ISTRED
INSTITUTO SUPERIOR TÉCNICO REY DAVID

**PERIODO I - 2024 - 2025**

# GUÍA DEL APRENDIZAJE DEL ESTUDIANTE

## ELEMENTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS

**ING. FERNANDO DOMINGUEZ RAMOS**

GESTIÓN DE BASE DE DATOS
2DO. SEMESTRE

**Contenido**

I. DATOS GENERALES DE LA ASIGNATURA ..... 8

Información General ..... 8

Requisitos ..... 8

Descripción de la asignatura. ..... 8

Objetivos de la asignatura: ..... 9

Introducción de la Asignatura ..... 9

I. DATOS GENERALES DE LA UNIDAD 1 ..... 11

DESARROLLO ..... 12

Unidad 1: Introducción a la Programación Orientada a Objetos ..... 12

Tema 1: Introducción a la Programación Orientada a Objetos (POO) ..... 12

Historia de la Programación Orientada a Objetos ..... 14

Clases y Objetos en Programación Orientada a Objetos ..... 18

ACTIVIDADES PRÁCTICAS ..... 20

FORO DE DEBATE ..... 21

PRÁCTICA INVESTIGATIVA ..... 21

PRÁCTICA TEST ..... 22

Tema 2: Constructor y Sobrecarga de Constructores en Programación Orientada a Objetos ..... 23

Conceptualización: ..... 23

Definiciones: ..... 23

Ejercicios Prácticos: ..... 24

Preguntas de Repaso: ..... 24

Herencia en Programación Orientada a Objetos ..... 25

Encapsulamiento en Programación Orientada a Objetos ..... 26

Ejemplos: ..... 26

Sobrecarga de Métodos en Programación Orientada a Objetos ..... 27

Ejemplos: ..... 27

Preguntas de Repaso: ..... 28

Polimorfismo, Interfaces, clases abstractas en Programación Orientada a Objetos ..... 28

Definiciones: ..... 28

Uso de Polimorfismo en Java: ..... 29

Ejercicios Prácticos: ..... 29

Interfaces y Clases Abstractas en Programación Orientada a Objetos ..... 30

Ejemplos: ..... 30

Preguntas de Repaso: ..... 31

ACTIVIDADES PRÁCTICAS ..... 32

FORO DE DEBATE ..... 33

PRÁCTICA INVESTIGATIVA ..... 33

PRÁCTICA TEST ..... 34

II. DATOS GENERALES DE LA UNIDAD 2 ..... 35

Información General ..... 35

Resultado de Aprendizaje ..... 35

Metodología ..... 35

DESARROLLO ..... 36

Unidad 2: Controles básicos y lenguaje de programación Java entorno Netbeans ..... 36

Tema 1: Entorno de IDE de POO ..... 36

Conceptualización: ..... 36

Entorno de NETBANS, cuadro de herramientas, Control ingresos y salidas de textos. Propiedades y Eventos ..... 39

Ejemplo ..... 47

Control JButton y JOption Button. Propiedades y Eventos ..... 49

Propiedades en NetBeans: ..... 51

Mensajes de error. ..... 54

ACTIVIDADES PRÁCTICAS ..... 55

FORO DE DEBATE ..... 56

PRÁCTICA INVESTIGATIVA ..... 56

PRÁCTICA TEST ..... 57

Tema 2: Objetos de Presentación de Información ..... 58

Conceptualización: ..... 58

Entorno de NETBANS, cuadro de herramientas, Control ingresos y salidas de textos. Propiedades y Eventos ..... 62

Ejemplo ..... 69

Estructura del Lenguaje de Programación en JAVA ..... 72

Uso de la Instrucción If-ElseIf-EndIf en JAVA ..... 73

Instrucciones Select-Switch ..... 74

ACTIVIDADES PRÁCTICAS ..... 75

FORO DE DEBATE ..... 76

PRÁCTICA INVESTIGATIVA ..... 76
PRÁCTICA TEST ..... 77
III. DATOS GENERALES DE LA UNIDAD 3 ..... 78
Información General ..... 78
Resultado de Aprendizaje ..... 78
Metodología ..... 78
DESARROLLO ..... 79
Unidad 3: Estructuras de control y bucles en Java ..... 79
Tema 1: Estructuras de Control ..... 79
Introducción Ciclos repetitivos ..... 79
Uso de contadores y acumuladores en Java ..... 81
Uso de While y Do While en Java ..... 83
Comparación y Elección de Estructura ..... 85
Sentencias for y ForEach iteraciones ..... 86
ACTIVIDADES PRÁCTICAS ..... 92
FORO DE DEBATE ..... 93
PRÁCTICA INVESTIGATIVA ..... 93
PRÁCTICA TEST ..... 94
Tema 2: Variables y Métodos en POO ..... 95
Definición y Propósito: ..... 95
Sintaxis de Declaración: ..... 95
Comparación y Contraste: ..... 96

Declaración de Variables, Métodos Funciones Constructores, Polimorfismo y Herencia en Programación Orientada a Objetos en Java ..... 97
Comparación y Contraste: ..... 100
Variables vs. Métodos: ..... 100
Funciones Constructores vs. Métodos: ..... 100
Polimorfismo vs. Herencia: ..... 100
Resumen: ..... 100
Declaración y Creación de Clases, Lógica de negocio con Listas de Objetos. 100
Ejemplo: ..... 100
Creación de Objetos: ..... 101
Lógica de Negocio con Listas de Objetos: ..... 101
Comparación y Contraste: ..... 102
Clases vs. Objetos: ..... 102
Creación de Objetos vs. Lógica de Negocio: ..... 102
ACTIVIDADES PRÁCTICAS ..... 104
FORO DE DEBATE ..... 105
PRÁCTICA INVESTIGATIVA ..... 105
PRÁCTICA TEST ..... 106
IV. DATOS GENERALES DE LA UNIDAD 4 ..... 107
Información General ..... 107
Resultado de Aprendizaje ..... 107
Metodología ..... 107
DESARROLLO ..... 108

Unidad 4: Diseño de interfaces en Swing ..... 108
Tema 1: Introducción a la librería Swing ..... 108
Introducción a la Librería Swing ..... 108
Componentes Básicos: ..... 108
Objeto JFrame y JOptionPanel ..... 111
Comparación y Contraste: ..... 114
JPanel, JLabel, JTextField ..... 115
JTextField: ..... 117
Comparación y Contraste: ..... 117
ACTIVIDADES PRÁCTICAS ..... 119
FORO DE DEBATE ..... 120
PRÁCTICA INVESTIGATIVA ..... 120
PRÁCTICA TEST ..... 121
Tema 2: Objetos de listas en Java ..... 122
Métodos Comunes de Listas ..... 122
Comparación con Otros Tipos de Listas ..... 123
JButton, JList, JComboBox ..... 124
JList ..... 125
JComboBox ..... 126
Comparación y Contraste ..... 127
JButton vs. JList vs. JComboBox ..... 127
Acciones y Eventos ..... 127
JMenu, JMenuBar, JScrollBar ..... 128

Comparación y Contraste ..... 131

ACTIVIDADES PRÁCTICAS ..... 132

FORO DE DEBATE ..... 133

PRÁCTICA INVESTIGATIVA ..... 133

PRÁCTICA TEST ..... 134

## I. DATOS GENERALES DE LA ASIGNATURA

### Información General

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.1. | Asignatura | Elementos de Programación Orientada a Objetos | | | | | |
| 1.2. | Carrera | Gestión de Base de Datos | | | | | |
| 1.3. | Código de asignatura | GBD22310 | | | | | |
| 1.4. | Créditos | 1,2 | | | | | |
| 1.5. | Nivel | Segundo | | | | | |
| 1.6. | Detalle de horas | ACD | 24 | AA | 16 | APE | 20 |

### Requisitos

| Prerrequisitos | | Correquisitos | |
|---|---|---|---|
| Asignatura | Código | Asignatura | Código |
| N/A | N/A | N/A | N/A |

### Descripción de la asignatura.

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en la organización de un programa en torno a objetos, que son instancias de clases. En POO, un objeto es una unidad independiente que contiene datos y métodos relacionados. Cada objeto puede interactuar con otros objetos para lograr la funcionalidad deseada del programa. Este enfoque se basa en los conceptos de encapsulación, herencia y polimorfismo.

La encapsulación es un principio fundamental de la POO que se refiere a la ocultación de los detalles internos de un objeto y la exposición de una interfaz pública. Esto permite mantener la seguridad y la integridad de los datos, evitando el acceso no autorizado. La herencia es otro concepto importante que permite la creación de nuevas clases a partir de clases existentes, heredando sus atributos y métodos. Esto promueve la reutilización de código y la creación de jerarquías de clases. Por último, el polimorfismo permite que objetos de diferentes clases se comporten de manera similar, lo que facilita la flexibilidad y la extensibilidad del código.

La Programación Orientada a Objetos es ampliamente utilizada en el desarrollo de software moderno debido a su capacidad para modelar problemas de manera más

natural y estructurada. Proporciona una forma efectiva de organizar y gestionar el código, lo que resulta en un desarrollo más mantenible y escalable. Comprender los conceptos de la POO es esencial para los programadores que desean crear aplicaciones robustas y flexibles.

**Objetivos de la asignatura:**

El objetivo general de la asignatura de Programación Orientada a Objetos es proporcionar a los estudiantes una comprensión profunda de los conceptos y principios fundamentales de la Programación Orientada a Objetos, así como las habilidades necesarias para aplicar estos conceptos en el desarrollo de software.

Comprender los conceptos clave de la Programación Orientada a Objetos: Los estudiantes deberán adquirir un conocimiento sólido de los principios básicos de la POO, como la encapsulación, la herencia, el polimorfismo y la abstracción. Deben ser capaces de explicar estos conceptos y aplicarlos en la resolución de problemas de programación.

Desarrollar habilidades prácticas en la creación de software orientado a objetos: Los estudiantes deben ser capaces de diseñar, implementar y depurar programas utilizando lenguajes de programación que admiten la POO, como Java, C++, o Python. Deben estar familiarizados con la creación de clases y objetos, la gestión de herencia y la implementación de interfaces.

Aplicar la Programación Orientada a Objetos en proyectos reales: Los estudiantes deberán ser capaces de aplicar los conceptos de POO en proyectos de desarrollo de software del mundo real. Deben ser capaces de analizar problemas, diseñar soluciones orientadas a objetos y trabajar en equipo para desarrollar aplicaciones robustas y mantenibles utilizando metodologías de desarrollo ágiles.

**Introducción de la Asignatura**

La Programación Orientada a Objetos (POO) es una disciplina fundamental en el mundo de la informática y el desarrollo de software. Esta asignatura introduce a los estudiantes en un paradigma de programación que ha revolucionado la forma en que se diseña y construye software en la actualidad. La POO se basa en la idea de que los programas pueden modelarse como una colección de objetos interconectados, cada uno de los cuales encapsula datos y funcionalidad. A través de la asignatura de Programación Orientada a Objetos, los estudiantes tendrán la oportunidad de explorar

los conceptos y principios esenciales que sustentan este enfoque de programación y desarrollar las habilidades necesarias para aplicarlos de manera efectiva en la resolución de problemas y el desarrollo de aplicaciones.

En el transcurso de esta asignatura, los estudiantes se adentrarán en los pilares fundamentales de la POO, como la encapsulación, la herencia y el polimorfismo. Aprenderán a diseñar y crear clases y objetos, lo que les permitirá construir programas más organizados, reutilizables y mantenibles. Además, se familiarizarán con lenguajes de programación orientada a objetos ampliamente utilizados, como Java y C++, y ganarán experiencia en la implementación de soluciones prácticas en proyectos de software del mundo real. La Programación Orientada a Objetos no solo es una habilidad técnica esencial, sino también un enfoque conceptual que fomenta la resolución de problemas de manera eficiente y efectiva, lo que la convierte en una competencia valiosa para cualquier futuro profesional de la informática o el desarrollo de software.

## I. DATOS GENERALES DE LA UNIDAD 1

### Información General

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.1. | Nivel | Segundo | | | | | |
| 1.2. | Nombre Unidad | Introducción a la Programación Orientada a Objetos | | | | | |
| 1.3. | Tema 1 – Unidad | Introducción a la Programación Orientación a Objetos | | | | | |
| 1.4. | Tema 2 – Unidad | Constructor y Sobrecarga de Constructores | | | | | |
| 1.5. | Unidad organizacional | Unidad Profesional | | | | | |
| 1.6. | Unidad | 1 | | | | | |
| 1.7. | Total, Horas Unidad | 15 | | | | | |
| 1.8. | Detalle de horas Unidad | ACD | 6 | AA | 4 | APE | 5 |

### Resultado de Aprendizaje

Al finalizar esta unidad, los estudiantes serán capaces de comprender y aplicar las estructuras de control fundamentales en Java, incluyendo if, else, if-else if, y switch, para tomar decisiones condicionales.

### Metodología

La Metodología de se basa en un enfoque activo y colaborativo. Incluye sesiones de clase estructuradas en torno a temas específicos, lecturas y tareas, estudio de casos, proyectos prácticos y debates en clase. Los estudiantes también se involucran en trabajos en grupo y mantienen diarios de reflexión. Se fomenta la participación en debates y preguntas y respuestas, y se proporcionan recursos en línea para una comprensión más profunda. La evaluación se realiza mediante pruebas, proyectos y una evaluación final, con retroalimentación continua para guiar el aprendizaje. Esta metodología busca que los estudiantes comprendan los conceptos fundamentales y puedan aplicar eficazmente las metodologías de desarrollo de software en situaciones prácticas.

**DESARROLLO**

**Unidad 1: Introducción a la Programación Orientada a Objetos**

**Tema 1: Introducción a la Programación Orientada a Objetos (POO)**

**Conceptualización**:

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en la creación y manipulación de objetos, los cuales son instancias de clases. Se centra en la organización del código en torno a objetos que tienen propiedades (atributos) y comportamientos (métodos).

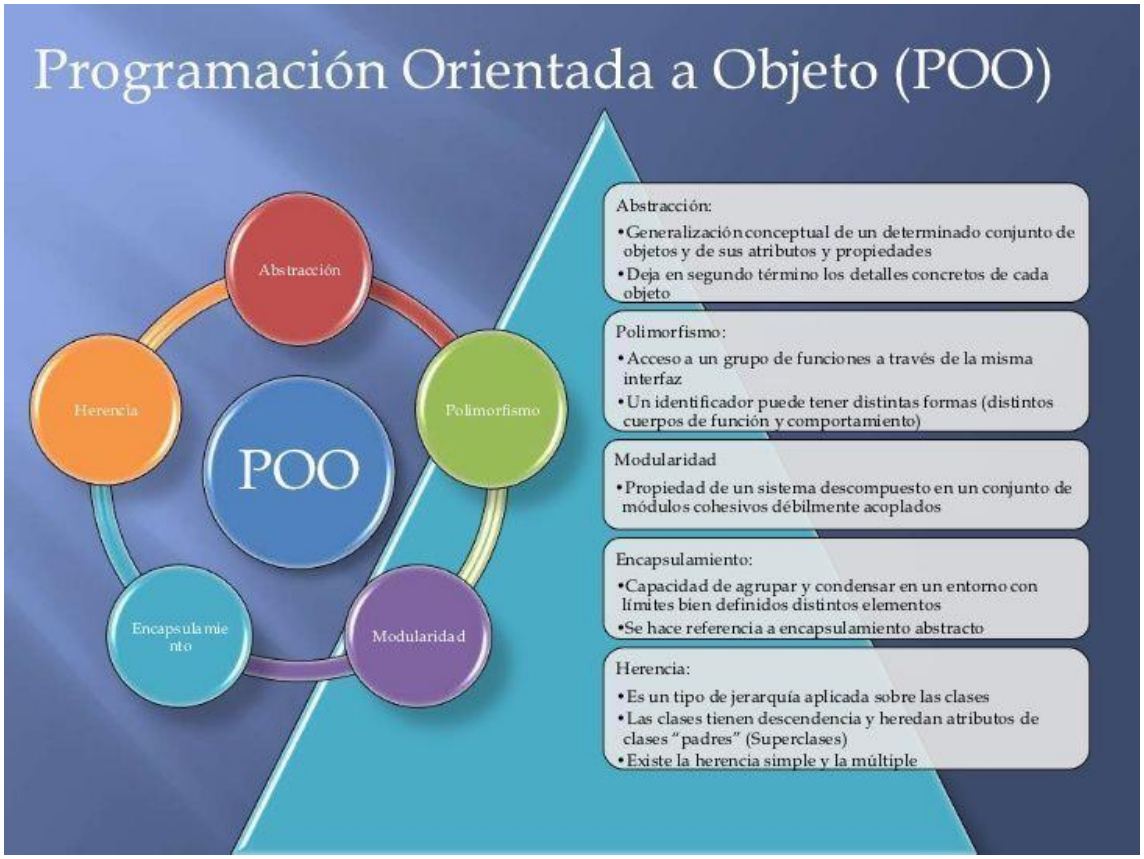



**Imagen 1: Conceptos Básicos de POO**

**Fuente**: https://issuu.com/jesusdavidmaldonadogrisales3/docs/revista_2_junio_2023

**Definiciones**

**Clase**: Una plantilla que define la estructura de un objeto.

**Objeto**: Una instancia de una clase que contiene datos y métodos para operar sobre esos datos.

**Encapsulación**: El ocultamiento de los detalles internos de una clase y la exposición de una interfaz pública.

**Herencia**: El mecanismo que permite que una clase herede propiedades y métodos de otra.

**Polimorfismo**: La capacidad de objetos de diferentes clases de responder a un mismo mensaje de manera diferente.

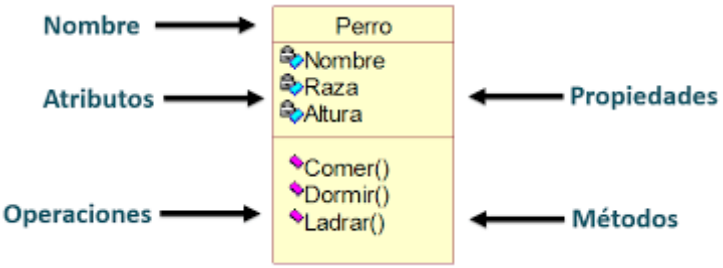



**Imagen 2: Conceptos Básicos de POO**

**Fuente**: https://issuu.com/jesusdavidmaldonadogrisales3/docs/revista_2_junio_2023

**Ejemplo**: Creación de una clase en Java.

```
public class Persona {
    String nombre;
    int edad;
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    public void saludar() {
        System.out.println("Hola, soy " + nombre + " y tengo " + edad + " años.");
```

```
    }
}
```

Caso de estudio: Desarrollo de una aplicación de gestión de biblioteca que utiliza clases como "Libro", "Autor" y "Biblioteca" para modelar los datos y operaciones.

**Historia de la Programación Orientada a Objetos**

**Conceptualización**: La POO se originó en la década de 1960 y se ha convertido en uno de los paradigmas de programación más utilizados. Fue desarrollada para abordar la complejidad de los sistemas de software al organizar el código en entidades autónomas y reutilizables.

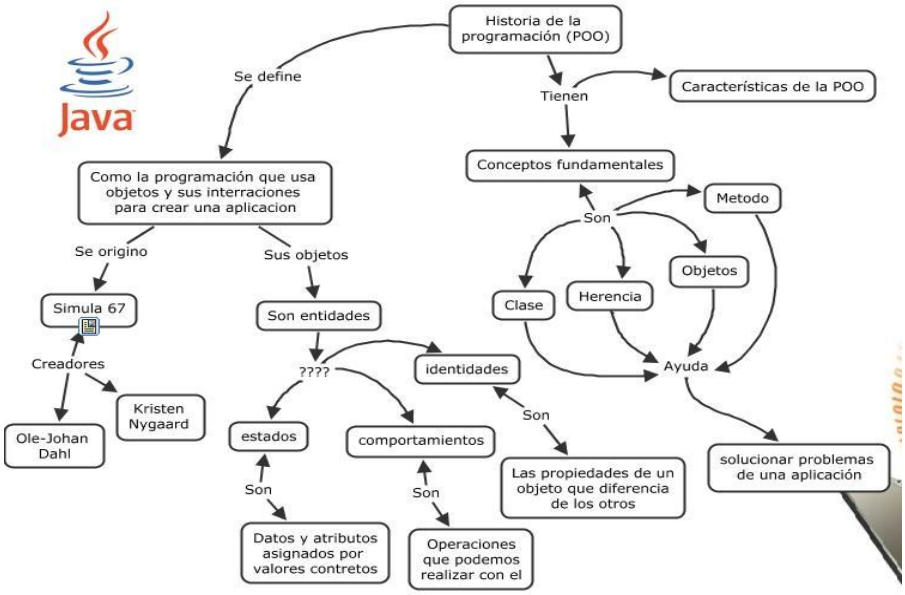



**Imagen 3: Historia de POO**

**Fuente**: https://maaz.ihmc.us/rid=1K2C1D2MH-2701JTD-VBH/Historia%20de%20POO.cmap

**Década de 1960:**

Los orígenes de la POO se pueden rastrear hasta el lenguaje de programación Simula, desarrollado por Ole-Johan Dahl y Kristen Nygaard en Noruega en la década de 1960. Simula introdujo el concepto de "clases" y "objetos" para modelar sistemas de simulación.

La palabra "clase" se utilizó por primera vez para describir una plantilla para la creación de objetos.

**Década de 1970:**

Alan Kay, un científico de la computación estadounidense, desarrolló el lenguaje de programación Smalltalk en los Laboratorios Xerox PARC. Smalltalk es ampliamente considerado como el primer lenguaje de programación orientado a objetos puro. Introdujo conceptos fundamentales de la POO, como el polimorfismo y la herencia.

Smalltalk tenía una influencia significativa en el desarrollo futuro de la POO.

**Década de 1980:**

Bjarne Stroustrup, un ingeniero de software danés, desarrolló C++ como una extensión del lenguaje de programación C. C++ incorporó conceptos de la POO, como clases, objetos, herencia y polimorfismo. Esto marcó la transición de la programación estructurada a la programación orientada a objetos.

C++ se convirtió en un lenguaje ampliamente utilizado para el desarrollo de sistemas y aplicaciones de software.

