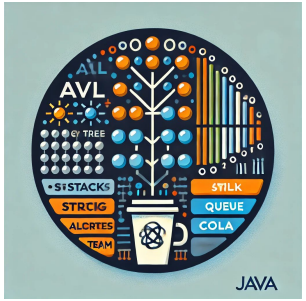


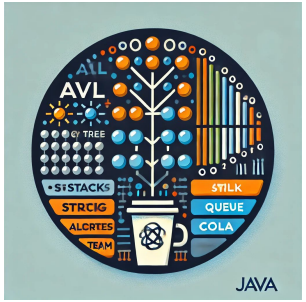
CLASE 2

Ing. Silvestre Alejandro
Informática III
IUA - 2024

Objetivos del día

1. Metodología de trabajo práctico.
2. Introducción a JAVA.
3. Entorno de desarrollo.
4. Programación Orientada a Objetos.

Metodología de trabajo práctico



Metodología de trabajo práctico

Formación de Grupos:

- Organiza a los estudiantes en grupos de dos para trabajar de manera colaborativa en clase. La colaboración en parejas permite una mejor distribución del trabajo y facilita el intercambio de ideas, promoviendo un aprendizaje más profundo.

Asignación de Tópicos:

- En cada sesión de clase, se presentará un tema o problema específico que los grupos deberán resolver. Este enfoque permite que los estudiantes se concentren en aplicar conceptos teóricos a situaciones prácticas, reforzando así su comprensión.

Metodología de trabajo práctico

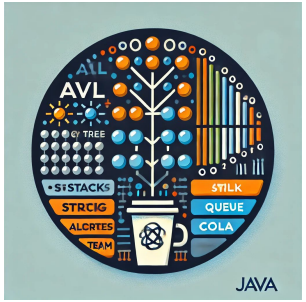
Resolución del Problema:

- Los grupos trabajarán juntos durante la clase para resolver el problema asignado. Este tiempo se utilizará para discutir posibles soluciones, escribir código y probar las implementaciones, fomentando habilidades de colaboración y resolución de problemas.

Revisión de Código (Code Review):

- En la siguiente clase, los primeros 30 minutos estarán dedicados a una revisión de código (Code Review). Un grupo seleccionado al azar presentará su solución implementada al resto de la clase.
- Durante la presentación, el grupo explicará su enfoque, decisiones de diseño y cualquier desafío que hayan enfrentado.
- Los demás estudiantes y el instructor proporcionarán comentarios constructivos, identificando fortalezas y áreas de mejora en el código presentado. Esta práctica no solo ayuda al grupo a mejorar su trabajo, sino que también beneficia a toda la clase al exponerlos a diferentes enfoques y técnicas de programación.

JAVA


AVL
AVL TREE
STACKS
STRCIG
ALCRTES
TEAM
STILK
QUEUE
COLA
JAVA


CENTRO REGIONAL UNIVERSITARIO CORDOBA UTN
1941

Introducción a JAVA - Historia

Java es un lenguaje de programación potente y versátil que ha tenido un impacto significativo en el mundo de la informática.

Creado por James Gosling y su equipo en Sun Microsystems a principios de los años 90, Java ha evolucionado hasta convertirse en uno de los lenguajes más populares y ampliamente utilizados en la actualidad.

Introducción a JAVA - ¿Por qué aprenderlo?

Java es un lenguaje de programación fundamental debido a su capacidad para construir una amplia variedad de aplicaciones, que van desde programas sencillos hasta sistemas complejos y de gran escala. Esta versatilidad se debe a varias razones clave:

Portabilidad: Uno de los mayores beneficios de Java es su lema "escribe una vez, ejecuta en cualquier lugar". Esto significa que los programas escritos en Java pueden ejecutarse en diferentes plataformas sin necesidad de realizar cambios significativos en el código. Esta portabilidad se logra gracias a la máquina virtual de Java (JVM), que permite que el mismo código funcione en cualquier dispositivo que tenga una JVM.

Introducción a JAVA - ¿Por qué aprenderlo?

Robustez y Seguridad: Java está diseñado para ser robusto, con características como la gestión automática de memoria y la detección de errores en tiempo de compilación, lo que reduce significativamente el riesgo de errores de programación. Además, cuenta con sólidas características de seguridad, como la gestión de permisos y el uso de sandboxing, que protegen las aplicaciones de amenazas potenciales.

Gran Ecosistema y Bibliotecas: Java posee un extenso ecosistema de bibliotecas y frameworks que facilitan el desarrollo de aplicaciones. Herramientas como Spring, Hibernate y Apache Struts permiten a los desarrolladores construir aplicaciones de manera más rápida y eficiente, aprovechando soluciones preexistentes y probadas.

