funciones** desde su interior, lo cual sí es muy común y totalmente válido.

- Ejemplo válido: la función main() puede llamar a otras funciones.
- Ejemplo no válido: no se puede definir una función nueva dentro del cuerpo de main().

EJERCICIO 10

En C, una **expresión** es cualquier combinación válida de **operandos** (valores, variables o constantes) y **operadores** que el lenguaje puede evaluar para producir un resultado.

El lenguaje C proporciona un manejo flexible y potente de expresiones, permitiendo realizar desde operaciones simples hasta cálculos complejos.

TIPOS DE EXPRESIONES EN C

1. ### EXPRESIONES ARITMÉTICAS
   Permiten realizar operaciones matemáticas usando operadores como +, -, *, /, %.

   Ejemplo: resultado = a + b * c;

2. ### EXPRESIONES RELACIONALES
   Comparan valores y devuelven un resultado booleano (0 o 1 en C).

   Operadores: ==, !=, <, >, <=, >=
   Ejemplo: esMayor = (x > y);

3. ### EXPRESIONES LÓGICAS
   Combinan valores booleanos usando operadores lógicos: && (AND), || (OR), ! (NOT).

   Ejemplo: if (edad > 18 && tieneLicencia)

4. EXPRESIONES DE ASIGNACIÓN
   Se usan para asignar un valor a una variable. El operador principal es =.

   Ejemplo: x = 10;

5. EXPRESIONES CONDICIONALES (OPERADOR TERNARIO)
   Proporcionan una forma abreviada de tomar decisiones.

   Ejemplo: resultado = (a > b) ? a : b;

6. EXPRESIONES CON OPERADORES A NIVEL DE BITS
   Permiten manipular los bits individuales de datos.

   Ejemplo: x = x << 1; // Desplazamiento a la izquierda

PRECEDENCIA Y ASOCIATIVIDAD

C define un **orden de precedencia** entre operadores que determina cómo se agrupan las expresiones cuando hay más de un operador. También establece una **asociatividad** (de izquierda a derecha o viceversa) para resolver empates de precedencia.

Por ejemplo: En a + b * c, primero se evalúa b * c por tener mayor precedencia que +.

LENGUAJES – PYTHON, RUBY. PHP

EJERCICIO 11

PYTHON

- ¿QUÉ TIPO DE PROGRAMAS SE PUEDEN ESCRIBIR?
  Python es un lenguaje muy versátil, ideal para:
  - Desarrollo web (con frameworks como Django o Flask).
  - Automatización de tareas y scripting.
  - Análisis de datos y ciencia de datos.
  - Inteligencia artificial y aprendizaje automático.
  - Desarrollo de aplicaciones de escritorio.
  - Educación y prototipado rápido.
- PARADIGMA:
  Multiparadigma, con fuerte soporte para:
  - Programación **imperativa**.
  - Programación **orientada a objetos**.
  - Programación **funcional**.
- CARACTERÍSTICAS QUE DETERMINAN SU PARADIGMA:

- Definición de clases y objetos.
- Funciones como ciudadanos de primera clase (pueden pasarse como argumentos).
- Control de flujo estructurado con condicionales y bucles.

RUBY

- ¿QUÉ TIPO DE PROGRAMAS SE PUEDEN ESCRIBIR?
  Ruby se usa especialmente en:
  - Desarrollo web (destaca el framework **Ruby on Rails**).
  - Automatización de tareas y scripts.
  - Herramientas de línea de comandos.
  - Prototipado rápido de aplicaciones.
- PARADIGMA:
  Orientado a objetos en su núcleo, aunque también admite otros estilos como el **funcional** e **imperativo**.
- CARACTERÍSTICAS QUE DETERMINAN SU PARADIGMA:
  - **Todo** en Ruby es un objeto, incluyendo tipos primitivos como enteros y booleanos.
  - Clases y métodos se definen de forma muy flexible.
  - Apoyo a bloques, lambdas y closures (programación funcional).