**Década de 1990:**

La POO se volvió más popular y se adoptó ampliamente en la industria del software. Lenguajes como Java y C# se desarrollaron con un enfoque en la POO y se convirtieron en lenguajes de programación populares en la programación empresarial y la programación en red.

La popularidad de la POO se vio impulsada por conceptos como encapsulación, reutilización de código y facilidad de mantenimiento.

**Década de 2000 y más allá:**

La POO siguió evolucionando con la introducción de marcos y bibliotecas orientados a objetos en una variedad de lenguajes.

También se han desarrollado lenguajes y plataformas más modernos que fomentan aún más los principios de la POO, como Python, Ruby y JavaScript (a través de ECMAScript 6).

En resumen, la Programación Orientada a Objetos ha experimentado un desarrollo constante a lo largo de las décadas y ha influido en muchos aspectos de la programación y el desarrollo de software. Los principios de la POO han demostrado ser efectivos para la creación de sistemas complejos, y su importancia sigue siendo relevante en la actualidad.

**Ventajas de la Programación Orientada a Objetos**

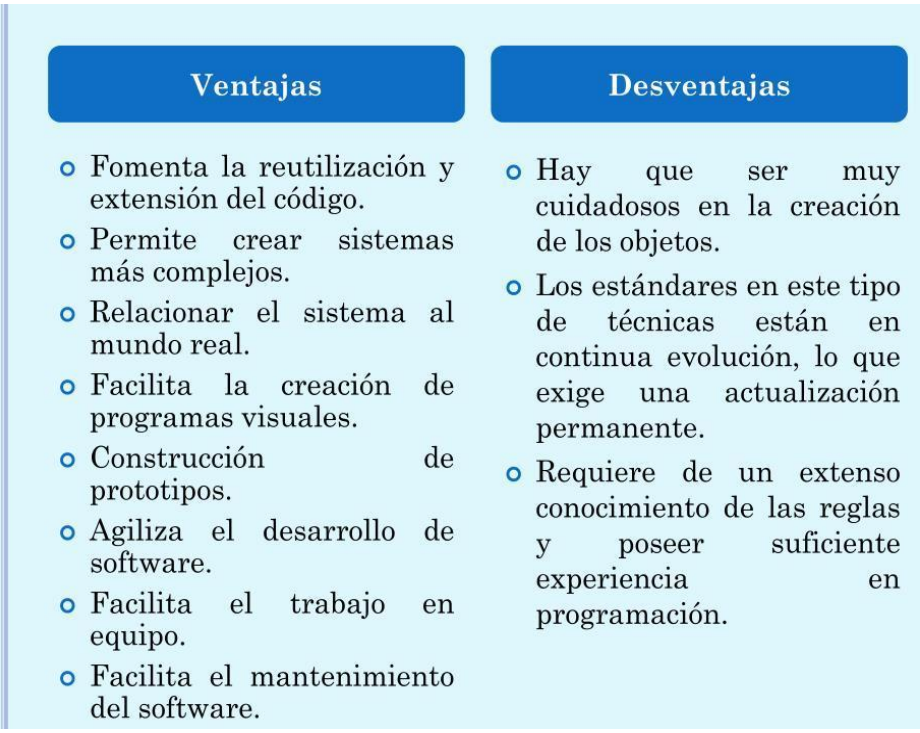



**Imagen 4: Ventajas y Desventajas de POO**

**Fuente**: https://www.slideserve.com/lionel/unidad-iii-programaci-n-orientada-a-objetos-poo

**Conceptualización**: La POO ofrece diversas ventajas en el desarrollo de software, Reutilización de código (Herencia y Composición): La POO permite la reutilización de código de manera eficiente. Dos mecanismos clave para lograr esto son la herencia y la composición. La herencia permite que una clase herede propiedades y métodos de otra, lo que facilita la creación de nuevas clases basadas en clases existentes. La composición permite construir objetos complejos combinando múltiples objetos más pequeños.

**Modularidad**: La POO fomenta la creación de unidades de código independientes y cohesivas llamadas objetos. Cada objeto encapsula datos y comportamientos relacionados, lo que facilita la organización y el mantenimiento del código.

Encapsulación: La encapsulación implica ocultar los detalles internos de un objeto y proporcionar una interfaz pública para interactuar con él. Esto mejora la seguridad y la integridad del código, ya que los datos privados solo son accesibles a través de métodos específicos.

**Facilita el mantenimiento:** La POO hace que el mantenimiento del código sea más manejable. Cambiar el comportamiento de un objeto implica modificar solo la clase relacionada, sin afectar otras partes del programa. Esto reduce el riesgo de efectos secundarios no deseados.

**Flexibilidad y escalabilidad:** La POO permite que los sistemas se escalen más fácilmente. Se pueden agregar nuevas clases y objetos sin modificar el código existente. Esto facilita la adaptación a las cambiantes necesidades del software.

**Polimorfismo**: El polimorfismo permite que objetos de diferentes clases respondan de manera similar a un conjunto común de mensajes o métodos. Esto simplifica la escritura de código genérico y flexible, lo que facilita la gestión de objetos de distintos tipos.

**Abstracción**: La abstracción es la capacidad de simplificar sistemas complejos mediante la creación de modelos conceptuales. En la POO, las clases actúan como abstracciones de entidades del mundo real, lo que facilita la comprensión y la resolución de problemas.

**Mejora la colaboración en equipo**: La POO se presta a un enfoque de desarrollo colaborativo, ya que los desarrolladores pueden trabajar en clases y objetos independientes. Además, se pueden definir interfaces claras entre los componentes del sistema.

**Facilita la depuración**: La organización modular y la encapsulación hacen que la depuración sea más sencilla. Los errores tienden a estar confinados en las clases específicas, lo que simplifica la identificación y corrección de problemas.

**Reusabilidad de bibliotecas y componentes**: La POO fomenta la creación de bibliotecas de clases y componentes reutilizables. Esto ahorra tiempo y esfuerzo, ya que los desarrolladores pueden aprovechar soluciones preexistentes.

En resumen, la Programación Orientada a Objetos es un enfoque poderoso y versátil que ofrece numerosas ventajas en términos de organización del código, reutilización, mantenimiento y flexibilidad. Estas ventajas contribuyen a la creación de sistemas de software más robustos y fáciles de desarrollar y mantener.

**Ejemplo**: Comparación de un programa no orientado a objetos con uno orientado a objetos para mostrar la claridad y la reutilización de código.

**Clases y Objetos en Programación Orientada a Objetos**

**Conceptualización**:

La Programación Orientada a Objetos se basa en la creación y manipulación de objetos.

Las "clases" son plantillas o modelos para crear objetos, mientras que los "objetos" son instancias concretas de esas clases.

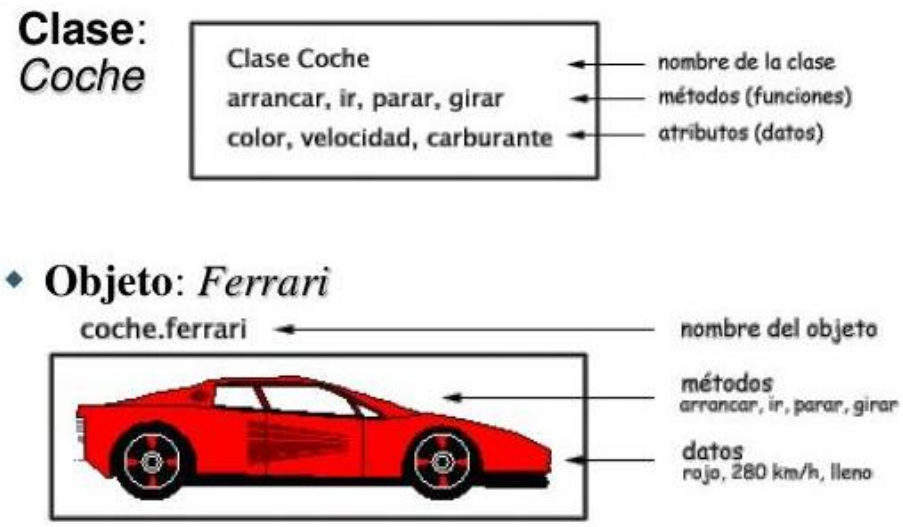



**Imagen 5: Clases y Objetos de POO**

**Fuente**: https://aprendiendoarduino.wordpress.com/2019/09/11/clases-y-objetos-2/

**Definiciones**:

**Clase**: Una clase es una estructura que define un conjunto de propiedades y comportamientos que tendrán los objetos creados a partir de ella. Es como un plano o un molde para los objetos.

**Objeto**: Un objeto es una instancia concreta de una clase. Contiene datos (propiedades o atributos) y métodos (funciones o comportamientos) específicos.

**Ejemplos**:

**Definición de una Clase en Java:**

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

**Creación de Objetos en Java:**

```
Persona persona1 = new Persona();
Persona persona2 = new Persona();
```

**Asignación de Propiedades a un Objeto en Java:**

```
persona1.nombre = "Juan";
persona1.edad = 30;
```

**Ejercicios Prácticos:**

Crea una clase llamada "Coche" con propiedades como "marca", "modelo" y "año". Luego, crea dos objetos de la clase "Coche" e inicialízalos con información de coches reales.

Extiende la clase "Persona" para incluir un método llamado "saludar" que muestre un saludo personalizado con el nombre y la edad de la persona.

**Preguntas de Repaso:**

¿Cuál es la diferencia entre una clase y un objeto en la POO?

¿Qué representan las propiedades de un objeto?

¿Qué son los métodos en una clase y cómo se utilizan en objetos?

## ACTIVIDADES PRÁCTICAS

**Tabla**

Detalle de la Práctica

| Practicas | Instrucciones | Duración de la Practica | Mecanismo de Evaluación |
|---|---|---|---|
| **Foro de debate** | Revisar Actividades Prácticas | 0:30 mm | Por rúbricas |
| **Práctica investiga: Taller/Ensayo/Investigación** | Revisar Actividades Prácticas | 3:30 h | Por rúbricas |
| **Tutorías Prácticas** | Revisar Actividades Prácticas | 2:00 mm | Por participación en clase |
| **Test evaluativo** | Revisar Actividades Prácticas | 0:30 mm | Formativa |

*Nota.* Elaboración propia

**Consideraciones Generales**

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

**FORO DE DEBATE**

**Tema:** Define tres atributos del objeto carro y define con tus palabras por qué los describiste.

**Instrucciones para Participar:**

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

**PRÁCTICA INVESTIGATIVA**

No aplica para este tema.

**Consigna:**

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

**PRÁCTICA TEST**

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

**Consideraciones para realizar el test**

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

**Tema 2: Constructor y Sobrecarga de Constructores en Programación Orientada a Objetos**

**Conceptualización:**

Un "constructor" es un método especial en una clase que se llama cuando se crea un objeto de esa clase. Su función principal es inicializar los atributos del objeto.

La "sobrecarga de constructores" es la capacidad de una clase de tener varios constructores con diferentes parámetros, lo que permite crear objetos de diferentes maneras.

**Definiciones:**

**Constructor**: Un constructor es un método con el mismo nombre que la clase y no tiene tipo de retorno. Se utiliza para inicializar los atributos de un objeto cuando se crea.

Sobrecarga de Constructores: La sobrecarga de constructores implica tener varios constructores en una clase con diferentes listas de parámetros. Los constructores pueden ser llamados con diferentes argumentos para crear objetos de diversas formas.

**Ejemplos**:

Constructor Simple en Java:

```
public class Coche {

    String marca;

    String modelo;



    public Coche() {

        marca = "Desconocida";

        modelo = "Desconocido";

    }
```

```
}
```

**Sobrecarga de Constructores en Java:**

```
public class Estudiante {
    String nombre;
    int edad;
    public Estudiante(String nombre) {
        this.nombre = nombre;
        edad = 0; // Inicializa la edad en 0 por defecto
    }
    public Estudiante(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}
```

**Ejercicios Prácticos:**

Crea una clase "Libro" con atributos como "título", "autor" y "año de publicación". Implementa constructores para permitir la creación de objetos de libros con diferentes conjuntos de información.

Define una clase "Círculo" con atributos "radio" y "color". Crea constructores que permitan crear círculos con diferentes radios y colores.

**Preguntas de Repaso:**

¿Qué es un constructor y cuál es su función principal en la POO?

¿Qué es la sobrecarga de constructores y cómo se implementa en una clase?

¿Cuándo es útil utilizar la sobrecarga de constructores en un programa?

**Herencia en Programación Orientada a Objetos**

**Conceptualización**:

La "herencia" es un mecanismo en la POO que permite que una clase herede propiedades y métodos de otra clase.

En la herencia, una clase base (o superclase) se utiliza como punto de partida para crear una nueva clase derivada (o subclase).

**Clase Base (Superclase):** La clase que proporciona propiedades y métodos que se pueden heredar.

**Clase Derivada (Subclase):** La clase que hereda propiedades y métodos de la clase base y puede agregar sus propias propiedades y métodos.

**Ejemplos**:

**Herencia en Java:**

```
public class Figura {
    int ancho;
    int alto;

    public void mostrarArea() {
        System.out.println("Área desconocida");
    }
}
public class Rectangulo extends Figura {
    public void mostrarArea() {
        int area = ancho * alto;
        System.out.println("Área del rectángulo: " + area);
```

```
    }
}
```

**Encapsulamiento en Programación Orientada a Objetos**

El "encapsulamiento" es un principio de la POO que consiste en ocultar los detalles internos de un objeto y proporcionar una interfaz pública para interactuar con él.

Esto se logra mediante la declaración de atributos como privados y la exposición de métodos públicos para acceder y modificar esos atributos.

**Atributos Privados:** Los atributos de una clase que solo pueden ser accedidos o modificados desde dentro de la propia clase.

**Métodos Públicos:** Los métodos de una clase que se utilizan para acceder o modificar los atributos privados de manera controlada.

**Ejemplos:**

**Encapsulamiento en Java:**

```
public class CuentaBancaria {
    private double saldo;

    public double getSaldo() {
        return saldo;
    }
    public void depositar(double cantidad) {
        if (cantidad > 0) {
            saldo += cantidad;
        }
    }
```

}

**Sobrecarga de Métodos en Programación Orientada a Objetos**

La "sobrecarga de métodos" permite que una clase tenga múltiples métodos con el mismo nombre, pero con diferentes listas de parámetros.

Los métodos sobrecargados comparten el mismo nombre, pero pueden realizar tareas diferentes según los parámetros utilizados.

**Métodos Sobrecargados**: Son métodos en una clase con el mismo nombre, pero diferentes listas de parámetros.

**Ejemplos:**

**Sobrecarga de Métodos en Java:**

```
public class Calculadora {
    public int sumar(int a, int b) {
        return a + b;
    }
    public double sumar(double a, double b) {
        return a + b;
    }
}
```

**Ejercicios Prácticos:**

Crea una clase "Vehiculo" con atributos como "marca" y "modelo". Luego, crea una subclase "Coche" que herede de "Vehiculo" y tenga atributos adicionales como "año" y "color".

Define una clase "Rectángulo" con atributos "ladoA" y "ladoB" y métodos para calcular el área. Luego, utiliza la sobrecarga de métodos para permitir el cálculo del área de un cuadrado (lados iguales) o un rectángulo (lados diferentes).

**Preguntas de Repaso:**

¿Qué es la herencia en la Programación Orientada a Objetos y cuál es su propósito?

¿Qué significa el encapsulamiento y por qué es importante en la POO?

¿Qué es la sobrecarga de métodos y cómo se implementa en una clase?

**Polimorfismo, Interfaces, clases abstractas en Programación Orientada a Objetos**

El "polimorfismo" es un concepto clave en la Programación Orientada a Objetos que se basa en la capacidad de objetos de diferentes clases para responder a un mismo mensaje de manera diferente.

El polimorfismo permite tratar objetos de clases diferentes de manera uniforme y es una parte esencial del principio de "abstracción".

**Definiciones:**

**Polimorfismo**: Es la capacidad de objetos de diferentes clases para responder de manera única a un mismo mensaje o método. Puede lograrse mediante herencia y la implementación de interfaces en POO.

**Ejemplos**:

Polimorfismo en Java:

```
public interface Animal {
    void hacerSonido();
}
public class Perro implements Animal {
    public void hacerSonido() {
        System.out.println("Un perro ladra.");
    }
}
```

```
public class Gato implements Animal {
    public void hacerSonido() {
        System.out.println("Un gato maulla.");
    }
}
```

**Uso de Polimorfismo en Java:**

```
Animal miMascota;
miMascota = new Perro();
miMascota.hacerSonido(); // Llama al método hacerSonido de la clase Perro
miMascota = new Gato();
miMascota.hacerSonido(); // Llama al método hacerSonido de la clase Gato
```

**Ejercicios Prácticos:**

Crea una interfaz "FiguraGeometrica" con un método "calcularArea". Luego, implementa dos clases, "Círculo" y "Rectángulo", que implementen esta interfaz y proporcionen su propia implementación del método "calcularArea".

Define una clase "Vehiculo" con un método "acelerar". Luego, crea dos subclases, "Coche" y "Motocicleta", que hereden de "Vehiculo" y proporcionen su propia implementación del método "acelerar".

**Preguntas de Repaso:**

¿Qué es el polimorfismo en la Programación Orientada a Objetos y por qué es importante?

¿Cómo se logra el polimorfismo en POO?

¿Cuál es la relación entre el polimorfismo y las interfaces en Java?

**Interfaces y Clases Abstractas en Programación Orientada a Objetos**

Las "interfaces" y las "clases abstractas" son dos conceptos que permiten definir contratos y proporcionar estructura a las clases en la POO.

Las interfaces y las clases abstractas definen métodos que deben ser implementados por las clases concretas, lo que fomenta la estandarización del comportamiento.

**Definiciones**:

**Interface**: Una interfaz es un conjunto de métodos abstractos (sin implementación) que se definen en una interfaz y deben ser implementados por las clases que la utilicen. Una clase puede implementar múltiples interfaces.

**Clase Abstracta:** Una clase abstracta es una clase que puede contener métodos abstractos, pero también puede contener métodos concretos con implementación. Las clases concretas que heredan de una clase abstracta deben implementar los métodos abstractos.

**Ejemplos:**

**Interfaz en Java:**

```
public interface Animal {
    void emitirSonido();
    void comer();
}
```

**Clase Abstracta en Java:**

```
public abstract class Figura {
    private int x;
    private int y;
   public abstract double calcularArea();
```

```
    public void mover(int newX, int newY) {
        x = newX;
        y = newY;
    }
}
```

**Ejercicios Prácticos:**

Crea una interfaz "FiguraGeometrica" con métodos abstractos para calcular el área y el perímetro. Luego, implementa dos clases, "Círculo" y "Rectángulo", que implementen esta interfaz y proporcionen su propia implementación de los métodos.

Define una clase abstracta "Vehiculo" con un método abstracto "acelerar". Luego, crea dos clases, "Coche" y "Motocicleta", que hereden de "Vehiculo" y proporcionen su propia implementación del método "acelerar".

**Preguntas de Repaso:**

¿Cuál es la diferencia entre una interfaz y una clase abstracta en la Programación Orientada a Objetos?

¿Por qué son importantes las interfaces y las clases abstractas en la POO?

¿Cómo se implementa una interfaz en una clase y cuáles son las reglas para hacerlo?

## ACTIVIDADES PRÁCTICAS

**Tabla**

Detalle de la Práctica

| Practicas | Instrucciones | Duración de la Practica | Mecanismo de Evaluación |
|---|---|---|---|
| **Foro de debate** | Revisar Actividades Prácticas | 0:30 mm | Por rúbricas |
| **Práctica investiga: Taller/Ensayo/Investigación** | Revisar Actividades Prácticas | 3:30 h | Por rúbricas |
| **Tutorías Prácticas** | Revisar Actividades Prácticas | 2:00 mm | Por participación en clase |
| **Test evaluativo** | Revisar Actividades Prácticas | 0:30 mm | Formativa |

*Nota.* Elaboración propia

**Consideraciones Generales**

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

**FORO DE DEBATE**

**Tema:** Diferencia entre Polimorfismo y Herencia.

**Instrucciones para Participar:**

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

**PRÁCTICA INVESTIGATIVA**

**Taller:** Crear un modelo MER.

**Consigna:**

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

**PRÁCTICA TEST**

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

**Consideraciones para realizar el test**

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

## II. DATOS GENERALES DE LA UNIDAD 2

### Información General

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.1. | Nivel | Segundo | | | | | |
| 1.2. | Nombre Unidad | Controles básicos y lenguaje de programación Java entorno Netbeans | | | | | |
| 1.3. | Tema 1 – Unidad | Entorno de IDE de POO | | | | | |
| 1.4. | Tema 2 – Unidad | Objetos de Presentación de Información | | | | | |
| 1.5. | Unidad organizacional | Unidad Profesional | | | | | |
| 1.6. | Unidad | 2 | | | | | |
| 1.7. | Total, Horas Unidad | 15 | | | | | |
| 1.8. | Detalle de horas Unidad | ACD | 6 | AA | 4 | APE | 5 |

### Resultado de Aprendizaje

Al finalizar esta unidad, los estudiantes serán capaces de comprender y aplicar las estructuras de control fundamentales en Java, incluyendo if, else, if-else if, y switch, para tomar decisiones condicionales. Se fomenta la participación en debates y preguntas y respuestas, y se proporcionan recursos en línea para una comprensión más profunda. La evaluación se realiza mediante pruebas, proyectos y una evaluación final, con retroalimentación continua para guiar el aprendizaje. Esta metodología busca que los estudiantes comprendan los conceptos fundamentales y puedan aplicar eficazmente las metodologías de desarrollo de software en situaciones prácticas.

### Metodología

La Metodología de se basa en un enfoque activo y colaborativo. Incluye sesiones de clase estructuradas en torno a temas específicos, lecturas y tareas, estudio de casos, proyectos prácticos y debates en clase. Los estudiantes también se involucran en trabajos en grupo  y mantienen diarios de reflexión. Se fomenta la participación en debates y preguntas y respuestas, y se proporcionan recursos en línea para una comprensión más profunda. La evaluación se realiza mediante pruebas, proyectos y una evaluación final, con retroalimentación continua para guiar el aprendizaje.