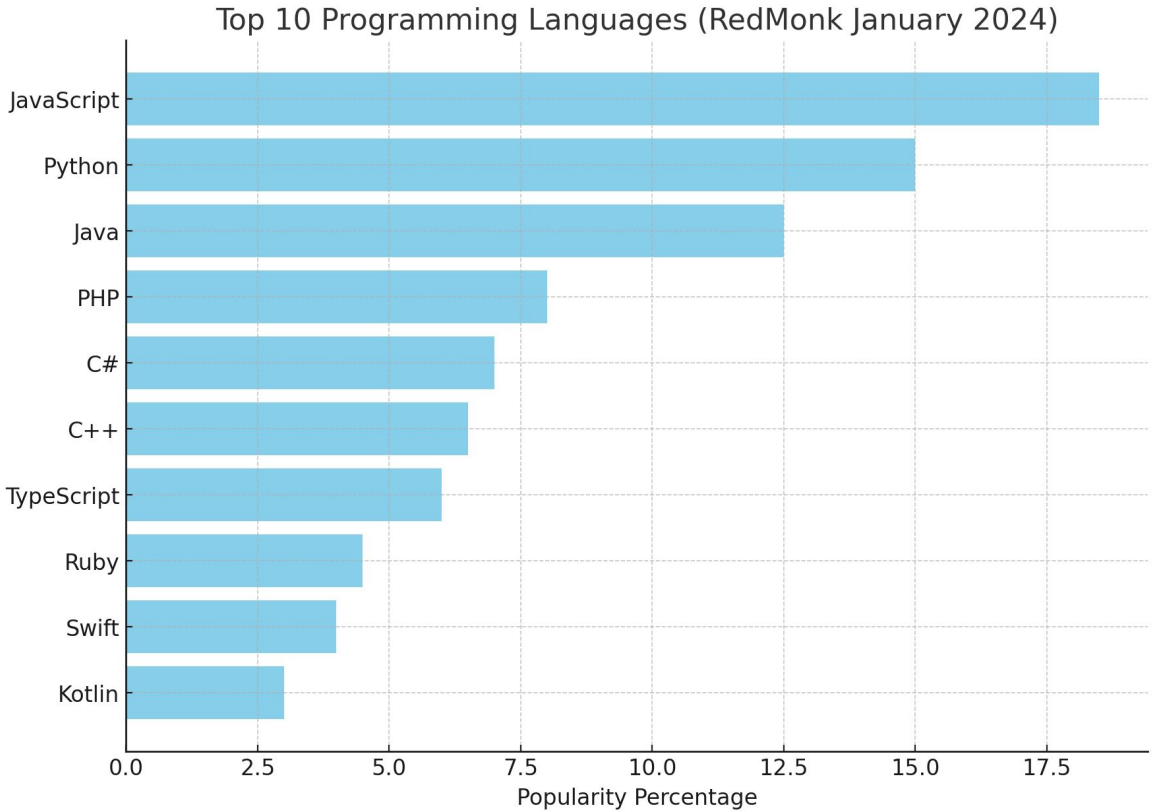
Introducción a JAVA - ¿Por qué aprenderlo?

Comunidad Activa: La comunidad de desarrolladores de Java es una de las más grandes y activas del mundo. Esto significa que hay una gran cantidad de recursos disponibles, como tutoriales, foros, y documentación, que pueden ayudar a los desarrolladores a resolver problemas y aprender nuevas técnicas.

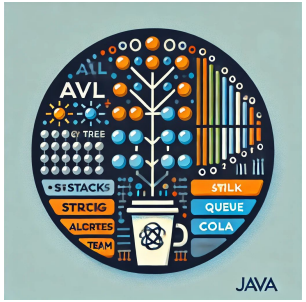
Escalabilidad: Java es ideal para construir aplicaciones escalables, lo que lo hace perfecto para el desarrollo de grandes sistemas empresariales y aplicaciones de alto rendimiento. Su capacidad para manejar grandes cantidades de datos y usuarios de manera eficiente es crucial para muchas empresas.

Soporte a Largo Plazo: Oracle y otros grandes actores de la industria continúan ofreciendo soporte y actualizaciones para Java, asegurando que se mantenga relevante y actualizado con las últimas tendencias y necesidades tecnológicas.

Introducción a JAVA - Ranking



Entorno de desarrollo



Ambiente de Desarrollo

Configurar el entorno de desarrollo para Java es un paso crucial para comenzar a programar de manera eficiente. A continuación se detallan los componentes esenciales y las herramientas que necesitarás:

JDK (Java Development Kit)

- **Versión Recomendable: JDK v21** El JDK es el kit de desarrollo necesario para compilar y ejecutar programas Java. Incluye el compilador (javac), el intérprete/loader (java), y otras herramientas como el depurador (jdb) y el archivador (jar). La versión 21 es una de las versiones LTS (Long-Term Support), lo que significa que tendrá soporte extendido y es ideal para proyectos de largo plazo.