PHP

- ¿QUÉ TIPO DE PROGRAMAS SE PUEDEN ESCRIBIR?
  PHP es ampliamente utilizado para:
  - **Desarrollo web dinámico del lado del servidor**.
  - Creación de sitios y aplicaciones web (WordPress, Laravel).
  - Sistemas de gestión de contenido (CMS).
  - Scripts en servidores web.
- PARADIGMA:
  Inicialmente **imperativo**, luego evolucionó hacia un lenguaje **multiparadigma**, hoy con soporte para:
  - Programación **orientada a objetos**.
  - Programación **funcional** en cierta medida.
- CARACTERÍSTICAS QUE DETERMINAN SU PARADIGMA:

- Permite escribir scripts secuenciales.
- Soporte para clases, objetos, herencia, interfaces y más.
- Uso de funciones anónimas y cierres.

EJERCICIO 12

Lenguaje	Tipado de datos	Organización del programa	Características adicionales
Python	Dinámico, fuerte	Basado en funciones, módulos y clases. Usa indentación obligatoria.	Sintaxis clara. Multiparadigma. Amplio ecosistema de librerías. Ideal para ciencia de datos, IA, automatización.
Ruby	Dinámico, fuerte	Todo es objeto. Programas estructurados en clases, métodos y bloques.	Muy orientado a objetos. Sintaxis expresiva. Flexible. Usado en desarrollo web (Ruby on Rails).
PHP	Dinámico, débil (mejorado en versiones recientes)	Scripts web estructurados con funciones o clases. Se ejecuta en el servidor.	Pensado para la web. Integra HTML y PHP. Muy usado en CMS como WordPress.
Gobstones	Tipado implícito (educativo)	Basado en procedimientos y pasos secuenciales.	Usado en educación. Entorno visual. Sirve para enseñar lógica y algoritmos sin complejidad sintáctica.
Processing	Dinámico, fuerte (basado en Java)	Funciones especiales `setup()` y `draw()` controlan el flujo del programa.	Ideal para arte, animaciones y visualización. Simplifica el uso de gráficos. Basado en Java.

LENGUAJES - JAVASCRIPT

EJERCICIO 13

PARADIGMA:

JavaScript es un lenguaje **multiparadigma**, ya que permite programar:

- De forma **imperativa** (instrucción por instrucción).
- Con un enfoque **orientado a objetos** (aunque basado en prototipos y no en clases clásicas como Java).
- De forma **funcional**, permitiendo funciones de orden superior, funciones anónimas y programación declarativa.

TIPO DE LENGUAJE:

JavaScript es un lenguaje **interpretado**, **dinámico** y **débilmente tipado**. Fue diseñado principalmente para ejecutarse en el navegador, pero hoy en día también se usa del lado del servidor (por ejemplo, con Node.js).

EJERCICIO 14

TIPADO DE DATOS:

JavaScript es un lenguaje **dinámico** y **débilmente tipado**. Las variables no requieren una declaración explícita de tipo, y pueden cambiar de tipo en tiempo de ejecución.

MANEJO DE EXCEPCIONES:

Utiliza las sentencias try, catch, finally y throw para capturar y manejar errores en tiempo de ejecución.

DECLARACIÓN DE VARIABLES:

Se pueden declarar con:

- var: declaración con ámbito de función.
- let: declaración con ámbito de bloque, introducida en ES6.
- const: para declarar constantes (no se puede reasignar).

FUNCIONES:

Las funciones son ciudadanos de primera clase (se pueden pasar como argumentos, retornar, asignar a variables). También existen funciones flecha (=>) introducidas en ES6.

OBJETOS:

Los objetos son fundamentales. JavaScript utiliza un sistema de herencia basado en **prototipos** en lugar de clases tradicionales (aunque desde ES6 incluye sintaxis de clases).

EVENTOS Y ASINCRONÍA:

Es altamente asincrónico, con soporte para **callbacks**, **promesas** y async/await. Ideal para manejar eventos en entornos web.

EJECUCIÓN EN EL NAVEGADOR Y SERVIDOR:

Nació como lenguaje del navegador, pero hoy también se ejecuta del lado del servidor con plataformas como **Node.js**.