**DESARROLLO**

**Unidad 2: Controles básicos y lenguaje de programación Java entorno Netbeans**

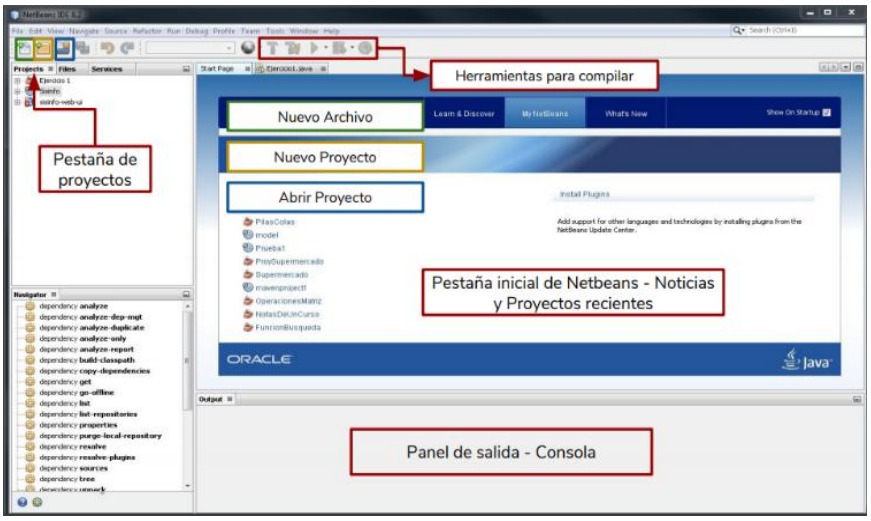
**Tema 1: Entorno de IDE de POO**

**Objetivo:** Explorar el entorno de desarrollo integrado (IDE) en Programación Orientada a Objetos (POO), específicamente en Java con NetBeans, comprendiendo su estructura, herramientas y funcionalidades, con el propósito de facilitar la implementación de aplicaciones eficientes y estructuradas.

**Conceptualización:**

El entorno de desarrollo NetBeans es una herramienta integral para el desarrollo de software en Java. Ofrece un entorno gráfico intuitivo que facilita la creación, edición, compilación y depuración de programas. El cuadro de herramientas de NetBeans proporciona acceso a componentes gráficos que simplifican la construcción de interfaces de usuario. Algunos de estos controles básicos incluyen etiquetas (JLabel), campos de texto (JTextField), áreas de texto (JTextArea), botones (JButton), entre otros. El control de ingreso y salida de textos se gestiona mediante estos componentes, permitiendo la interacción fluida con el usuario.

**Imagen 1:** Entorno de NetBeans

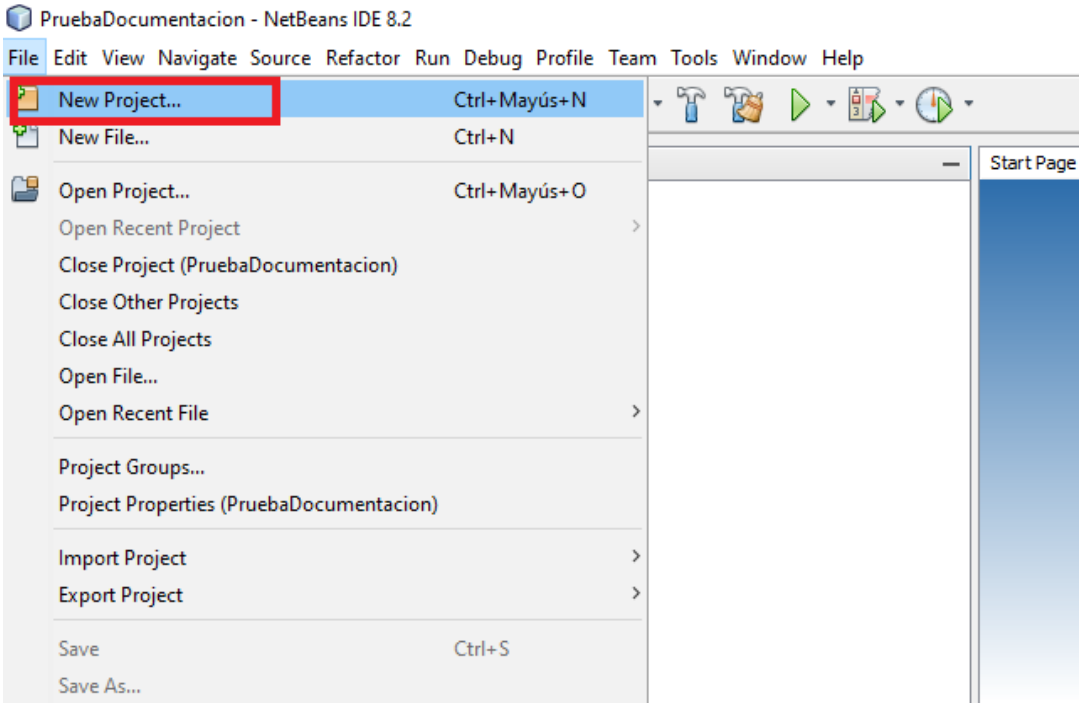



**Fuente**: https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html

A continuación, se detalla los pasos para crear un proyecto dentro del entorno IDE de NetBeans:

Paso 1: Ir a la opción nuevo proyecto o File >> New Project

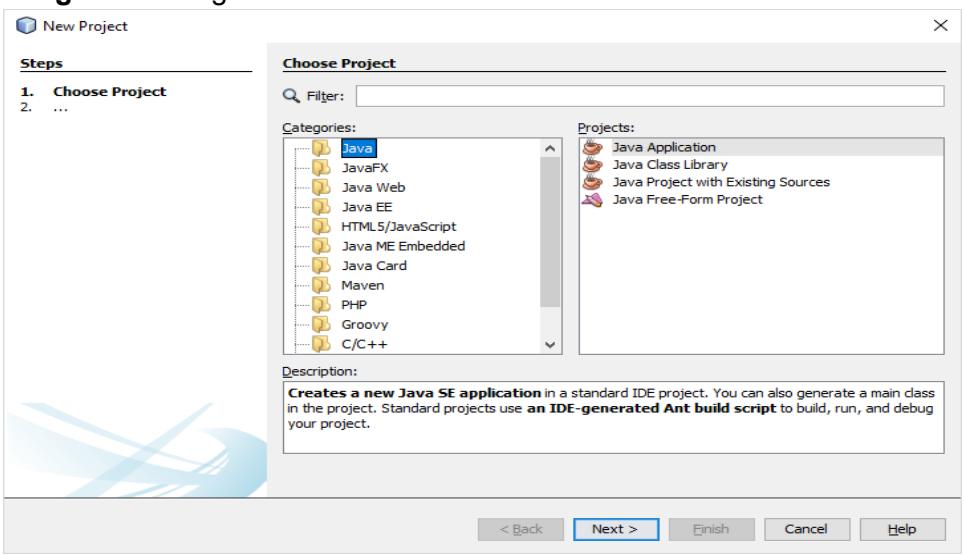
**Imagen 2:** Nuevo Proyecto de NetBeans




**Fuente**: https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html

Paso 2: Seleccionar la categoría Java y el tipo Java Application

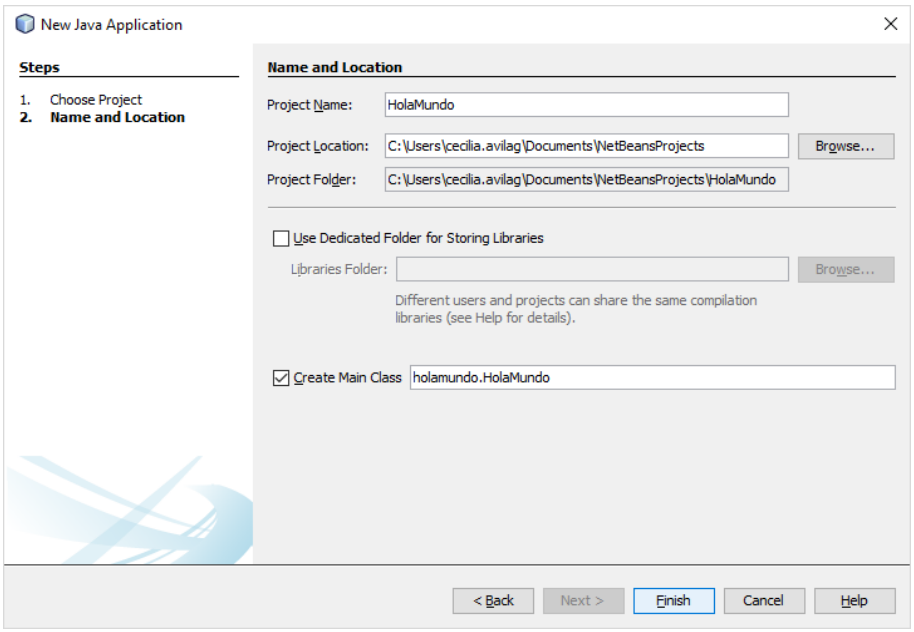
**Imagen 3:** Categoría de NetBeans




**Fuente**: https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html

Paso 3: Escribir el nombre del proyecto en el campo Project Name y verificar que esté seleccionada la opción "Create Main Class".

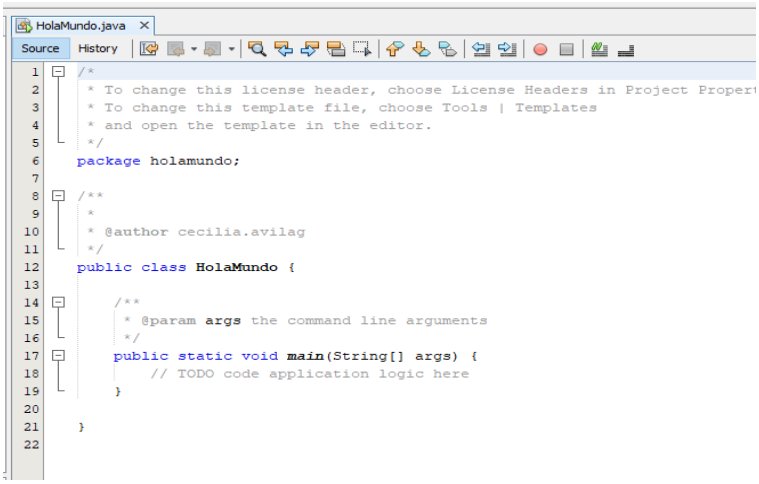
**Imagen 4:** Nombre de Proyecto de NetBeans




**Fuente**: https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html

Paso 4: Observar la clase que se genera. Por defecto aparece creada una clase que lleva el mismo nombre del proyecto, la cual tiene el método main:

**Imagen 5:** Proyecto de NetBeans



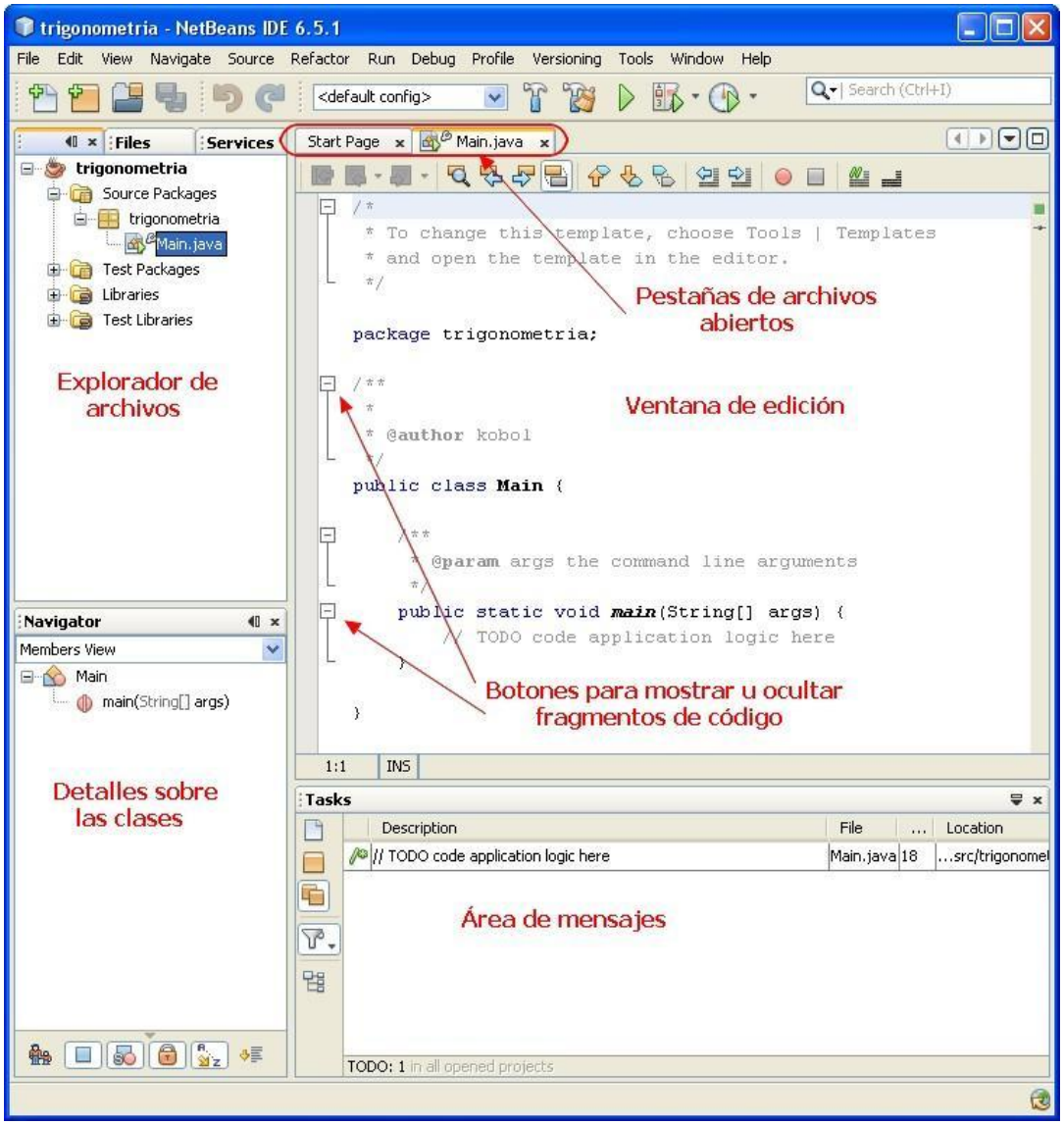

**Fuente**: [https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html](https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html)

Guía para descargar e Instalar NetBeans:
https://www.youtube.com/watch?v=n3tcNwtxPuE

**Entorno de NETBANS, cuadro de herramientas, Control ingresos y salidas de textos. Propiedades y Eventos**

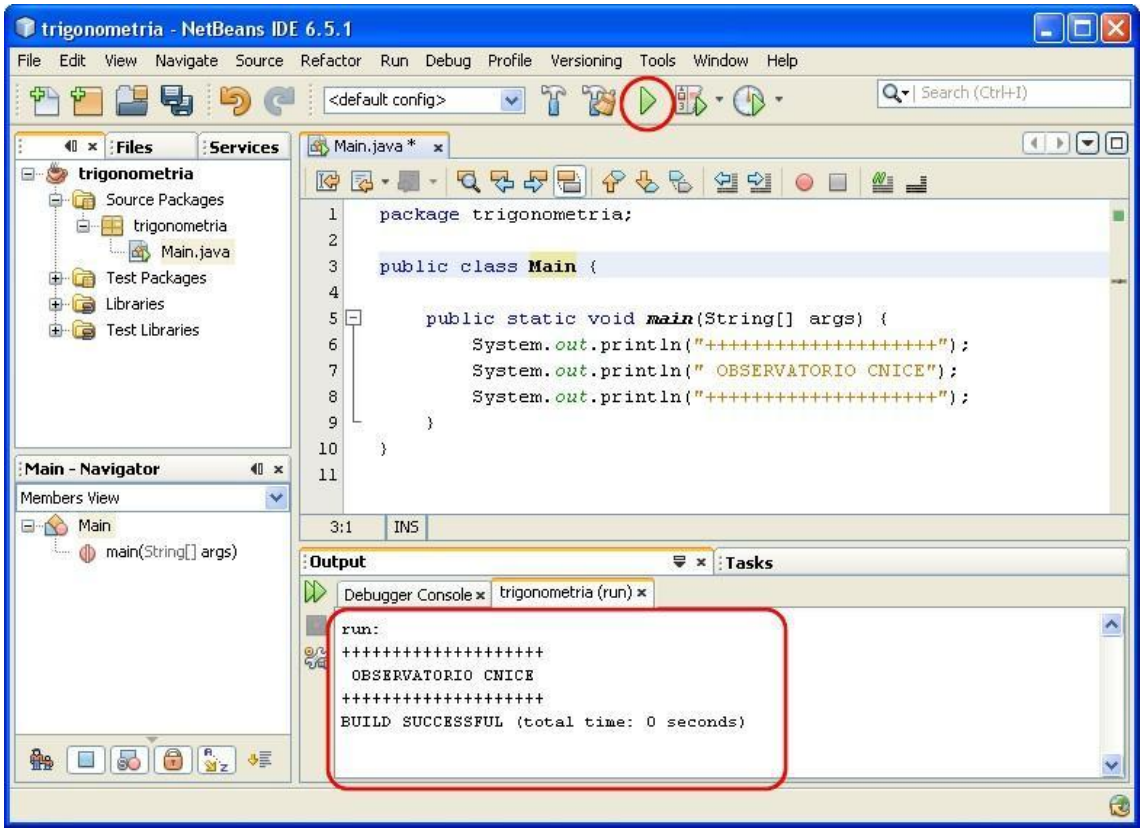
En una ventana posterior daremos nombre al proyecto y escogeremos una ubicación en el disco para almacenarlo. También nos dará la opción para crear automáticamente una clase principal, la que hasta ahora hemos llamado Test, y que aquí por defecto se llamará Main. Una vez hecho esto, la ventana de trabajo nos resultará sin duda compleja, pero también familiar:

NetBeans nos ofrece un esqueleto de clase Main, que podemos utilizar para hacer nuestra primera prueba. Por defecto, NetBeans compila automáticamente los archivos



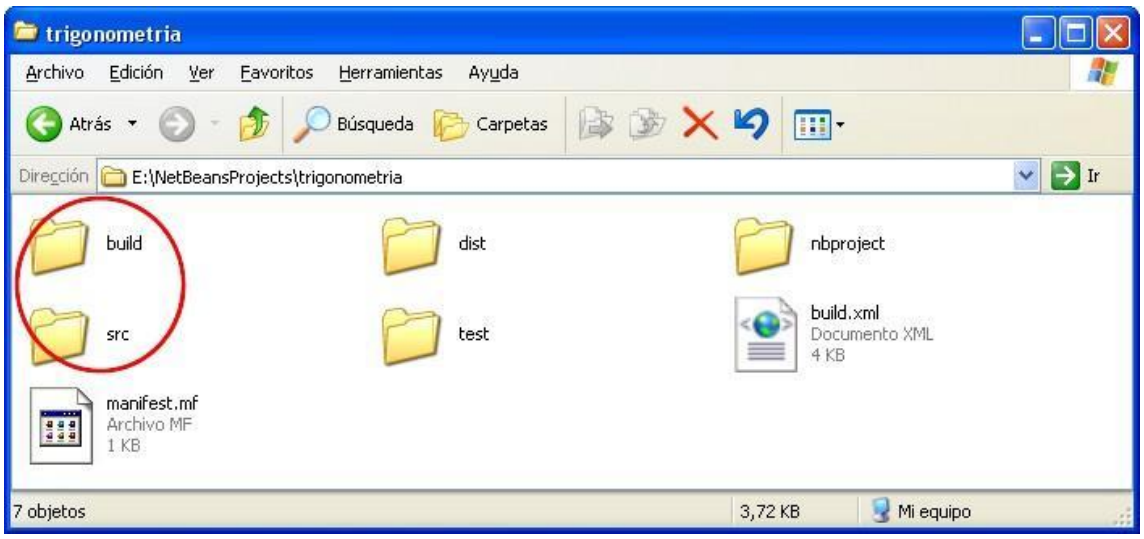

en el momento en que los grabas (icono del diskette), de forma que únicamente

necesitamos utilizar el botón que representa una flecha verde, como el *play* de un mando a distancia. Verás el resultado de la ejecución en la ventana inferior:

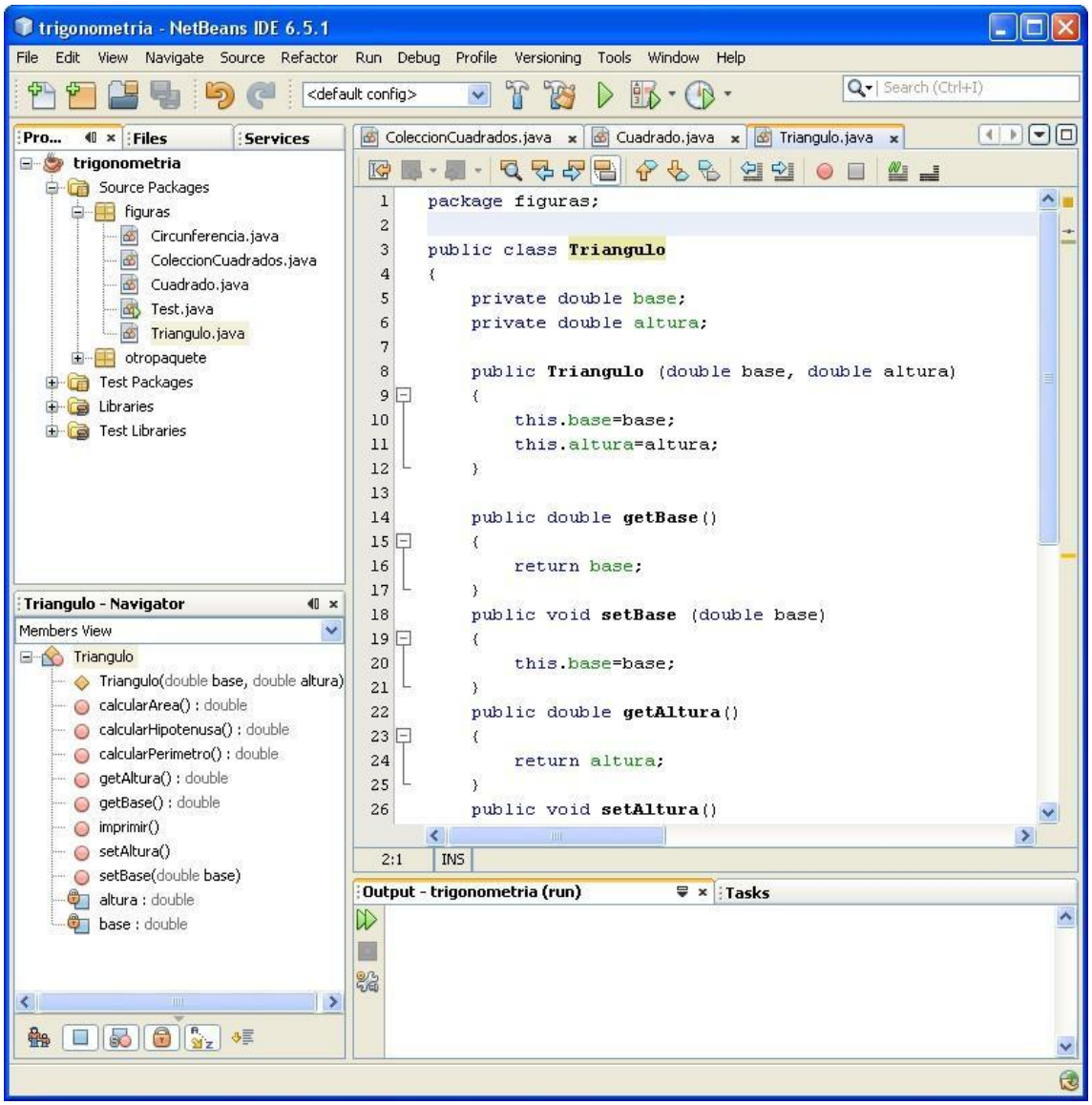



Como ves, no ha sido nada difícil empezar a trabajar con NetBeans; en la siguiente sección esbozaremos las capacidades de esta potente aplicación.