Ambiente de Desarrollo

IDEs (Integrated Development Environments)

- **Eclipse** Eclipse es una IDE de código abierto muy popular entre los desarrolladores Java. Ofrece una amplia gama de plugins y herramientas que facilitan el desarrollo, depuración y despliegue de aplicaciones Java. Es especialmente útil para proyectos grandes debido a su capacidad de gestión de proyectos y su integración con herramientas de desarrollo y sistemas de control de versiones como Git.
- **IntelliJ IDEA** IntelliJ IDEA es una IDE desarrollada por JetBrains que es conocida por su inteligencia y robustez. Ofrece características avanzadas como autocompletado inteligente, refactorización y análisis de código en tiempo real. IntelliJ IDEA está disponible en una versión comunitaria gratuita y una versión comercial con características adicionales. Es altamente recomendada para desarrolladores que buscan un entorno de desarrollo potente y eficiente.
- **NetBeans** NetBeans es otra IDE de código abierto que proporciona un entorno de desarrollo completo para Java. Además de ofrecer soporte para desarrollo Java SE y Java EE, NetBeans es excelente para trabajar con otros lenguajes como PHP, HTML5, y C/C++. Su integración con servidores de aplicaciones y herramientas de base de datos lo hace ideal para el desarrollo de aplicaciones web y empresariales.
- **Visual Studio Code** Visual Studio Code, desarrollado por Microsoft, es un editor de código ligero pero potente que soporta Java a través de extensiones. Aunque no es una IDE tradicional como las anteriores, su flexibilidad y la gran cantidad de extensiones disponibles permiten una experiencia de desarrollo Java muy personalizable. Es especialmente útil para desarrolladores que trabajan con múltiples lenguajes de programación y tecnologías.

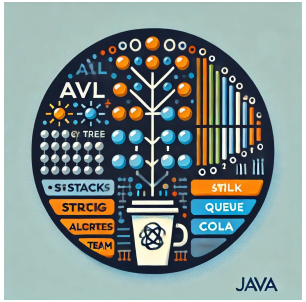
Pasos para Configurar el Entorno de Desarrollo Java

1. **Descargar e Instalar el JDK**
   - Visita el sitio web oficial de Oracle o Adoptium para descargar la versión adecuada del JDK.
   - Sigue las instrucciones de instalación específicas para tu sistema operativo (Windows, macOS, Linux).
   - Configura la variable de entorno `JAVA_HOME` y actualiza el `PATH` del sistema para incluir el directorio `bin` del JDK.
2. **Elegir e Instalar una IDE**
   - Descarga la IDE de tu elección desde sus sitios oficiales:
     - Eclipse: eclipse.org
     - IntelliJ IDEA: jetbrains.com/idea
     - NetBeans: netbeans.apache.org
     - Visual Studio Code: code.visualstudio.com
   - Sigue las instrucciones de instalación proporcionadas.
3. **Configurar la IDE**
   - Abre la IDE y configura el JDK en las preferencias o configuraciones de la misma.
   - Instala cualquier plugin o extensión necesaria para tu flujo de trabajo específico (por ejemplo, soporte para Maven, Gradle, etc.).
   - Configura tus proyectos y entornos de trabajo según las necesidades de tu desarrollo.
4. **Verificar la Instalación**
   - Crea y ejecuta un proyecto Java sencillo para asegurar que todo está configurado correctamente. Un programa "Hello World" es una buena manera de verificar que tu entorno de desarrollo está funcionando correctamente.

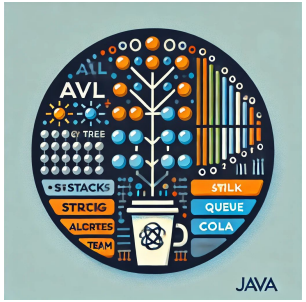
Con estos pasos, estarás listo para comenzar a desarrollar aplicaciones en Java de manera eficiente y productiva.

Visual Studio

Probemos nuestro IDE



Programación Orientada a Objetos



Objetos

Son instancias de clases que agrupan datos y comportamientos.

Los objetos representan **entidades del mundo real o conceptos dentro de un programa** y poseen propiedades (atributos) y métodos (funciones) que definen sus comportamientos.

Clases

Son **plantillas o modelos** a partir de los cuales se crean los objetos.

Una clase define un conjunto de atributos y métodos que los objetos de esa clase compartirán.