Examinemos en detalle la carpeta de proyecto. Verás que se han creado múltiples subcarpetas en tu directorio de trabajo:



De ellas sólo nos van a interesar dos: en *src* se encuentran los archivos fuente de nuestro programa, mientras que en *build* es donde guarda NetBeans los archivos compilados (*.class*). Para añadir una nueva clase al proyecto sólo tienes que utilizar el botón *New Class*; si lo que quieres es agregar al proyecto clases ya existentes, la forma más rápida consiste en copiarlas directamente a la carpeta *src*.

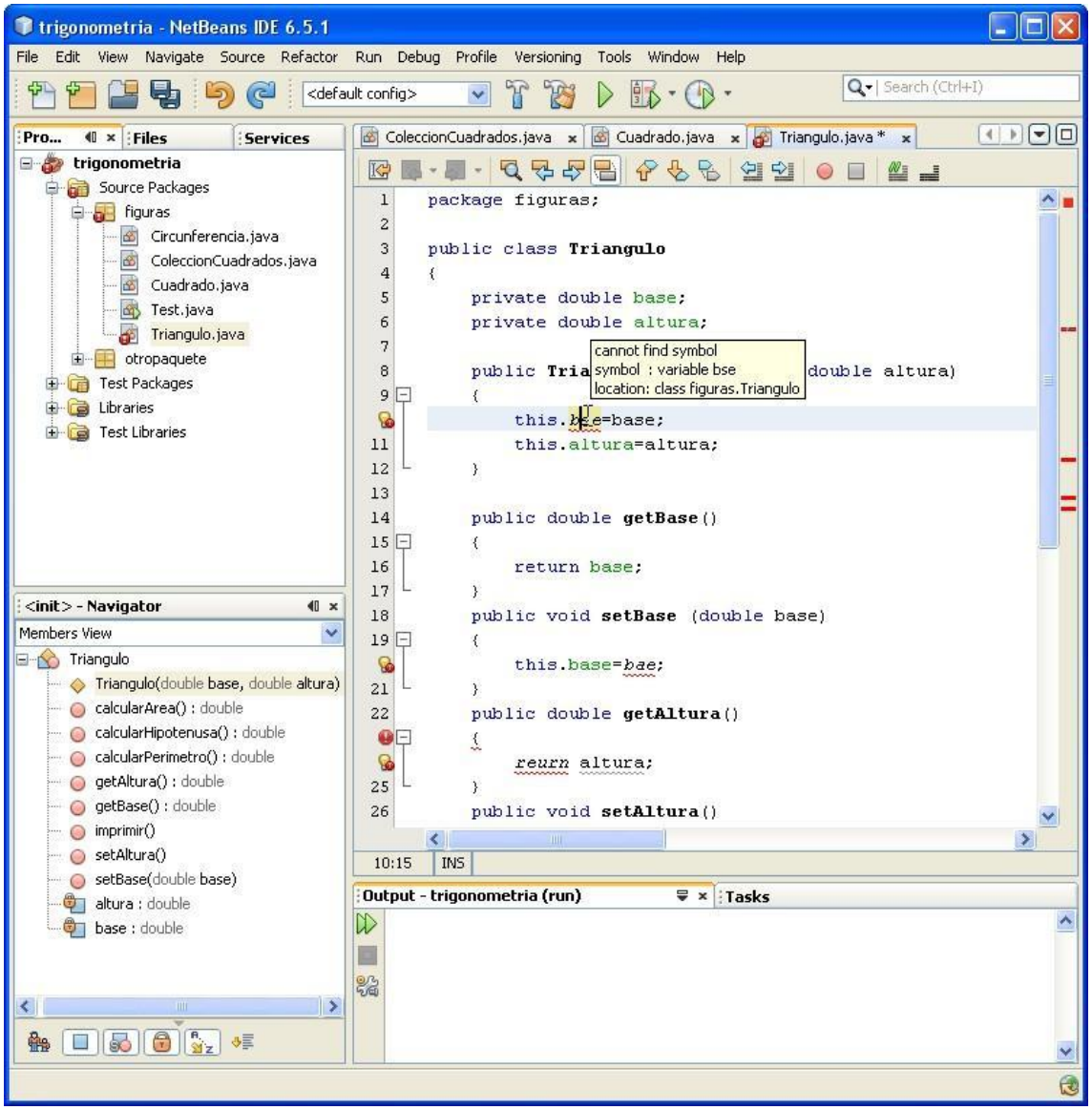



Si examinamos en detalle la ventana de trabajo, apreciaremos lo siguiente:

- A diferencia de los entornos anteriormente estudiados, en NetBeans puedes ver las clases en la parte izquierda agrupadas por el paquete al que pertenecen
- La clase que contiene el método principal main está destacada con una flecha verde en su icono

- La parte inferior izquierda es como un diagrama de clase: incluye toda la información sobre atributos y métodos de la clase que estemos editando
- NetBeans también puede mostrar números de línea, mediante el menú View > Show line numbers.

Como se ha dicho antes, en NetBeans no existe la necesidad explícita de compilar, puesto que cuenta con una función de autocompilado. Piensa en un editor de textos avanzado: según vas escribiendo, el programa comprueba tu ortografía y, si has cometido una falta, la destaca en rojo sobre la marcha. NetBeans funciona exactamente igual; es más, la forma en que resalta los errores sigue la misma estética que los correctores ortográficos. Si queremos saber el motivo del problema, bastará con situar el cursor sobre una de las marcas de error:



En jGrasp o BlueJ, el primer error de compilación provocará la detención del proceso: aquí puedes seguir trabajando aunque se acumulen varios errores, corrigiéndolos después en el orden que quieras. Fíjate en la parte izquierda de la ventana: NetBeans distinguirá con iconos aquellas clases que presentan errores de compilación. Y puesto que la clase pertenece al paquete *figuras*, también marca el paquete completo como erróneo. De igual forma, el proyecto *trigonometría* queda marcado en rojo por la existencia de al menos un error en sus archivos.

Sin duda, y pese a la complejidad, resulta muy cómodo programar en NetBeans. Se trata de un completo entorno que nos permitirá realizar no sólo programas sencillos sino auténticos proyectos profesionales de programación. Como todos los IDEs dirigidos a programadores profesionales, ofrece herramientas avanzadas que nosotros, en un curso de iniciación a Java, raramente vamos a utilizar. Pero para que te hagas una idea de lo que estamos hablando, repasaremos algunas de sus posibilidades:

- Permite la integración de la herramienta Ant de Apache, que permite automatizar mediante guiones o scripts la compilación de proyectos complejos y realizarla después dándole a un solo botón. Las posibilidades de Ant son enormes, permitiendo especificar, por ejemplo, que los archivos fuente se almacenan en unas determinadas carpetas mientras los compilados se deben copiar a otras. También se pueden programar scripts de limpieza que eliminen todos los archivos compilados o no utilizados.

- Soporta control de versiones con la herramienta gratuita CVS. Esta funcionalidad es extremadamente útil al programar en equipo, y básicamente consiste en tener un único almacén de archivos compartido por todos los colaboradores de un proyecto de programación. El sistema registra todos los cambios realizados de forma que cada programador puede ver resaltado en colores qué cambios ha hecho su compañero en cada archivo. Cuando el equipo se da cuenta de que ha cometido un error, puede dar marcha atrás y recuperar rápidamente una versión anterior, deshaciendo los cambios realizados. Más aún: dos programadores pueden modificar un mismo archivo a la vez. Cuando el segundo intente devolver el archivo modificado al almacén, el programa le facilitará la revisión para *mezclar* sus cambios con los de su compañero, siempre que no resulten incompatibles.

- Ofrece la posibilidad de refactorizar código. Utilizamos esta palabra para designar cambios menores en el código que, sin modificar la funcionalidad de nuestro programa, lo simplifican o facilitan su legibilidad. El ejemplo más básico de refactorización es cambiar el nombre de una variable: el entorno se ocupará de realizar el cambio en todos los puntos del programa en que aparezca, incluso en otros archivos del proyecto. NetBeans también permite ordenar automáticamente el código, introduciendo tabulaciones y espaciados para que el programa se lea con facilidad.

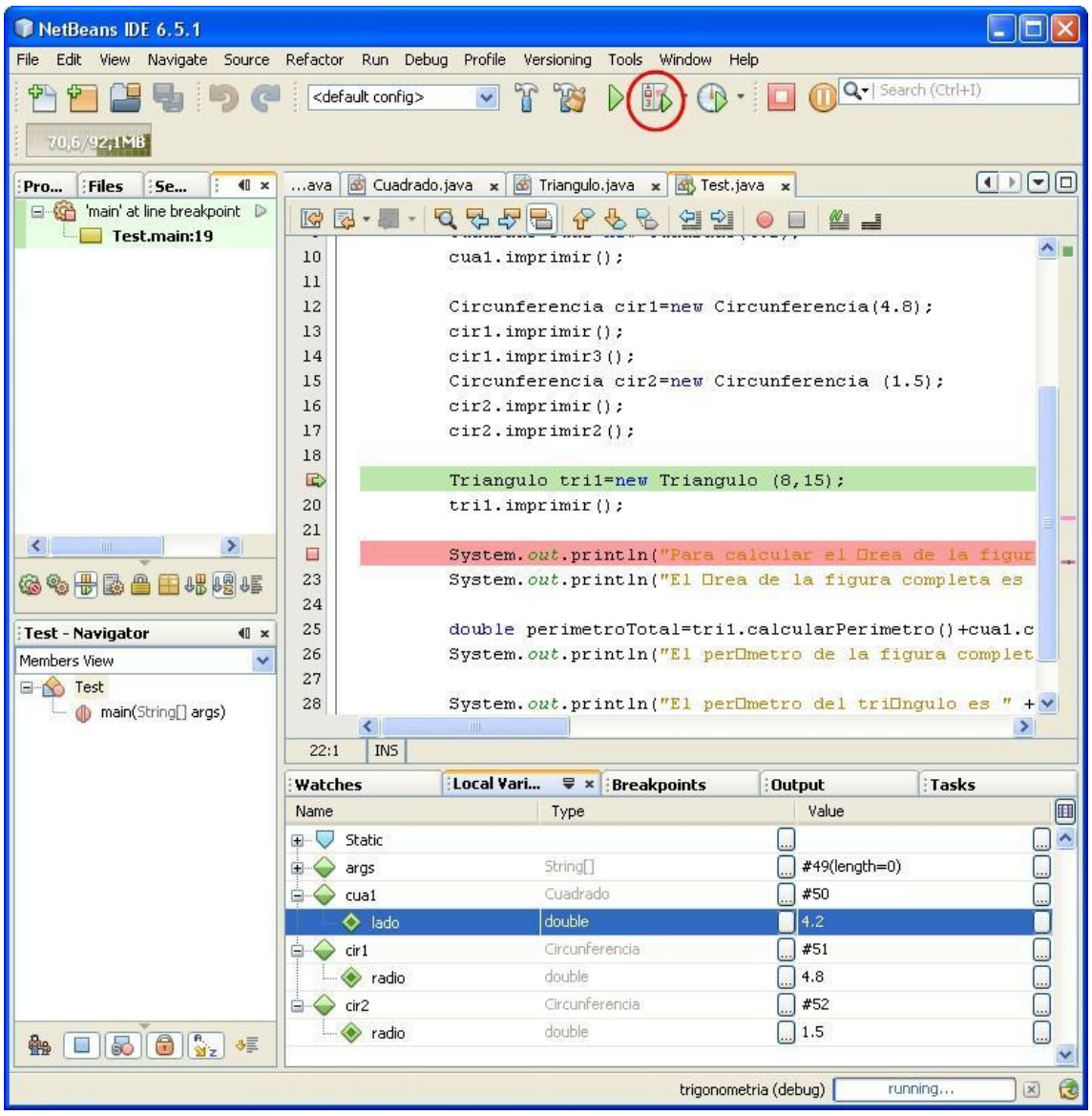
- Incorpora el autocompletado de código. Imagina que estás escribiendo el nombre de una variable o de un objeto, pero no recuerdas qué nombre le diste exactamente: pues bien, simplemente escribe las primeras letras, pulsa la tecla Control y la barra espaciadora, y el programa te indicará cuáles son los nombres disponibles. Algo parecido sucede con los atributos y métodos de una clase. Si escribimos el nombre de un objeto seguido de un punto, sabemos que a continuación debe ir o bien un atributo o bien un método de los que ofrezca la clase a la que pertenece el objeto en cuestión. Si no recordamos su nombre repetiremos la combinación de teclado y el entorno nos ofrecerá las distintas posibilidades:




Las dos primeras funcionalidades están muy orientadas al uso profesional y tendría poco sentido utilizarlas en un curso de iniciación, pero es posible que algún profesor quiera ilustrar las posibilidades de trabajo en equipo incorporando un servidor de control de versiones. El autocompletado, sin embargo, puede ser utilizado en clase con frecuencia; otra cuestión sería el considerarlo o no adecuado para la labor docente. La función de NetBeans es permitir programar deprisa, y funciones como el formateado automático del texto y el autocompletado de código ahorran al programador tiempo y errores. Algunos profesores considerarán que, cuando empezamos, no deberíamos recibir tantas ayudas y sino enfrentarnos precisamente a esos errores. Otros valorarán estas herramientas para que los alumnos se sientan más guiados en su tarea de programar.

NetBeans no permite la creación interactiva de objetos (una iniciativa interesante a nivel didáctico, pero no tanto en un IDE profesional). Pero podemos completar nuestro

paseo por el entorno comprobando cómo funciona la depuración. La forma de realizarla es igual que en los otros entornos, aunque NetBeans proporciona posibilidades de inspección y monitorización mucho más avanzadas. Lo que a nosotros nos interesa, sin embargo, se realiza de forma similar: haz click en la columna gris a la izquierda de las líneas donde quieras añadir puntos de ruptura, y pulsa el icono de depurar programa:




En verde se destaca el punto de ruptura en el que estamos detenidos, y en rojo otros puntos de ruptura que hayamos marcado. En la ventana inferior podremos examinar los valores de todos los objetos en este punto, y a través del menú Debug encontraremos las opciones ya conocidas para controlar la ejecución (*step, step into*...) aparte de otras posibilidades que sobrepasan nuestros intereses.

Los menús y mensajes de NetBeans están escritos, al igual que su documentación, en inglés. Sin embargo, existe un grupo de trabajo dedicado a traducir el IDE al español, de forma que algunas versiones se encuentran traducidas a nuestro idioma. En cuanto a la documentación, iniciativas personales o universitarias de España o Iberoamérica permiten encontrar en la red ciertos recursos para comprender mejor esta popular y potente plataforma.

**Ejemplo**

Supongamos que deseamos crear una interfaz simple que solicita al usuario ingresar su nombre y luego muestra un saludo. Utilizaremos JTextField y JLabel para la entrada y salida de texto, respectivamente. El código en NetBeans podría verse así:

```
import javax.swing.*;


public class MiVentana extends JFrame {
    private JLabel etiqueta;
    private JTextField campoTexto;


    public MiVentana() {
        // Configuración de la ventana
        setTitle("Saludo");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());


        // Componentes
        etiqueta = new JLabel("Ingrese su nombre:");
        campoTexto = new JTextField(15);


        // Agregar componentes a la ventana
        add(etiqueta);
```

```
        add(campoTexto);


        // Mostrar la ventana
        setVisible(true);
    }


    public static void main(String[] args) {
        new MiVentana();
    }
}
```

**Control JButton y JOption Button. Propiedades y Eventos**

El control JButton es un botón interactivo que puede desencadenar eventos en respuesta a las acciones del usuario. Por otro lado, JOptionPane es una clase que proporciona cuadros de diálogo (ventanas emergentes) para interactuar con el usuario. Estos controles son esenciales para crear interfaces dinámicas y responder a eventos específicos, como clics de botones.

**Ejemplo**:

Vamos a extender el ejemplo anterior. Ahora, al presionar un botón, mostraremos un cuadro de diálogo emergente con un saludo personalizado. Utilizaremos JButton y JOptionPane para lograr esto.

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;


public class MiVentana extends JFrame {

    private JLabel etiqueta;

    private JTextField campoTexto;

    private JButton boton;


    public MiVentana() {

        // Configuración de la ventana

        setTitle("Saludo");

        setSize(300, 200);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
setLayout(new FlowLayout());

// Componentes
etiqueta = new JLabel("Ingrese su nombre:");
campoTexto = new JTextField(15);
boton = new JButton("Saludar");

// Agregar componentes a la ventana
add(etiqueta);
add(campoTexto);
add(boton);

// Manejador de eventos para el botón
boton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String nombre = campoTexto.getText();
        JOptionPane.showMessageDialog(null, "¡Hola, " + nombre + "!");
    }
});

// Mostrar la ventana
setVisible(true);
```

```
    }

    public static void main(String[] args) {
        new MiVentana();
    }
}
```

En este ejemplo, hemos agregado un JButton que, al ser presionado, desencadena un evento que recupera el texto ingresado y muestra un saludo personalizado a través de un cuadro de diálogo JOptionPane. Este tipo de interactividad es esencial al diseñar aplicaciones que requieren respuestas directas del usuario.

**Propiedades en NetBeans:**

En el entorno de desarrollo NetBeans, las propiedades son atributos configurables de un componente gráfico que afectan su apariencia, comportamiento u otras características. Estas propiedades se pueden ajustar fácilmente a través de la interfaz gráfica, lo que facilita la personalización de la apariencia y el comportamiento de los elementos visuales en una aplicación.

**Ejemplo**:

Supongamos que tenemos un botón (JButton) en nuestra interfaz y queremos cambiar su texto. En NetBeans, podemos hacer esto seleccionando el botón, y en la ventana de propiedades, buscar la propiedad llamada "text" y cambiar su valor a, por ejemplo, "Hacer clic". Esta acción ajustará dinámicamente el texto visible en el botón.

**Eventos en NetBeans:**

Los eventos son acciones o situaciones específicas que pueden ocurrir durante la ejecución de un programa. En NetBeans, gestionar eventos es fundamental para responder a las interacciones del usuario. Puedes asociar métodos específicos, llamados "manejadores de eventos", a componentes gráficos para que se ejecuten cuando ocurra un evento particular, como hacer clic en un botón.

**Ejemplo**:

Si queremos manejar el evento de hacer clic en un botón (JButton), podemos hacerlo en NetBeans seleccionando el botón, y en la ventana de propiedades, buscamos la sección de eventos. Allí, encontraremos el evento "ActionPerformed" asociado al clic del botón. Podemos agregar un método para manejar este evento haciendo doble clic en el cuadro blanco junto a "ActionPerformed" en la ventana de eventos.

Después de hacer doble clic, NetBeans creará automáticamente un método en nuestro código Java para manejar el evento. Podemos escribir el código dentro de este método para definir la acción que queremos que ocurra cuando el botón sea clicado. Por ejemplo, cambiar el texto de una etiqueta o mostrar un cuadro de diálogo.

En resumen, NetBeans simplifica la gestión de propiedades y eventos al proporcionar interfaces visuales intuitivas que permiten ajustar configuraciones y asociar acciones a elementos gráficos sin la necesidad de escribir manualmente todo el código. Esto facilita el desarrollo y mantenimiento de aplicaciones Java.

**Atajos del teclado.**

Los ***shortcuts***, ***hot keys*** o atajos del teclado, son una combinación de teclas que todo software utiliza para ejecutar una acción determinada, en lugar de obligar al usuario a ir por el camino largo: click en el menú adecuado y posteriormente en el comando requerido. Y Netbeans no es la excepción. Consulta aquí (shortcuts-80) el listado de atajos de teclado que utiliza este IDE. Además de ayudarnos con los comandos del entorno, Netbeans agrega combinaciones de teclas para ayudarnos a escribir código. Por ejemplo, si quieres escribir la línea para imprimir un mensaje de texto en la consola de salida, puedes escribir *sout* y la tecla tabulador : 

para que Netbeans la sustituya por System.out.println(“”);

**Autocompletar**.

Creo que lo más genial que tiene Netbeans es la capacidad de completar de forma automática cuando estás usando clases de la API de Java. Sólo ten paciencia y cuidado de no equivocarte con la palabra inicial para que Netbeans reconozca el paquete y la clase que necesitas. Te muestro un ejemplo en la figura 6.

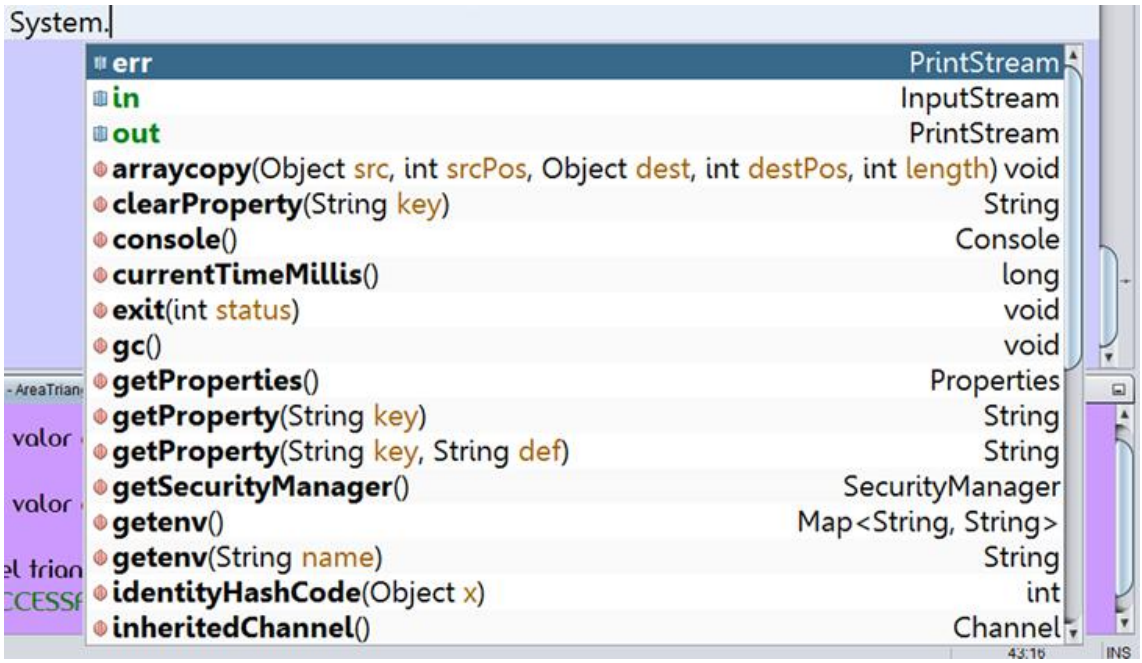



*Figura 6. Ejemplo de la característica autocompletar del editor de Netbeans*

**Sistema de colores y estilos de fuentes**.

Netbeans te guía para que sepas si estás realizando las cosas bien con todo un sistema de colores y estilos de fuentes . Presta atención, y conforme vayas editando tus programas te darás cuenta cuando algo no está bien gracias al color o estilo de fuente en que Netbeans va colocando lo que escribes. Por ejemplo, los tipos de datos nativos de Java los coloca en color azul (por lo menos en la configuración estándar). Las cadenas de texto las coloca en naranja. Y así existen otras guías usando colores.

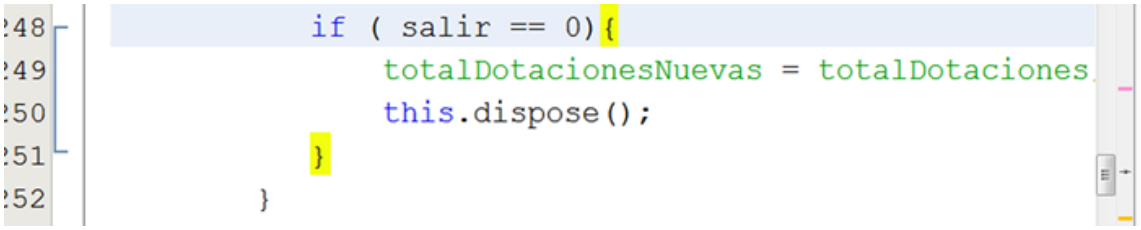



*Figura 7. Ejemplo de detección automática de llaves, corchetes y paréntesis*

**Detección de errores**.

Obviamente la ayuda más importante de Netbeans es indicarte los errores mientras estás escribiendo. Incluso antes de compilar, este IDE te va indicando los errores, y lo hace de diversas formas. En la figura 8 te muestro las múltiples formas en que Netbeans te indica que tienes errores de sintaxis.

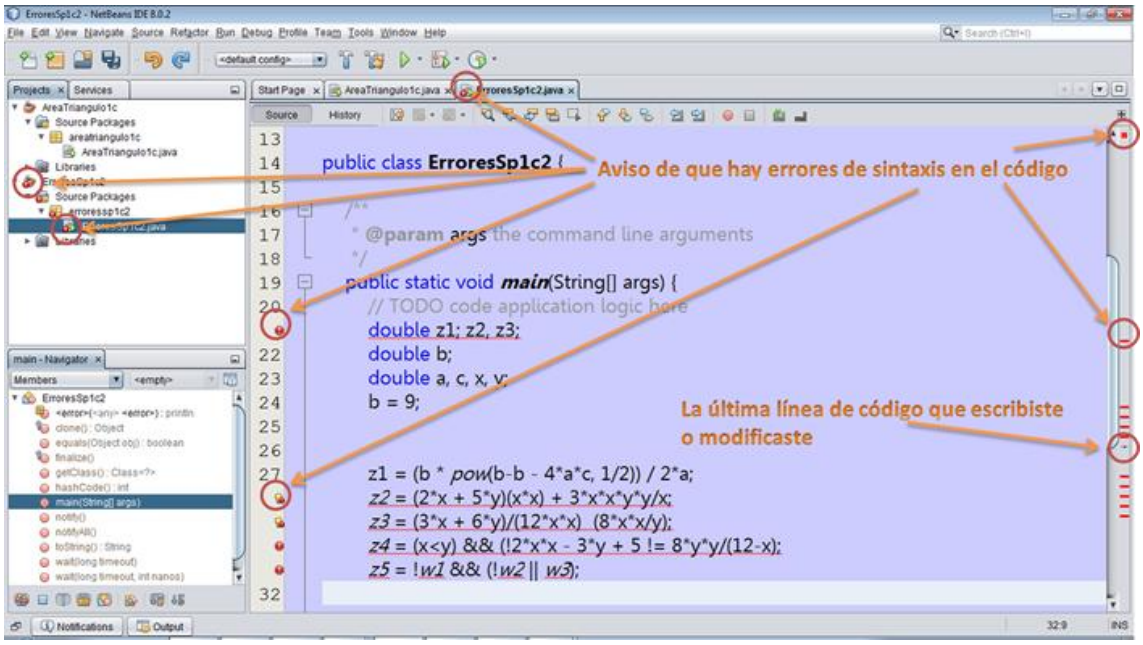



*Figura 8. Formas en la que Netbeans detecta los errores de sintaxis.*

**Mensajes de error.**

Aunque Netbeans te indica rápidamente que existe un error, corregirlo quizá no resulte tan fácil, sobre todo porque muchos de estos errores se marcan en la línea en la que el error no se genera. Recuerda que el error probablemente se encuentre una línea arriba. Otra cosa que no ayuda, es que la explicación del error se te muestra en inglés, y como ya mencioné en otros posts sino eres amig@ de este idioma se te puede complicar el proceso de depuración. En la figura 9 te muestro algunos errores comunes y como puedes solucionarlos. Esta tabla aún no está completa, la hice hace tiempo, para la versión 7, por lo que más adelante la estaré actualizando.

| Mensaje de error | Significado | Solución |
|---|---|---|
| cannot find symbol | No puede encontrar el símbolo | Una variable no está declarada, o está mal escrita. También puede indicar que un método de una clase se encuentra mal escrito. |
| incompatible types | Tipos incompatibles | Para que se pueda realizar esa operación, la variable debe ser declarada con otro tipo de dato. |
| ; expected | Se requiere ; | Falta el ; con el que deben terminar todas las instrucciones Java<br>También puede marcar este error cuando alguna expresión aritmética no se ha escrito correctamente |
| unclosed string literal | Literal de cadena no está cerrada apropiadamente | Verificar que las " encierren adecuadamente una cadena de texto. |
| package ..... does not exist | Paquete ---- no existe | Netbeans buscó un paquete que no existe dentro de los paquetes estándar de Java. Lo más probable es que una clase se encuentre mal escrita |

*Figura 9. Algunos errores comunes*

## ACTIVIDADES PRÁCTICAS

**Tabla**

Detalle de la Práctica

| Practicas | Instrucciones | Duración de la Practica | Mecanismo de Evaluación |
|---|---|---|---|
| **Foro de debate** | Revisar Actividades Prácticas | 0:30 mm | Por rúbricas |
| **Práctica investiga: Taller/Ensayo/Investigación** | Revisar Actividades Prácticas | 3:30 h | Por rúbricas |
| **Tutorías Prácticas** | Revisar Actividades Prácticas | 2:00 mm | Por participación en clase |
| **Test evaluativo** | Revisar Actividades Prácticas | 0:30 mm | Formativa |

*Nota.* Elaboración propia

**Consideraciones Generales**

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

**FORO DE DEBATE**

**Tema:** Entorno de NETBEANS

**Instrucciones para Participar:**

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

**PRÁCTICA INVESTIGATIVA**

**Taller:** No aplica para este tema.

**Consigna:**

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

**PRÁCTICA TEST**

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

**Consideraciones para realizar el test**

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

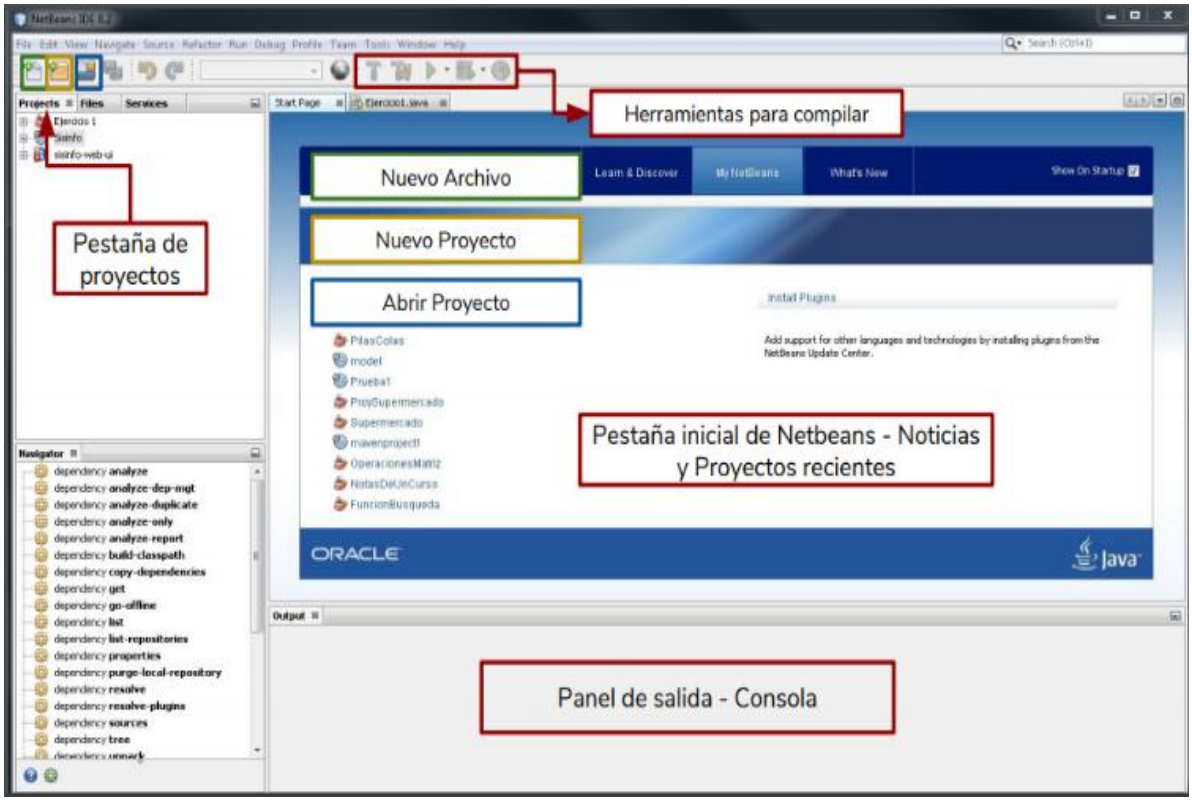
5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

**Tema 2: Objetos de Presentación de Información**

**Conceptualización:**

El entorno de desarrollo NetBeans es una herramienta integral para el desarrollo de software en Java. Ofrece un entorno gráfico intuitivo que facilita la creación, edición, compilación y depuración de programas. El cuadro de herramientas de NetBeans proporciona acceso a componentes gráficos que simplifican la construcción de interfaces de usuario. Algunos de estos controles básicos incluyen etiquetas (JLabel), campos de texto (JTextField), áreas de texto (JTextArea), botones (JButton), entre otros. El control de ingreso y salida de textos se gestiona mediante estos componentes, permitiendo la interacción fluida con el usuario.

**Imagen 1:** Entorno de NetBeans

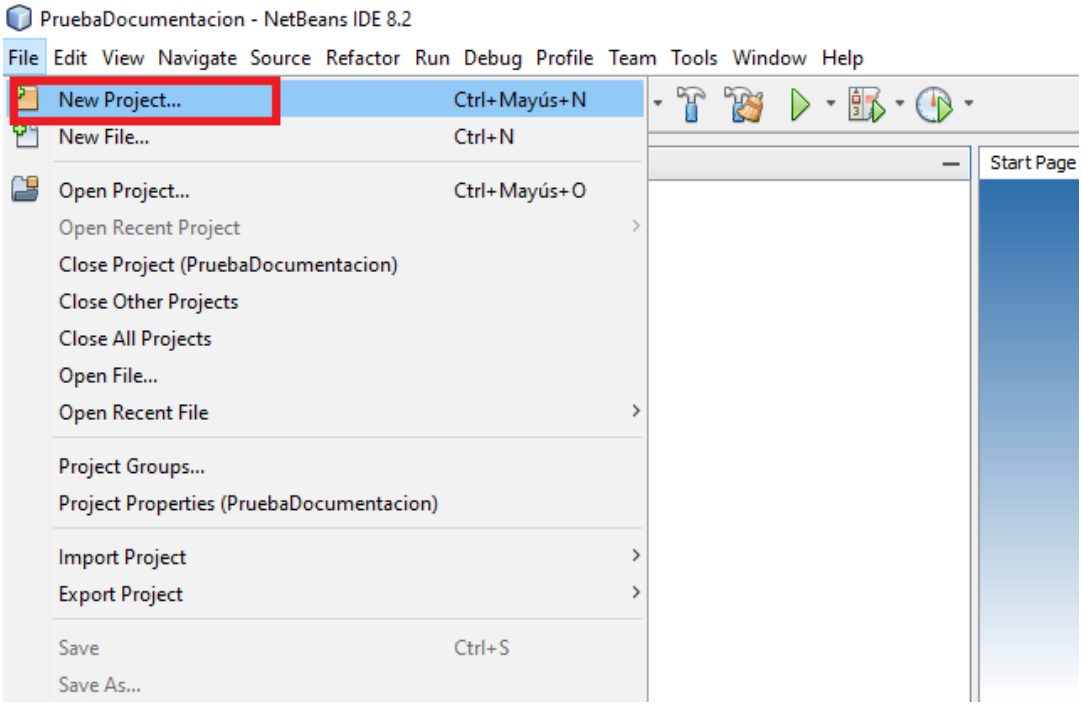



**Fuente**: https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html

A continuación, se detalla los pasos para crear un proyecto dentro del entorno IDE de NetBeans:

Paso 1: Ir a la opción nuevo proyecto o File >> New Project

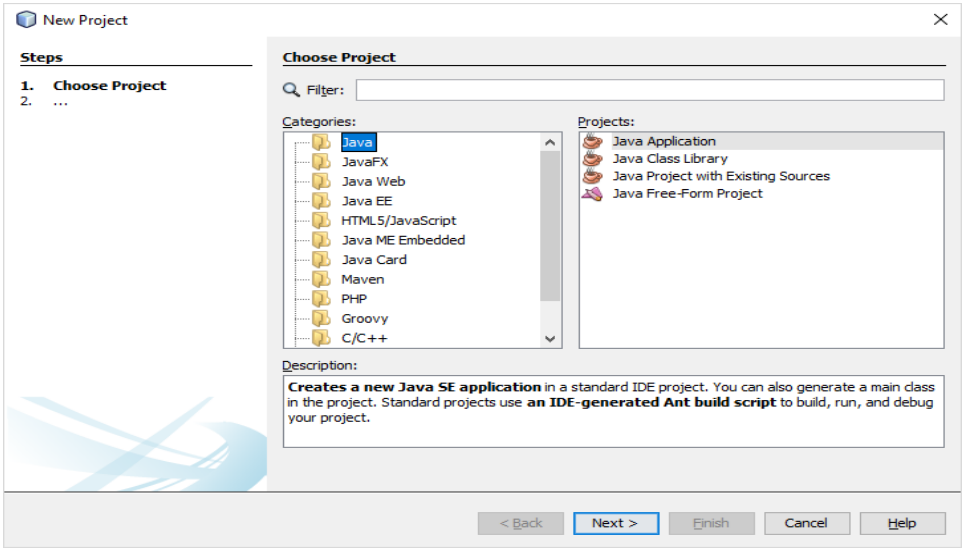
**Imagen 2:** Nuevo Proyecto de NetBeans




**Fuente**: https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html

Paso 2: Seleccionar la categoría Java y el tipo Java Application

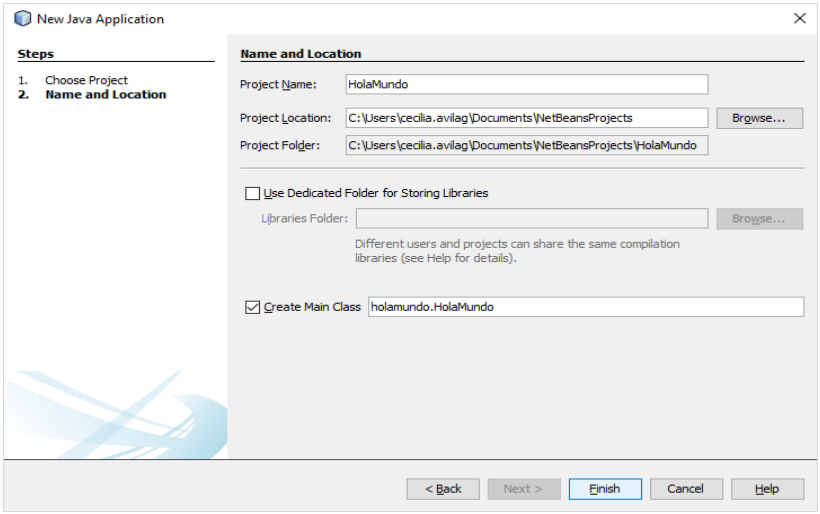
**Imagen 3:** Categoría de NetBeans



**Fuente**: https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html

Paso 3: Escribir el nombre del proyecto en el campo Project Name y verificar que esté seleccionada la opción "Create Main Class".

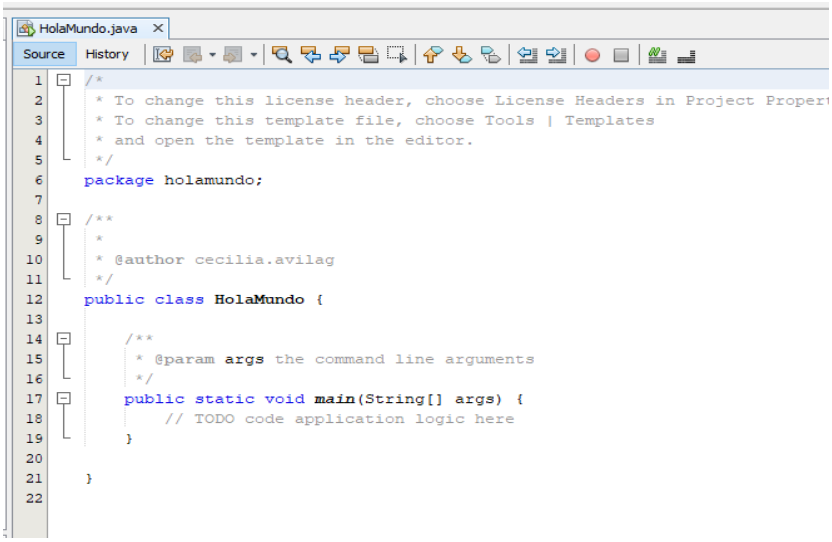