Objetos & Clases

```
// Definición de la clase
class Coche {
    // Atributos
    String color;
    String marca;
    int velocidad;

    // Método
    void acelerar(int incremento) {
        velocidad += incremento;
    }
}
```

```
// Uso de la clase (creación de un objeto)
public class Main {
    public static void main(String[] args) {
        // Creación de un objeto
        Coche miCoche = new Coche();
        miCoche.color = "Rojo";
        miCoche.marca = "Toyota";
        miCoche.velocidad = 0;

        // Llamada a un método
        miCoche.acelerar(50);
        System.out.println("Velocidad: " +
miCoche.velocidad);
    }
}
```

Atributos y Métodos

Las clases contienen atributos (variables) que representan las características de un objeto y métodos (funciones) que determinan su comportamiento.

Encapsulamiento

Consiste en ocultar los detalles internos de un objeto y **exponer solo lo necesario** a través de métodos públicos.

Esto mejora la modularidad y la protección de los datos.

Encapsulamiento

```
class Coche {
    // Atributos privados
    private String color;
    private int velocidad;

    // Métodos públicos para acceder a los atributos
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public int getVelocidad() {
        return velocidad;
    }
    public void acelerar(int incremento) {
        this.velocidad += incremento;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Coche miCoche = new Coche();
        miCoche.setColor("Azul");
        miCoche.acelerar(60);
        System.out.println("Color: " +
miCoche.getColor());
        System.out.println("Velocidad: " +
miCoche.getVelocidad());
    }
}
```

Herencia

Permite crear nuevas clases a partir de clases existentes, heredando atributos y métodos, y permitiendo agregar o modificar funcionalidades.

Esto fomenta la reutilización del código.

Herencia

```
// Clase base
class Coche {
    String marca;
    int velocidad;

    void acelerar(int incremento) {
        velocidad += incremento;
    }
}

// Clase derivada
class CocheDeportivo extends Coche {
    int turbo;

    void activarTurbo() {
        velocidad += turbo;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        CocheDeportivo miCoche = new
CocheDeportivo();
        miCoche.marca = "Ferrari";
        miCoche.turbo = 50;
        miCoche.acelerar(100);
        miCoche.activarTurbo();
        System.out.println("Velocidad: " +
miCoche.velocidad);
    }
}
```

Polimorfismo

Se refiere a la capacidad de diferentes objetos de responder al mismo mensaje (llamada de método) de manera específica según su clase.

Facilita la flexibilidad y extensibilidad del código.

Polimorfismo

```
class Coche {
    void arrancar() {
        System.out.println("El coche está arrancando");
    }
}
class CocheElectrico extends Coche {
    @Override
    void arrancar() {
        System.out.println("El coche eléctrico está
arrancando en silencio");
    }
}
class CocheDeportivo extends Coche {
    @Override
    void arrancar() {
        System.out.println("El coche deportivo está
rugiendo al arrancar");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Coche miCoche = new Coche();
        Coche miCocheElectrico = new CocheElectrico();
        Coche miCocheDeportivo = new
CocheDeportivo();

        miCoche.arrancar();
        miCocheElectrico.arrancar();
        miCocheDeportivo.arrancar();
    }
}
```

Abstracción

Es el proceso de simplificar la complejidad ocultando los detalles innecesarios y resaltando solo los aspectos esenciales de un objeto.

Esto ayuda a gestionar la complejidad del software.

Abstracción

```
// Clase abstracta
abstract class Figura {
    abstract void dibujar(); // Método abstracto
}

class Circulo extends Figura {
    void dibujar() {
        System.out.println("Dibujando un círculo");
    }
}

class Cuadrado extends Figura {
    void dibujar() {
        System.out.println("Dibujando un cuadrado");
    }
}
```

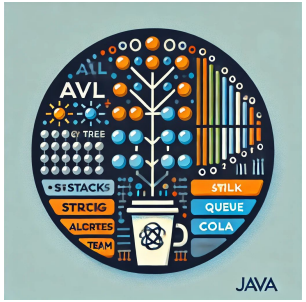
```
public class Main {
    public static void main(String[] args) {
        Figura figura1 = new Circulo();
        Figura figura2 = new Cuadrado();

        figura1.dibujar();
        figura2.dibujar();
    }
}
```

Interface

```
// Clase abstracta
interface Figura {
    void dibujar(); // Método abstracto
}

class Circulo implements Figura {
    void dibujar() {
        System.out.println("Dibujando un círculo");
    }
}

class Cuadrado implements Figura {
    void dibujar() {
        System.out.println("Dibujando un cuadrado");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Figura figura1 = new Circulo();
        Figura figura2 = new Cuadrado();

        figura1.dibujar();
        figura2.dibujar();
    }
}
```

¡Gracias!


AVL
AVL TREE
STACKS
STRCIG
ALCRTES
TEAM
STILK
QUEUE
COLA
JAVA


CENTRO REGIONAL UNIVERSITARIO CORDOBA UTN
1941