**Imagen 4:** Nombre de Proyecto de NetBeans




**Fuente**: https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html

Paso 4: Observar la clase que se genera. Por defecto aparece creada una clase que lleva el mismo nombre del proyecto, la cual tiene el método main:

**Imagen 5:** Proyecto de NetBeans



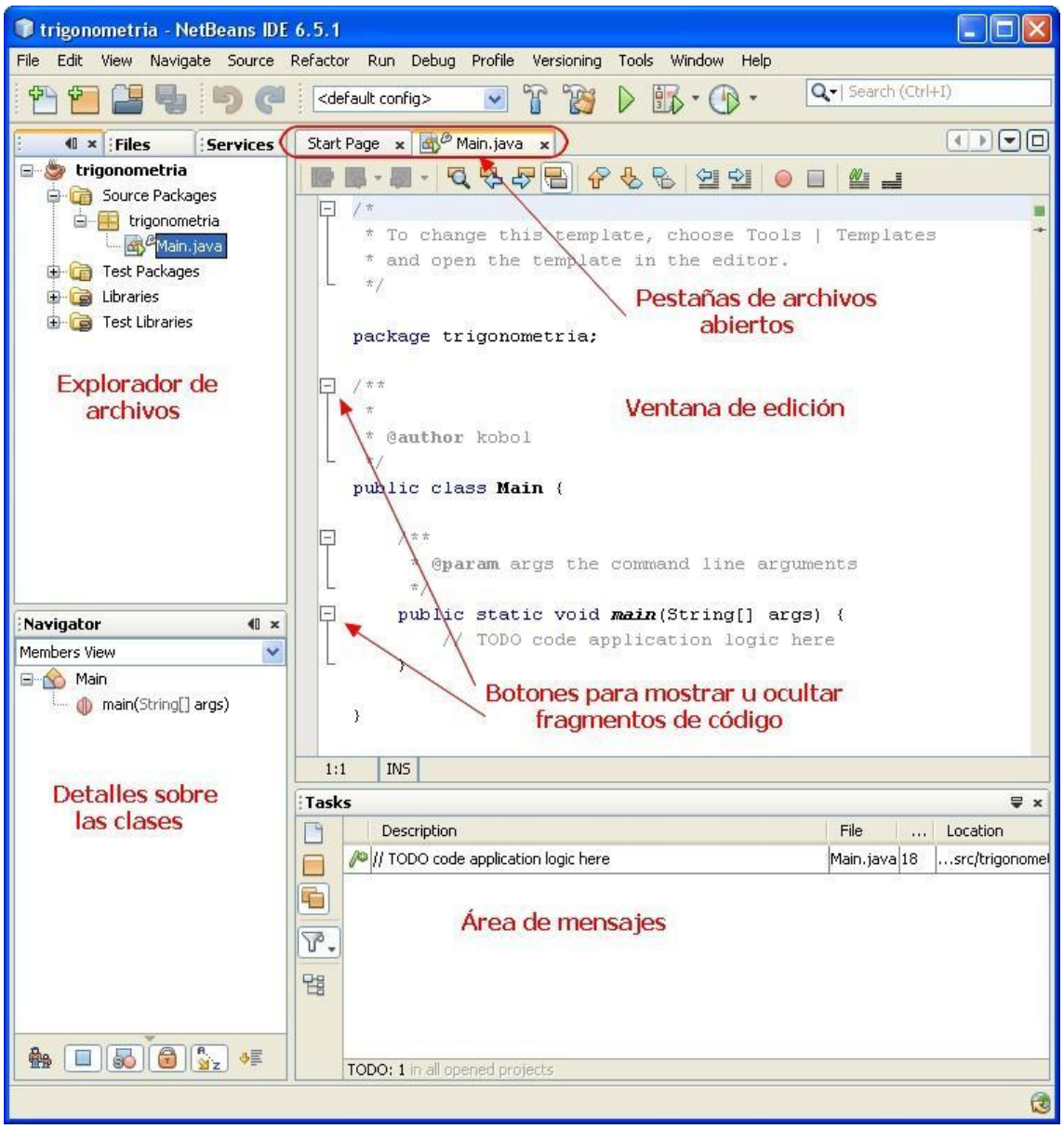

**Fuente**: https://repositorio.konradlorenz.edu.co/micrositios/001-985/entorno_netbeans.html

Guía para descargar e Instalar NetBeans:

https://www.youtube.com/watch?v=n3tcNwtxPuE

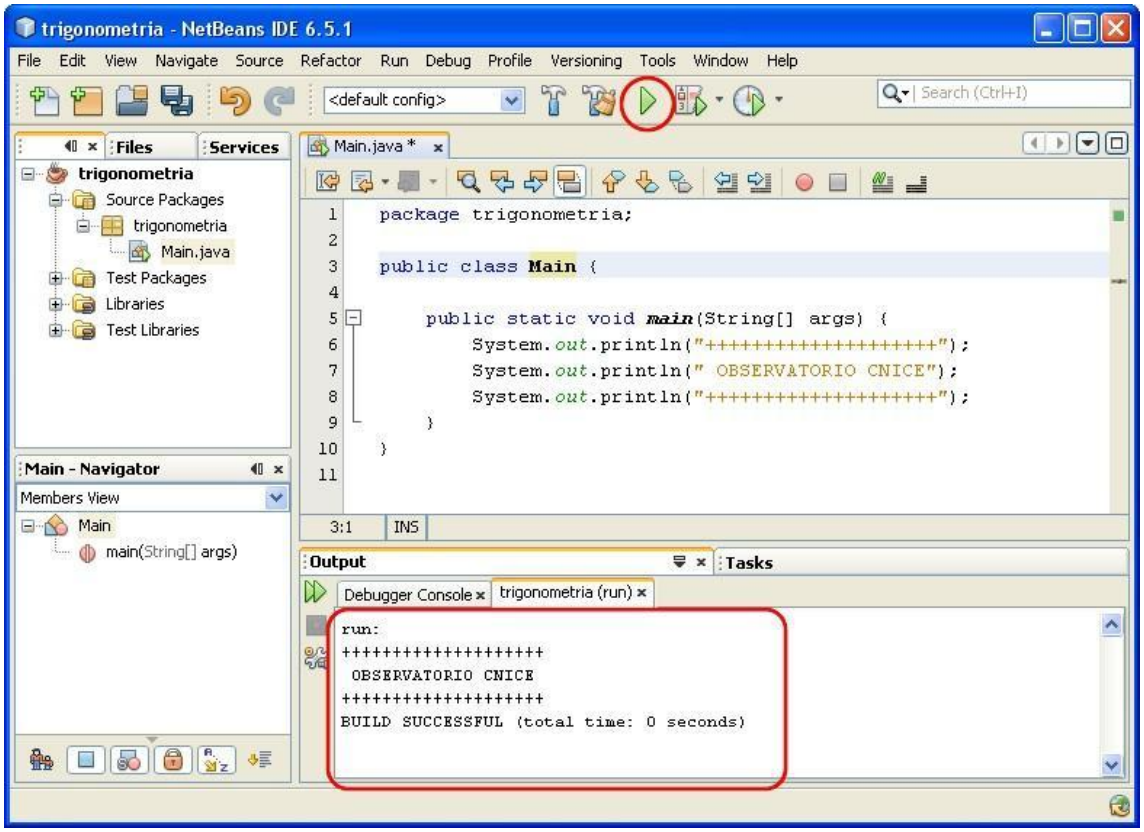
**Entorno de NETBANS, cuadro de herramientas, Control ingresos y salidas de textos. Propiedades y Eventos**

En una ventana posterior daremos nombre al proyecto y escogeremos una ubicación en el disco para almacenarlo. También nos dará la opción para crear automáticamente una clase principal, la que hasta ahora hemos llamado Test, y que aquí por defecto se llamará Main. Una vez hecho esto, la ventana de trabajo nos resultará sin duda compleja, pero también familiar:



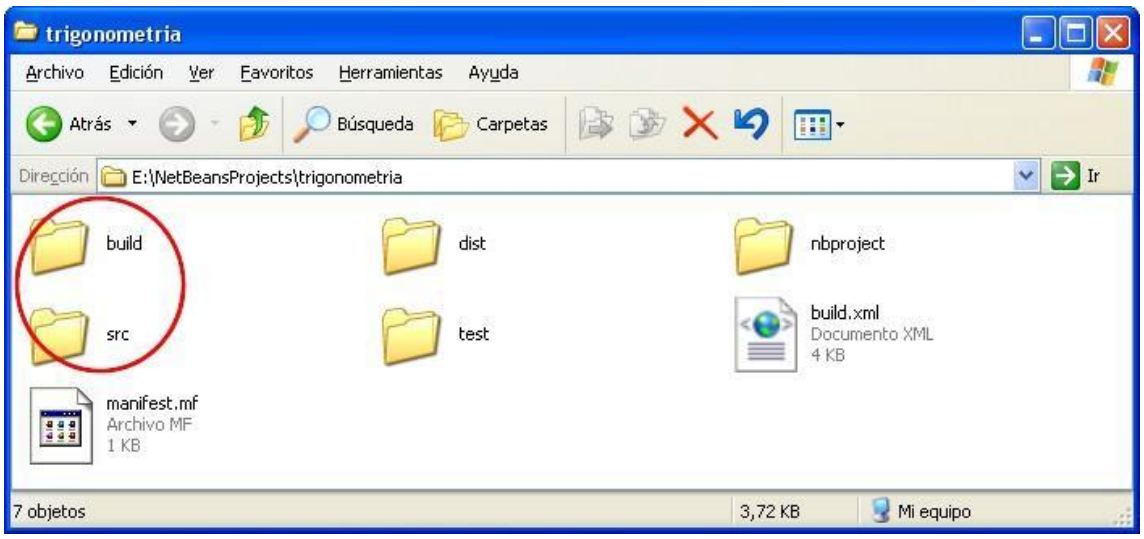

NetBeans nos ofrece un esqueleto de clase Main, que podemos utilizar para hacer nuestra primera prueba. Por defecto, NetBeans compila automáticamente los archivos en el momento en que los grabas (icono del diskette), de forma que únicamente

necesitamos utilizar el botón que representa una flecha verde, como el *play* de un mando a distancia. Verás el resultado de la ejecución en la ventana inferior:

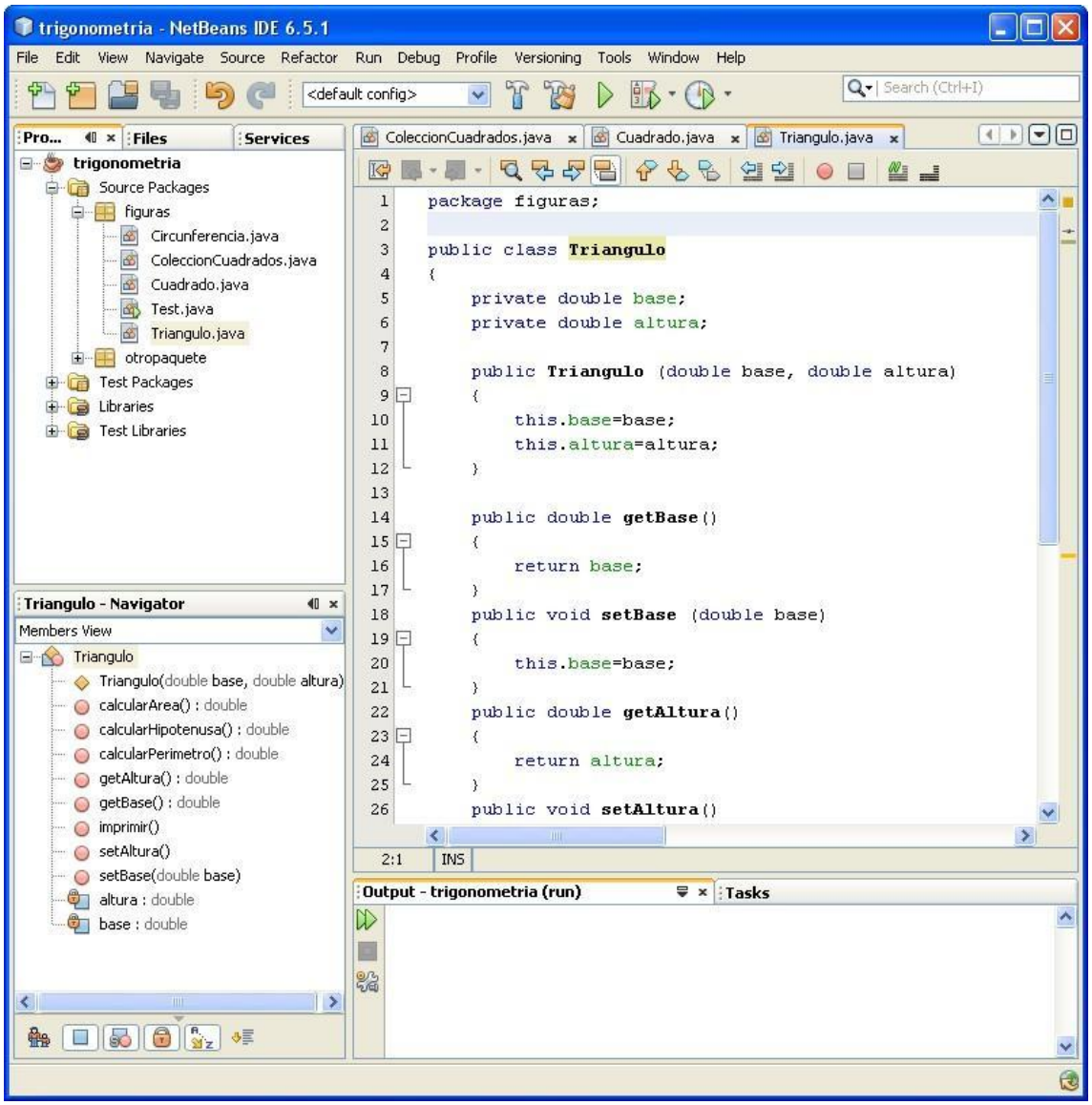



Como ves, no ha sido nada difícil empezar a trabajar con NetBeans; en la siguiente sección esbozaremos las capacidades de esta potente aplicación.

Examinemos en detalle la carpeta de proyecto. Verás que se han creado múltiples subcarpetas en tu directorio de trabajo:



De ellas sólo nos van a interesar dos: en *src* se encuentran los archivos fuente de nuestro programa, mientras que en *build* es donde guarda NetBeans los archivos compilados (*.class*). Para añadir una nueva clase al proyecto sólo tienes que utilizar el botón *New Class*; si lo que quieres es agregar al proyecto clases ya existentes, la forma más rápida consiste en copiarlas directamente a la carpeta *src*.

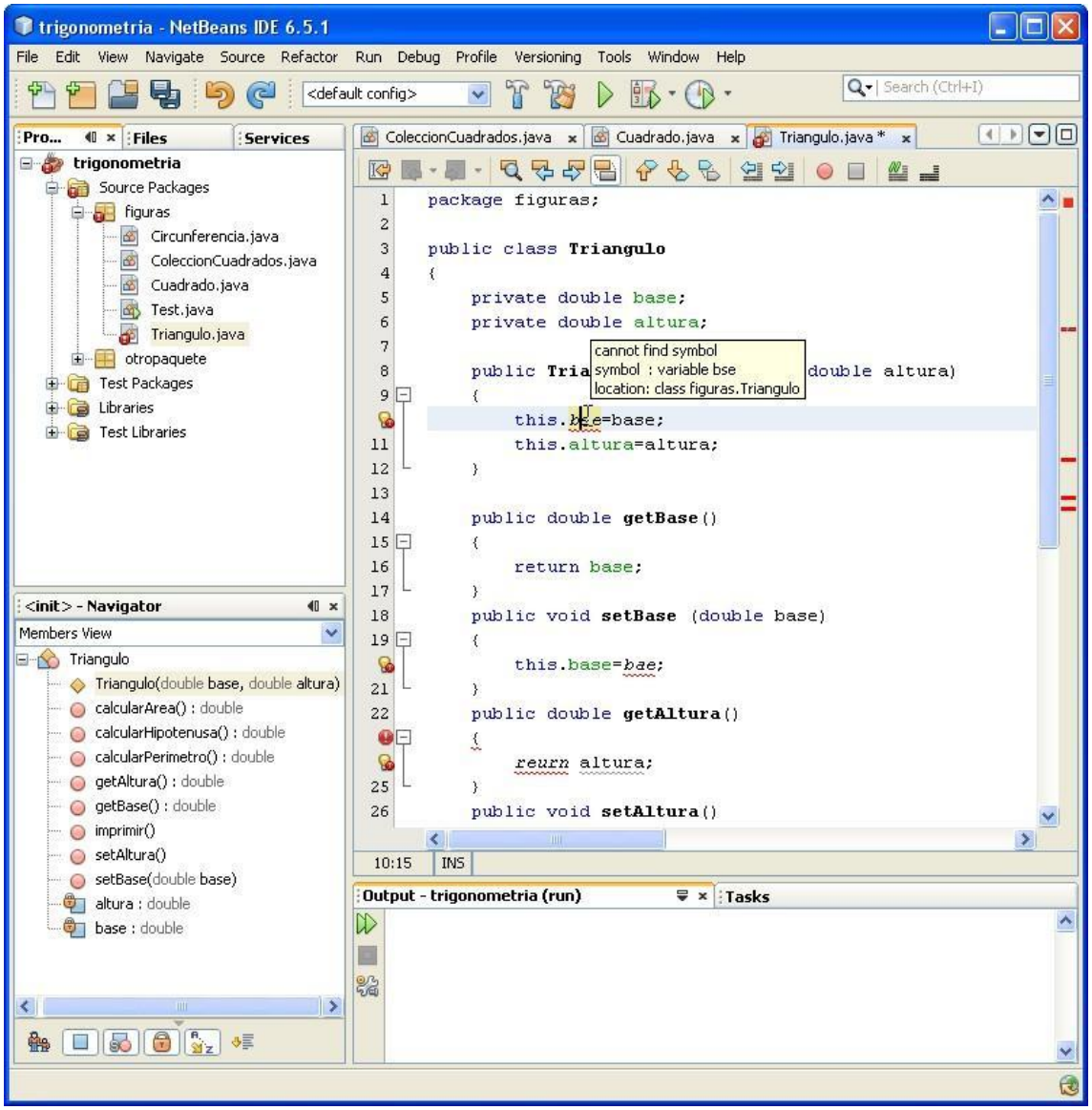



Si examinamos en detalle la ventana de trabajo, apreciaremos lo siguiente:

- A diferencia de los entornos anteriormente estudiados, en NetBeans puedes ver las clases en la parte izquierda agrupadas por el paquete al que pertenecen
- La clase que contiene el método principal main está destacada con una flecha verde en su icono

- La parte inferior izquierda es como un diagrama de clase: incluye toda la información sobre atributos y métodos de la clase que estemos editando
- NetBeans también puede mostrar números de línea, mediante el menú View > Show line numbers.

Como se ha dicho antes, en NetBeans no existe la necesidad explícita de compilar, puesto que cuenta con una función de autocompilado. Piensa en un editor de textos avanzado: según vas escribiendo, el programa comprueba tu ortografía y, si has cometido una falta, la destaca en rojo sobre la marcha. NetBeans funciona exactamente igual; es más, la forma en que resalta los errores sigue la misma estética que los correctores ortográficos. Si queremos saber el motivo del problema, bastará con situar el cursor sobre una de las marcas de error:



En jGrasp o BlueJ, el primer error de compilación provocará la detención del proceso: aquí puedes seguir trabajando aunque se acumulen varios errores, corrigiéndolos después en el orden que quieras. Fíjate en la parte izquierda de la ventana: NetBeans distinguirá con iconos aquellas clases que presentan errores de compilación. Y puesto que la clase pertenece al paquete *figuras*, también marca el paquete completo como erróneo. De igual forma, el proyecto *trigonometría* queda marcado en rojo por la existencia de al menos un error en sus archivos.

Sin duda, y pese a la complejidad, resulta muy cómodo programar en NetBeans. Se trata de un completo entorno que nos permitirá realizar no sólo programas sencillos sino auténticos proyectos profesionales de programación. Como todos los IDEs dirigidos a programadores profesionales, ofrece herramientas avanzadas que nosotros, en un curso de iniciación a Java, raramente vamos a utilizar. Pero para que te hagas una idea de lo que estamos hablando, repasaremos algunas de sus posibilidades:

- Permite la integración de la herramienta Ant de Apache, que permite automatizar mediante guiones o scripts la compilación de proyectos complejos y realizarla después dándole a un solo botón. Las posibilidades de Ant son enormes, permitiendo especificar, por ejemplo, que los archivos fuente se almacenan en unas determinadas carpetas mientras los compilados se deben copiar a otras. También se pueden programar scripts de limpieza que eliminen todos los archivos compilados o no utilizados.

- Soporta control de versiones con la herramienta gratuita CVS. Esta funcionalidad es extremadamente útil al programar en equipo, y básicamente consiste en tener un único almacén de archivos compartido por todos los colaboradores de un proyecto de programación. El sistema registra todos los cambios realizados de forma que cada programador puede ver resaltado en colores qué cambios ha hecho su compañero en cada archivo. Cuando el equipo se da cuenta de que ha cometido un error, puede dar marcha atrás y recuperar rápidamente una versión anterior, deshaciendo los cambios realizados. Más aún: dos programadores pueden modificar un mismo archivo a la vez. Cuando el segundo intente devolver el archivo modificado al almacén, el programa le facilitará la revisión para *mezclar* sus cambios con los de su compañero, siempre que no resulten incompatibles.

- Ofrece la posibilidad de refactorizar código. Utilizamos esta palabra para designar cambios menores en el código que, sin modificar la funcionalidad de nuestro programa, lo simplifican o facilitan su legibilidad. El ejemplo más básico de refactorización es cambiar el nombre de una variable: el entorno se ocupará de realizar el cambio en todos los puntos del programa en que aparezca, incluso en otros archivos del proyecto. NetBeans también permite ordenar automáticamente el código, introduciendo tabulaciones y espaciados para que el programa se lea con facilidad.

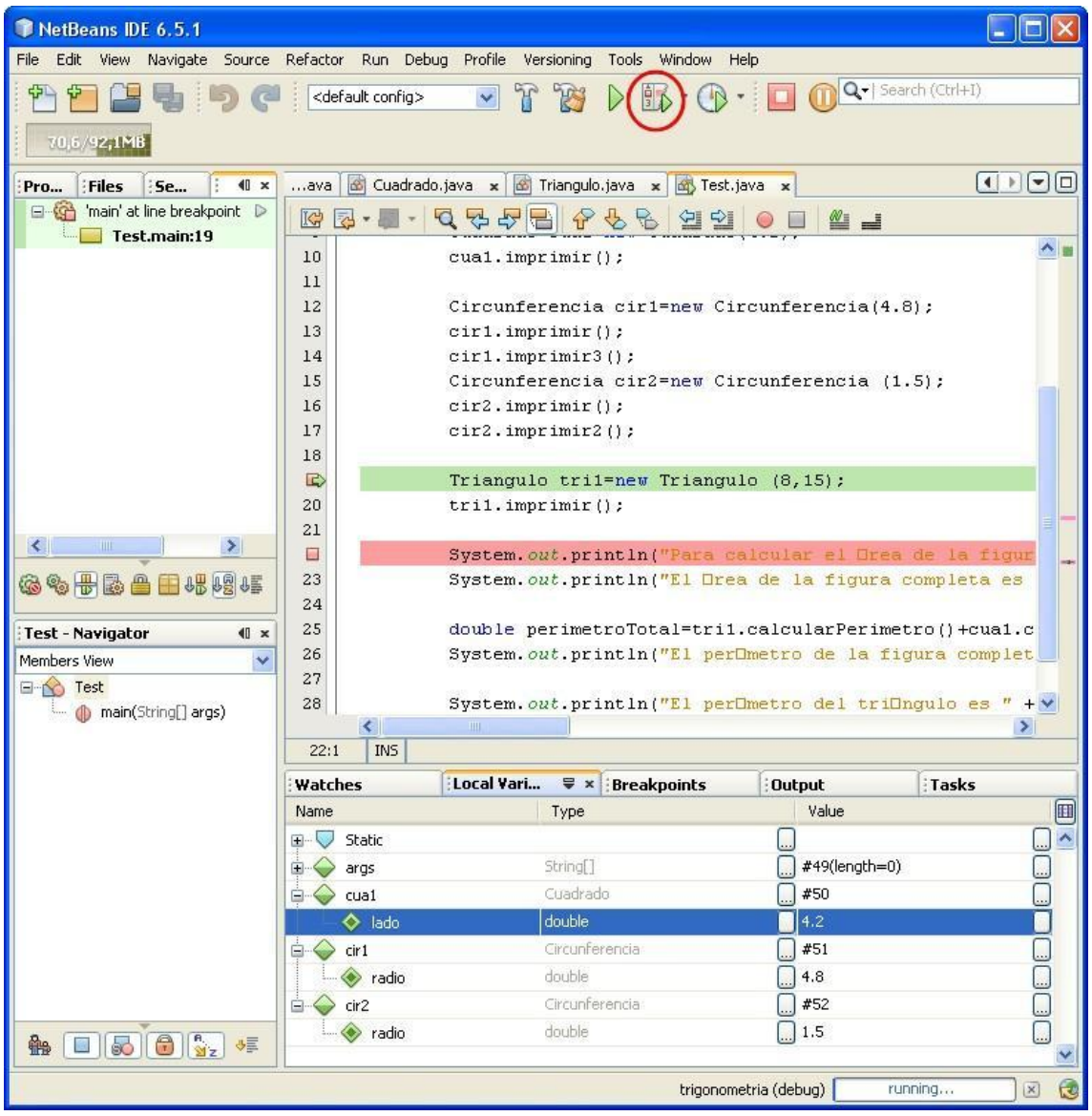
- Incorpora el autocompletado de código. Imagina que estás escribiendo el nombre de una variable o de un objeto, pero no recuerdas qué nombre le diste exactamente: pues bien, simplemente escribe las primeras letras, pulsa la tecla Control y la barra espaciadora, y el programa te indicará cuáles son los nombres disponibles. Algo parecido sucede con los atributos y métodos de una clase. Si escribimos el nombre de un objeto seguido de un punto, sabemos que a continuación debe ir o bien un atributo o bien un método de los que ofrezca la clase a la que pertenece el objeto en cuestión. Si no recordamos su nombre repetiremos la combinación de teclado y el entorno nos ofrecerá las distintas posibilidades:




Las dos primeras funcionalidades están muy orientadas al uso profesional y tendría poco sentido utilizarlas en un curso de iniciación, pero es posible que algún profesor quiera ilustrar las posibilidades de trabajo en equipo incorporando un servidor de control de versiones. El autocompletado, sin embargo, puede ser utilizado en clase con frecuencia; otra cuestión sería el considerarlo o no adecuado para la labor docente. La función de NetBeans es permitir programar deprisa, y funciones como el formateado automático del texto y el autocompletado de código ahorran al programador tiempo y errores. Algunos profesores considerarán que, cuando empezamos, no deberíamos recibir tantas ayudas y sino enfrentarnos precisamente a esos errores. Otros valorarán estas herramientas para que los alumnos se sientan más guiados en su tarea de programar.

NetBeans no permite la creación interactiva de objetos (una iniciativa interesante a nivel didáctico, pero no tanto en un IDE profesional). Pero podemos completar nuestro

paseo por el entorno comprobando cómo funciona la depuración. La forma de realizarla es igual que en los otros entornos, aunque NetBeans proporciona posibilidades de inspección y monitorización mucho más avanzadas. Lo que a nosotros nos interesa, sin embargo, se realiza de forma similar: haz click en la columna gris a la izquierda de las líneas donde quieras añadir puntos de ruptura, y pulsa el icono de depurar programa:




En verde se destaca el punto de ruptura en el que estamos detenidos, y en rojo otros puntos de ruptura que hayamos marcado. En la ventana inferior podremos examinar los valores de todos los objetos en este punto, y a través del menú Debug encontraremos las opciones ya conocidas para controlar la ejecución (*step, step into*...) aparte de otras posibilidades que sobrepasan nuestros intereses.

Los menús y mensajes de NetBeans están escritos, al igual que su documentación, en inglés. Sin embargo, existe un grupo de trabajo dedicado a traducir el IDE al español, de forma que algunas versiones se encuentran traducidas a nuestro idioma. En cuanto a la documentación, iniciativas personales o universitarias de España o Iberoamérica permiten encontrar en la red ciertos recursos para comprender mejor esta popular y potente plataforma.

**Ejemplo**

Supongamos que deseamos crear una interfaz simple que solicita al usuario ingresar su nombre y luego muestra un saludo. Utilizaremos JTextField y JLabel para la entrada y salida de texto, respectivamente. El código en NetBeans podría verse así:

```
import javax.swing.*;


public class MiVentana extends JFrame {
    private JLabel etiqueta;
    private JTextField campoTexto;


    public MiVentana() {
        // Configuración de la ventana
        setTitle("Saludo");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());


        // Componentes
        etiqueta = new JLabel("Ingrese su nombre:");
```

```
        campoTexto = new JTextField(15);


        // Agregar componentes a la ventana
        add(etiqueta);
        add(campoTexto);


        // Mostrar la ventana
        setVisible(true);
    }


    public static void main(String[] args) {
        new MiVentana();
    }
}
```

**Control Lista Desplegable (JComboBox)**

El control JButton es un botón interactivo que puede desencadenar eventos en respuesta a las acciones del usuario. Por otro lado, JOptionPane es una clase que proporciona cuadros de diálogo (ventanas emergentes) para interactuar con el usuario. Estos controles son esenciales para crear interfaces dinámicas y responder a eventos específicos, como clics de botones.

**Ejemplo**:

```
import javax.swing.*;


public class ListaDesplegableEjemplo {
```

```
    public static void main(String[] args) {
        JFrame frame = new JFrame("Lista Desplegable");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        String[] opciones = {"Opción 1", "Opción 2", "Opción 3"};
        JComboBox<String> comboBox = new JComboBox<>(opciones);

        frame.getContentPane().add(comboBox);
        frame.pack();
        frame.setVisible(true);
    }
}
```

**JTable:**

JTable es un componente para mostrar datos en forma de tabla. Permite organizar información en filas y columnas. Sus propiedades incluyen la capacidad de personalizar el modelo de datos y gestionar eventos como la selección de celdas.

**Ejemplo:**

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;

public class TablaEjemplo {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Tabla");
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        String[] columnas = {"Nombre", "Edad", "Ciudad"};

        Object[][] datos = {{"Juan", 25, "Madrid"}, {"María", 30, "Barcelona"}, {"Carlos", 22,
"Valencia"}};


        JTable table = new JTable(new DefaultTableModel(datos, columnas));


        JScrollPane scrollPane = new JScrollPane(table);

        frame.getContentPane().add(scrollPane);

        frame.pack();

        frame.setVisible(true);

    }

}
```

**Estructura del Lenguaje de Programación en JAVA**

La estructura básica de un programa en Java se compone de clases y métodos. Una clase es la estructura principal que encapsula el código, y un método, particularmente el método main, es el punto de entrada del programa. Dentro de un método, se escriben las instrucciones para realizar diversas operaciones.

**Ejemplo**:

```
public class MiPrograma {

    public static void main(String[] args) {

        // Código del programa

        System.out.println("¡Hola, mundo!");
```

```
    }
}
```

En este ejemplo, MiPrograma es una clase con un método main que imprime "¡Hola, mundo!" en la consola.

**Uso de la Instrucción If-ElseIf-EndIf en JAVA**

La instrucción if en Java se utiliza para tomar decisiones basadas en una condición. Si la condición es verdadera, se ejecutan las instrucciones dentro del bloque if. Si no es verdadera, se pueden especificar condiciones adicionales con else if. El bloque else se ejecuta si ninguna de las condiciones anteriores es verdadera.

**Ejemplo**:

```
public class EjemploIfElseIf {
    public static void main(String[] args) {
        int edad = 20;

        if (edad < 18) {
            System.out.println("Eres menor de edad.");
        } else if (edad >= 18 && edad < 65) {
            System.out.println("Eres adulto.");
        } else {
            System.out.println("Eres un adulto mayor.");
        }
    }
}
```

Este ejemplo verifica la edad de una persona e imprime un mensaje según la categoría de edad.

**Instrucciones Select-Switch**

La instrucción switch en Java se utiliza para realizar múltiples acciones basadas en el valor de una expresión. Proporciona una alternativa más eficiente y clara en comparación con múltiples instrucciones if-else. Cada caso representa un valor específico que puede coincidir con el valor de la expresión.

**Ejemplo**:

```
public class EjemploSwitch {

    public static void main(String[] args) {

        int opcion = 2;

        switch (opcion) {

            case 1:

                System.out.println("Seleccionaste la opción 1");

                break;

            case 2:

                System.out.println("Seleccionaste la opción 2");

                break;

            case 3:

                System.out.println("Seleccionaste la opción 3");

                break;

            default:

                System.out.println("Opción no válida");

        }

    }

}
```

## ACTIVIDADES PRÁCTICAS

**Tabla**

Detalle de la Práctica

| Practicas | Instrucciones | Duración de la Practica | Mecanismo de Evaluación |
|---|---|---|---|
| **Foro de debate** | Revisar Actividades Prácticas | 0:30 mm | Por rúbricas |
| **Práctica investiga: Taller/Ensayo/Investigación** | Revisar Actividades Prácticas | 3:30 h | Por rúbricas |
| **Tutorías Prácticas** | Revisar Actividades Prácticas | 2:00 mm | Por participación en clase |
| **Test evaluativo** | Revisar Actividades Prácticas | 0:30 mm | Formativa |

*Nota.* Elaboración propia

**Consideraciones Generales**

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

**FORO DE DEBATE**

**Tema:** Elementos u objetos de presentación de datos en java.

**Instrucciones para Participar:**

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

**PRÁCTICA INVESTIGATIVA**

**Taller:** No aplica para este tema.

**Consigna:**

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

**PRÁCTICA TEST**

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

**Consideraciones para realizar el test**

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

## III. DATOS GENERALES DE LA UNIDAD 3

### Información General

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.1. | Nivel | Segundo | | | | | |
| 1.2. | Nombre Unidad | Controles básicos y lenguaje de programación Java entorno Netbeans | | | | | |
| 1.3. | Tema 1 – Unidad | Entorno de IDE de POO | | | | | |
| 1.4. | Tema 2 – Unidad | Objetos de Presentación de Información | | | | | |
| 1.5. | Unidad organizacional | Unidad Profesional | | | | | |
| 1.6. | Unidad | 3 | | | | | |
| 1.7. | Total, Horas Unidad | 15 | | | | | |
| 1.8. | Detalle de horas Unidad | ACD | 6 | AA | 4 | APE | 5 |

### Resultado de Aprendizaje

Comprender y aplicar las estructuras de control fundamentales en Java, incluyendo if, else, if-else if, y switch, para tomar decisiones condicionales. Los estudiantes también se involucran en trabajos en grupo y mantienen diarios de reflexión. Se fomenta la participación en debates y preguntas y respuestas, y se proporcionan recursos en línea para una comprensión más profunda. La evaluación se realiza mediante pruebas, proyectos y una evaluación final, con retroalimentación continua para guiar el aprendizaje. Esta metodología busca que los estudiantes comprendan los conceptos fundamentales y puedan aplicar eficazmente las metodologías de desarrollo de software en situaciones prácticas.

### Metodología

La Metodología de se basa en un enfoque activo y colaborativo. Incluye sesiones de clase estructuradas en torno a temas específicos, lecturas y tareas, estudio de casos, proyectos prácticos y debates en clase. Los estudiantes también se involucran en trabajos en grupo y mantienen diarios de reflexión. Se fomenta la participación en debates y preguntas y respuestas, y se proporcionan recursos en línea para una comprensión más profunda. La evaluación se realiza mediante pruebas, proyectos y una evaluación final, con retroalimentación continua para guiar el aprendizaje.

**DESARROLLO**

**Unidad 3: Estructuras de control y bucles en Java**

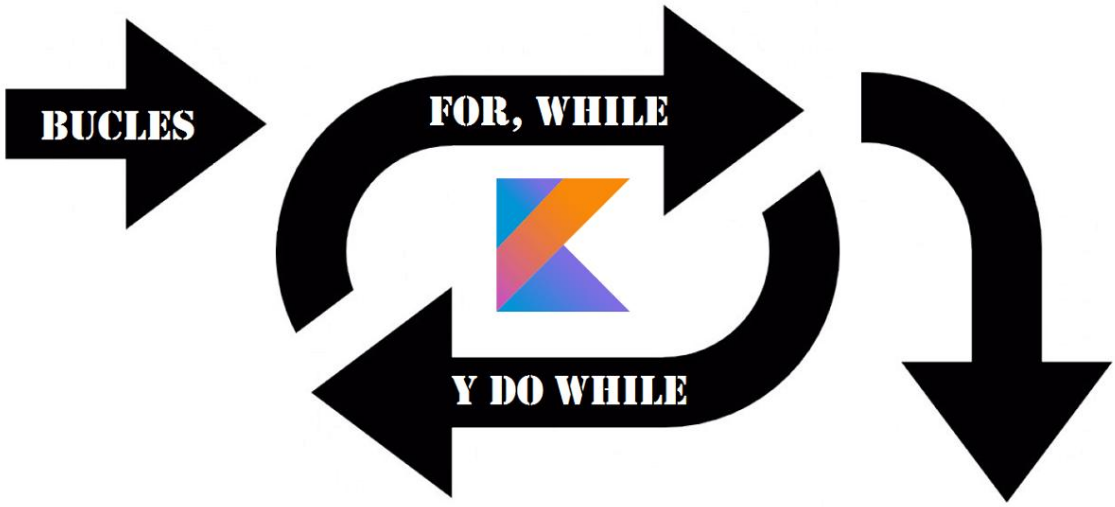
**Tema 1: Estructuras de Control**

**Objetivo:** Analizar las estructuras de control en Java, comprendiendo su funcionamiento, tipos y aplicaciones, con el propósito de optimizar la toma de decisiones y el flujo de ejecución en el desarrollo de programas eficientes y estructurados.

**Introducción Ciclos repetitivos**

Los ciclos repetitivos, también conocidos como bucles, son estructuras fundamentales en programación que permiten ejecutar un bloque de código varias veces. En Java, existen tres tipos principales de ciclos: for, while y do-while. Estos bucles son esenciales para la implementación de algoritmos que requieren ejecución repetitiva.

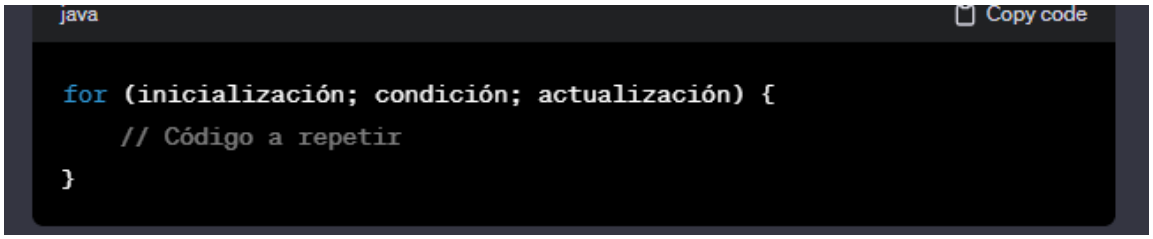
**Imagen 1**: Ciclos Repetitivos




**Fuente**: https://programandoointentandolo.com/2017/11/bucles-kotlin-for-while-y-do-while.html

**1. Ciclo for:**

El ciclo for es ideal cuando el número de repeticiones es conocido de antemano. Su sintaxis es:




**Ejemplo:**

```
for (int i = 0; i < 5; i++) {
    System.out.println("Iteración " + i);
}
```

**2. Ciclo while:**

El ciclo while se utiliza cuando la condición de repetición se evalúa antes de ejecutar el bloque de código. Su estructura es:

```
while (condición) {
    // Código a repetir
}
```

**Ejemplo**

```
int i = 0;
while (i < 5) {
    System.out.println("Iteración " + i);
    i++;
}
```

**3. Ciclo do-while:**

El ciclo do-while garantiza que el bloque de código se ejecute al menos una vez, ya que la condición se evalúa después de la ejecución. Su sintaxis es:

```
do {
```

```
    // Código a repetir
} while (condición);
```

**Ejemplo:**

```
int i = 0;
do {
    System.out.println("Iteración " + i);
    i++;
} while (i < 5);
```

**Comparación y Elección del Bucle Adecuado:**

**Criterios**:

- Si se conoce el número exacto de iteraciones, se prefiere el bucle for.
- Cuando la condición se evalúa antes de la ejecución, se opta por el bucle while.
- Si se necesita garantizar al menos una ejecución, se elige el bucle do-while.

Los ciclos repetitivos en Java son esenciales para la ejecución controlada de bloques de código. La elección entre for, while y do-while depende de la naturaleza del problema. La comprensión y dominio de estas estructuras son fundamentales para el desarrollo efectivo de algoritmos en Java.

**Uso de contadores y acumuladores en Java**

Los contadores y acumuladores son variables clave en programación utilizadas para contar repeticiones o acumular sumas. Estos conceptos son fundamentales para implementar algoritmos que requieren llevar un registro de eventos o sumar valores de manera iterativa.

```
var n = 1;
var suma = 0;
var num;

while (n <= 3)
{
  num = prompt('Introduzca numero:','');

  num = parseInt(num);
  suma = suma + num;
  n = n + 1;
}

document.write("La suma de todos los numeros es: " + suma);
```

**Imagen 2**: Acumuladores y Contadores

**Fuente**: https://www.aulafacil.com/cursos/programacion/javascript/uso-de-acumuladores-l7657

**1. Contadores:**

Un contador es una variable que se incrementa o decrementa en cada iteración de un bucle, generalmente utilizado para llevar un registro de la cantidad de repeticiones.

**Ejemplo: Contador con Bucle for:**

```
int contador = 0;

for (int i = 0; i < 5; i++) {

    contador++;

}

System.out.println("El valor del contador es: " + contador);
```

**2. Acumuladores:**

Un acumulador es una variable que se actualiza sumando o restando valores en cada iteración, utilizado para calcular la suma total de una secuencia de valores.

**Ejemplo: Acumulador con Bucle while:**

```
int acumulador = 0;
```

```
int i = 1;

while (i <= 5) {

    acumulador += i;

    i++;

}

System.out.println("El valor acumulado es: " + acumulador);
```

**Comparación y Elección del Concepto Adecuado**

**Contadores**:

Se utilizan para llevar un registro del número de iteraciones.

Comúnmente empleados con bucles for cuando el número de repeticiones es conocido de antemano.

**Acumuladores**:

Se utilizan para calcular la suma total de valores en una secuencia.

Se aplican en bucles while o for, acumulando valores en cada iteración.

Contadores y acumuladores son herramientas esenciales en programación para llevar un control numérico de repeticiones y calcular sumas iterativas. Su elección depende del propósito específico del algoritmo y del tipo de bucle que se utilice. El uso efectivo de estos conceptos es crucial para el diseño eficiente de algoritmos en Java.

**Uso de While y Do While en Java**

Las estructuras de control do-while y while en Java son utilizadas para implementar ciclos repetitivos. Ambas sirven para ejecutar un bloque de código mientras se cumpla una condición, pero difieren en cuándo se evalúa la condición.



```java
import java.util.Scanner;
/**
 *
 * @author Código Root >_
 */
public class TablaDoWhile {

     public static void main (String [] args){
        int Multiplicando ;


             System.out.println("¿Que tabla desea Realizar?");
Scanner valor1 = new Scanner(System.in);
Multiplicando = valor1.nextInt();

int multiplicador = 1;
do {   System.out.println (Multiplicando + " X " + multiplicador + " = " + Multiplicando

                    multiplicador += 1;

        } while (multiplicador<=10);
    }

}

```

**Imagen 3**: While – Do While

**Fuente**: https://codigoroot.net/blog/tablas-de-multiplicar-con-ciclo-do-while-java/

**1. Estructura do-while:**

La estructura do-while garantiza que el bloque de código se ejecute al menos una vez, ya que la condición se evalúa después de la ejecución del bloque.

**Sintaxis:**

```
do {
    // Código a repetir
} while (condición);
```

**Ejemplo: Ciclo do-while que imprime números del 1 al 5:**

```
int i = 1;
do {
    System.out.println(i);
    i++;
} while (i <= 5);
```

**2. Estructura while:**

La estructura while evalúa la condición antes de ejecutar el bloque de código. Si la condición es falsa desde el principio, el bloque de código no se ejecutará.

**Sintaxis**:

```
while (condición) {
    // Código a repetir
}
```

**Ejemplo: Ciclo while que imprime números del 1 al 5:**

```
int i = 1;
while (i <= 5) {
    System.out.println(i);
    i++;
}
```

**Comparación y Elección de Estructura**

**Do-While:**

Se utiliza cuando se desea que el bloque de código se ejecute al menos una vez.

Útil cuando se desea realizar una acción y luego verificar la condición.

**While:**

Se emplea cuando la condición se debe verificar antes de la ejecución del bloque.

Adecuado cuando el bloque de código puede no ejecutarse si la condición inicialmente no se cumple.

**Resumen**:

Las estructuras de control do-while y while son esenciales para implementar ciclos repetitivos en Java. La elección entre ellas depende de si se quiere garantizar al

menos una ejecución del bloque de código (do-while) o si la ejecución depende de la condición desde el principio (while). Ambas son fundamentales en el diseño de algoritmos y la implementación de lógica iterativa en programas Java.

**Sentencias for y ForEach iteraciones**

Las sentencias for y forEach en Java son utilizadas para realizar iteraciones sobre elementos de colecciones o secuencias de datos. Estas estructuras proporcionan un enfoque más compacto y legible para la implementación de ciclos repetitivos.

```java
public class ForEach {

    public static void main(String[] args) {

        // Listar un array
        String array[] = {"Avila", "Burgos", "León", "Palencia",
"Salamanca", "Segovia", "Soria", "Valladolid", "Zamora"};

        for (String elemento: array)
          System.out.println(elemento);

        // Listar una colección
        List<String> list = new ArrayList<String>();
        list.add("Victor");
        list.add("Amaya");
        list.add("Julio");

        for (String nombre: list)
          System.out.println(nombre);

    }

}
```

**Imagen 4**: ForEach

**Fuente**: https://lineadecodigo.com/java/bucle-for-each-en-java/

**1. Sentencia for:**

La sentencia for se utiliza para ejecutar un bloque de código un número específico de veces. Su estructura consta de tres partes: inicialización, condición y actualización.

**Sintaxis**:

for (inicialización; condición; actualización) {

    // Código a repetir

}

**Ejemplo: Ciclo for que imprime números del 1 al 5:**

```
for (int i = 1; i <= 5; i++) {
    System.out.println(i);
}
```

**2. Sentencia forEach:**

La sentencia forEach se introdujo en Java 5 y simplifica la iteración sobre colecciones (Arrays, Listas, Sets, Mapas, etc.). Se utiliza con una interfaz funcional llamada Consumer.

**Sintaxis**:

```
coleccion.forEach(elemento -> {
    // Código a repetir para cada elemento
});
```

**Ejemplo: Uso de forEach para imprimir elementos de una Lista:**

```
List<String> nombres = Arrays.asList("Juan", "María", "Carlos");

nombres.forEach(nombre -> {
    System.out.println(nombre);
});
```

**Comparación y Elección de Sentencia**

**for:**

Se utiliza cuando se conoce el número exacto de iteraciones.

Es adecuado para iterar sobre rangos o secuencias numéricas.

**forEach:**

Se prefiere para la iteración sobre colecciones.

Proporciona un enfoque más declarativo y legible.

**Resumen**:

Tanto la sentencia for como forEach son herramientas valiosas en Java para la implementación de iteraciones. La elección entre ellas depende de la naturaleza del problema y la estructura de datos sobre la cual se está iterando. Mientras que for es ideal para iteraciones numéricas y controladas, forEach ofrece un enfoque más conciso y legible para la iteración sobre colecciones.

**Uso de Try Catch. Controlador de ejecución de errores en una aplicación y excepciones**

La estructura Try-Catch en Java es fundamental para el manejo de excepciones, permitiendo a los desarrolladores controlar y gestionar errores durante la ejecución de un programa. Proporciona un mecanismo para identificar, procesar y responder a situaciones excepcionales que podrían ocurrir durante la ejecución del código.

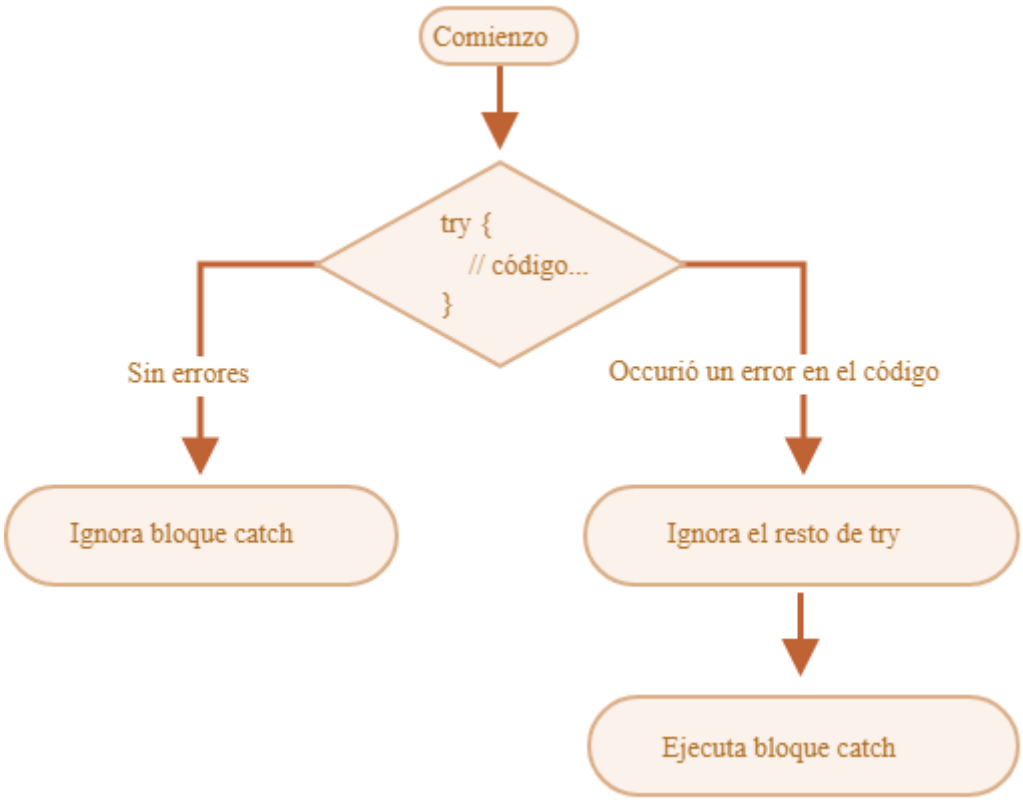


**Imagen 5**: ForEach

**Fuente**: https://es.javascript.info/try-catch

**1. Bloque try:**

El bloque try contiene el código que puede generar una excepción. Este bloque se monitorea en busca de excepciones y, si se produce alguna, el control se transfiere al bloque catch.

**Sintaxis**:

```
try {
    // Código que puede generar una excepción
} catch (TipoDeExcepcion e) {
    // Manejo de la excepción
}
```

**Ejemplo: Manejo de una Excepción de División por Cero:**

```
try {
    int resultado = 10 / 0; // Intento de división por cero
} catch (ArithmeticException e) {
    System.out.println("Error: " + e.getMessage());
}
```

**2. Bloque catch:**

El bloque catch se ejecuta si y solo si una excepción del tipo especificado en los paréntesis ocurre en el bloque try. Aquí se maneja la excepción.

**3. Bloque finally:**

Opcionalmente, se puede agregar un bloque finally que se ejecutará siempre, independientemente de si se produce o no una excepción. Se utiliza para realizar acciones de limpieza o liberación de recursos.

**Sintaxis con finally:**

```
try {
    // Código que puede generar una excepción
} catch (TipoDeExcepcion e) {
    // Manejo de la excepción
} finally {
    // Código que se ejecuta siempre
}
```

**Ejemplo con finally:**

```
try {
    // Código que puede generar una excepción
} catch (Exception e) {
    // Manejo de la excepción
} finally {
    System.out.println("Este bloque se ejecuta siempre, incluso si hay una excepción.");
}
```

**Comparación y Elección de Uso**

**try-catch:**

Se utiliza para envolver secciones de código que podrían lanzar excepciones.

Permite manejar las excepciones y continuar la ejecución del programa.

**finally**:

Se utiliza para especificar el código que se ejecutará independientemente de si se produce una excepción o no.

Se utiliza comúnmente para la liberación de recursos.

**Resumen**:

El uso de Try-Catch en Java proporciona un mecanismo robusto para manejar excepciones y controlar la ejecución del programa en situaciones de error. La estructura try-catch permite a los desarrolladores escribir código defensivo que gestiona situaciones imprevistas, mejorando la robustez y fiabilidad de las aplicaciones Java.

## ACTIVIDADES PRÁCTICAS

**Tabla**

Detalle de la Práctica

| Practicas | Instrucciones | Duración de la Practica | Mecanismo de Evaluación |
|---|---|---|---|
| **Foro de debate** | Revisar Actividades Prácticas | 0:30 mm | Por rúbricas |
| **Práctica investiga: Taller/Ensayo/Investigación** | Revisar Actividades Prácticas | 3:30 h | Por rúbricas |
| **Tutorías Prácticas** | Revisar Actividades Prácticas | 2:00 mm | Por participación en clase |
| **Test evaluativo** | Revisar Actividades Prácticas | 0:30 mm | Formativa |

*Nota.* Elaboración propia

**Consideraciones Generales**

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

**FORO DE DEBATE**

**Tema:** Estructuras de control. Según tu criterio

- ✓ ¿Qué estructura escogerías para ejecutar un proceso repetitivo en tu empresa?

**Instrucciones para Participar:**

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

**PRÁCTICA INVESTIGATIVA**

**Taller:** Ejercicios con estructuras condicionales if – if else switch.

- ✓ Desarrollar la lógica y adaptarse a la sintaxis de las estructuras de control del lenguaje de programación de JAVA, mediante la resolución de casos de estudio.

**Consigna:**

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

**PRÁCTICA TEST**

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

**Consideraciones para realizar el test**

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

**Tema 2: Variables y Métodos en POO**

**Definición y Propósito:**

En la programación orientada a objetos (POO) de Java, las variables y los métodos son elementos fundamentales que se utilizan para modelar y representar objetos del mundo real en el código. Aquí se detalla cada uno:

**1. Variables:**

Las variables en POO son contenedores que almacenan información relacionada con un objeto. Pueden representar características o propiedades del objeto. En Java, las variables se definen con un tipo de dato y un nombre.

**Sintaxis de Declaración:**

tipoDeDato nombreDeVariable;

**Ejemplo: Definición de una Variable en Java:**

// Definición de una variable de tipo entero

int edad;

// Asignación de un valor a la variable

edad = 25;

// O en una sola línea

int altura = 180;

**2. Métodos:**

Los métodos son bloques de código que realizan acciones o proporcionan funcionalidad específica para un objeto. En Java, los métodos son definidos dentro de una clase y pueden tener parámetros y un valor de retorno.

**Sintaxis de Declaración:**

tipoDeRetorno nombreDeMetodo(tipoDeParametro1 parametro1, tipoDeParametro2 parametro2) {

```
    // Código del método

    return valorDeRetorno;

}
```

**Ejemplo: Definición de un Método en Java:**

```
// Definición de un método que suma dos números

int sumar(int a, int b) {

    return a + b;

}

// Llamada al método

int resultado = sumar(3, 4);
```

**Comparación y Contraste:**

**Variables:**

Representan las características o propiedades del objeto.

Almacenan datos que describen el estado del objeto.

Pueden tener diferentes alcances (locales, de instancia o estáticas).

**Métodos:**

Realizan acciones o proporcionan funcionalidad para el objeto.

Encapsulan el comportamiento del objeto.

Pueden ser llamados y ejecutados para realizar operaciones específicas.

**Resumen**:

En Java, las variables y los métodos son elementos clave en la POO. Las variables representan el estado del objeto, mientras que los métodos encapsulan su comportamiento. El uso efectivo de variables y métodos permite la creación de clases

y objetos con una estructura coherente y modular, facilitando el diseño y mantenimiento de programas en Java.

**Declaración de Variables, Métodos Funciones Constructores, Polimorfismo y Herencia en Programación Orientada a Objetos en Java**

**1. Declaración de Variables:**

Las variables en Java se utilizan para almacenar datos y se definen con un tipo de dato y un nombre. Pueden ser de instancia, de clase o locales.

**Ejemplo**:

```
// Variable de instancia

int edad;

// Variable de clase (static)

static String nombre;

// Variable local

double altura;
```

**2. Métodos:**

Los métodos son bloques de código que realizan acciones o proporcionan funcionalidad. Pueden tener parámetros y un valor de retorno.

**Ejemplo:**

```
// Método sin parámetros ni retorno

void saludar() {

    System.out.println("¡Hola!");

}

// Método con parámetros y retorno

int sumar(int a, int b) {
```

```
    return a + b;
}
```

**3. Funciones Constructores:**

Los constructores son métodos especiales que se llaman al crear una instancia de una clase. Inicializan los atributos del objeto.

**Ejemplo**:

```
public class Persona {
    String nombre;
    // Constructor
    public Persona(String nombre) {
        this.nombre = nombre;
    }
}
// Creación de una instancia con el constructor
Persona persona1 = new Persona("Juan");
```

**4. Polimorfismo:**

El polimorfismo permite que un objeto pueda tomar varias formas. En Java, se logra mediante la sobreescritura de métodos en las clases derivadas.

**Ejemplo**:

```
public class Animal {
    void hacerSonido() {
        System.out.println("Sonido genérico");
    }
```

```
}

public class Perro extends Animal {

    @Override

    void hacerSonido() {

        System.out.println("Guau guau");

    }

}
```

**5. Herencia:**

La herencia permite que una clase herede los atributos y métodos de otra. La clase base se llama superclase, y la clase que hereda se llama subclase.

**Ejemplo**:

```
// Superclase

public class Figura {

    void dibujar() {

        System.out.println("Dibujando figura");

    }

}

// Subclase que hereda de Figura

public class Circulo extends Figura {

    @Override

    void dibujar() {

        System.out.println("Dibujando círculo");

    }
```

}

**Comparación y Contraste:**

**Variables vs. Métodos:**

Las variables almacenan datos, mientras que los métodos realizan acciones.

Las variables describen el estado, y los métodos encapsulan el comportamiento.

**Funciones Constructores vs. Métodos:**

Los constructores inicializan objetos durante la creación, mientras que los métodos realizan operaciones en objetos existentes.

**Polimorfismo vs. Herencia:**

El polimorfismo permite que los objetos tomen múltiples formas, mientras que la herencia permite compartir atributos y métodos entre clases.

**Resumen:**

La declaración de variables, métodos, funciones, constructores, polimorfismo y herencia son conceptos clave en la programación orientada a objetos en Java. Al comprender estos conceptos, los programadores pueden diseñar sistemas modulares, reutilizables y extensibles, facilitando el desarrollo de aplicaciones robustas y mantenibles.

**Declaración y Creación de Clases, Lógica de negocio con Listas de Objetos.**

En Java, una clase es una plantilla para la creación de objetos. Contiene atributos y métodos que definen las características y comportamientos de esos objetos.

**Ejemplo:**

```
public class Persona {

    // Atributos

    String nombre;

    int edad;
```

```
    // Método
    void saludar() {
        System.out.println("Hola, soy " + nombre + " y tengo " + edad + " años.");
    }
}
```

**Creación de Objetos:**

La creación de objetos se realiza mediante el uso del operador new, seguido por el constructor de la clase.

**Ejemplo**:

```
// Creación de un objeto Persona
Persona persona1 = new Persona();
// Acceso a los atributos y métodos
persona1.nombre = "Juan";
persona1.edad = 25;
persona1.saludar();
```

**Lógica de Negocio con Listas de Objetos:**

Las listas de objetos permiten almacenar y manipular múltiples instancias de una clase. Esto es esencial para la gestión de colecciones de datos en lógica de negocio.

**Ejemplo**:

```
import java.util.ArrayList;
public class Negocio {
    // Lista de objetos Persona
```

```
    ArrayList<Persona> listaPersonas = new ArrayList<>();

    // Método para agregar persona a la lista
    void agregarPersona(String nombre, int edad) {
        Persona nuevaPersona = new Persona();
        nuevaPersona.nombre = nombre;
        nuevaPersona.edad = edad;
        listaPersonas.add(nuevaPersona);
    }

    // Método para mostrar información de todas las personas
    void mostrarPersonas() {
        for (Persona persona : listaPersonas) {
            persona.saludar();
        }
    }
}
```

**Comparación y Contraste:**

**Clases vs. Objetos:**

Una clase es la plantilla, y un objeto es una instancia específica de esa plantilla.

Las clases definen estructuras y comportamientos, mientras que los objetos representan instancias individuales.

**Creación de Objetos vs. Lógica de Negocio:**

La creación de objetos implica instanciar y configurar objetos individuales.

La lógica de negocio con listas de objetos implica la gestión y manipulación de colecciones de objetos para satisfacer requisitos comerciales específicos.

La declaración y creación de clases en Java son fundamentales para la programación orientada a objetos. Al definir clases y crear objetos, los programadores pueden modelar eficazmente entidades del mundo real. La lógica de negocio con listas de objetos permite gestionar eficientemente colecciones de datos, proporcionando una base sólida para el desarrollo de aplicaciones escalables y mantenibles.

## ACTIVIDADES PRÁCTICAS

**Tabla**

Detalle de la Práctica

| Practicas | Instrucciones | Duración de la Practica | Mecanismo de Evaluación |
|---|---|---|---|
| **Foro de debate** | Revisar Actividades Prácticas | 0:30 mm | Por rúbricas |
| **Práctica investiga: Taller/Ensayo/Investigación** | Revisar Actividades Prácticas | 3:30 h | Por rúbricas |
| **Tutorías Prácticas** | Revisar Actividades Prácticas | 2:00 mm | Por participación en clase |
| **Test evaluativo** | Revisar Actividades Prácticas | 0:30 mm | Formativa |

*Nota.* Elaboración propia

**Consideraciones Generales**

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

**FORO DE DEBATE**

**Tema:** Según tu criterio Define una diferencia entre Método y Variable en Programación Orientada a Objetos en JAVA.

**Instrucciones para Participar:**

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

**PRÁCTICA INVESTIGATIVA**

**Taller:** Ejercicios con objetos en java utilizando métodos y variables.

Desarrollar la lógica y adaptarse a la sintaxis de las estructuras de control del lenguaje de programación de JAVA, mediante la resolución de casos de estudio.

**Consigna:**

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

**PRÁCTICA TEST**

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

**Consideraciones para realizar el test**

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

## IV. DATOS GENERALES DE LA UNIDAD 4

### Información General

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.1. | Nivel | Segundo | | | | | |
| 1.2. | Nombre Unidad | Controles básicos y lenguaje de programación Java entorno Netbeans | | | | | |
| 1.3. | Tema 1 – Unidad | Entorno de IDE de POO | | | | | |
| 1.4. | Tema 2 – Unidad | Objetos de Presentación de Información | | | | | |
| 1.5. | Unidad organizacional | Unidad Profesional | | | | | |
| 1.6. | Unidad | 4 | | | | | |
| 1.7. | Total, Horas Unidad | 13 | | | | | |
| 1.8. | Detalle de horas Unidad | ACD | 4 | AA | 4 | APE | 8 |

### Resultado de Aprendizaje

Comprender y aplicar las estructuras de control fundamentales en Java, incluyendo if, else, if-else if, y switch, para tomar decisiones condicionales. La Metodología de se basa en un enfoque activo y colaborativo. Incluye sesiones de clase estructuradas en torno a temas específicos, lecturas y tareas, estudio de casos, proyectos prácticos y debates en clase. Los estudiantes también se involucran en trabajos en grupo y mantienen diarios de reflexión. Se fomenta la participación en debates y preguntas y respuestas, y se proporcionan recursos en línea para una comprensión más profunda. La evaluación se realiza mediante pruebas, proyectos y una evaluación final, con retroalimentación continua para guiar el aprendizaje. prácticas.

### Metodología

La Metodología de se basa en un enfoque activo y colaborativo. Incluye sesiones de clase estructuradas en torno a temas específicos, lecturas y tareas, estudio de casos, proyectos prácticos y debates en clase. Los estudiantes también se involucran en trabajos en grupo y mantienen diarios de reflexión. Se fomenta la participación en debates y preguntas y respuestas, y se proporcionan recursos en línea para una comprensión más profunda. La evaluación se realiza mediante pruebas, proyectos y una evaluación final, con retroalimentación continua para guiar el aprendizaje.

### DESARROLLO

## DESARROLLO

### Unidad 4: Diseño de interfaces en Swing

### Tema 1: Introducción a la librería Swing

**Objetivo**: Explorar la librería Swing en Java, comprendiendo sus componentes, estructura y funcionalidades, con el propósito de desarrollar interfaces gráficas de usuario (GUI) interactivas y eficientes dentro del entorno de programación Java.

### Introducción a la Librería Swing

Swing es una librería gráfica incluida en el paquete javax.swing de Java. Permite la creación de interfaces gráficas de usuario (GUI) para aplicaciones Java. Swing es parte de la Plataforma de Java y proporciona componentes visuales avanzados y flexibles.

### Componentes Básicos:

### JFrame:

Es el marco principal de una aplicación Swing.

Representa la ventana de la aplicación.

```java
import javax.swing.JFrame;

public class MiVentana extends JFrame {
    public MiVentana() {
        // Configuración de la ventana
        setTitle("Mi Aplicación");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Otros componentes y lógica aquí
    }
}
```

### JPanel:

Representa un área donde se pueden agregar otros componentes.

Se utiliza para organizar y agrupar elementos en la interfaz.

```java
import javax.swing.JPanel;

public class MiPanel extends JPanel {
    // Contenido del panel y lógica aquí
}
```

**Componentes de Interfaz:**

**JButton:**

Botón interactivo que realiza una acción al hacer clic.

```java
import javax.swing.JButton;

JButton miBoton = new JButton("Haz clic");
```

**JLabel:**

Etiqueta que muestra texto o una imagen.

```java
import javax.swing.JLabel;

JLabel miEtiqueta = new JLabel("Hola, Mundo!");
```

**JTextField:**

Campo de texto donde el usuario puede ingresar información.

```java
import javax.swing.JTextField;

JTextField miCampoTexto = new JTextField("Texto por defecto");
```

**Ejemplo Completo:**

import javax.swing.JFrame;

import javax.swing.JButton;

import javax.swing.JPanel;

```
import javax.swing.JLabel;
import javax.swing.JTextField;

public class MiAplicacion extends JFrame {
    public MiAplicacion() {
        // Configuración de la ventana principal
        setTitle("Mi Aplicación Swing");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Creación de componentes
        JButton miBoton = new JButton("Haz clic");
        JLabel miEtiqueta = new JLabel("¡Bienvenido a Swing!");
        JTextField miCampoTexto = new JTextField("Ingrese texto");
        // Creación de un panel para organizar componentes
        JPanel miPanel = new JPanel();
        miPanel.add(miBoton);
        miPanel.add(miEtiqueta);
        miPanel.add(miCampoTexto);
        // Agregar el panel a la ventana
        add(miPanel);
        // Otros componentes y lógica aquí
    }
    public static void main(String[] args) {
```

```
        // Creación y visualización de la ventana
        MiAplicacion aplicacion = new MiAplicacion();
        aplicacion.setVisible(true);
    }
}
```

**Comparación y Contraste:**

**Swing vs. AWT:**

Swing ofrece componentes más avanzados y personalizables que AWT.

AWT (Abstract Window Toolkit) es una librería más antigua y simple.

**Componentes de Interfaz vs. Componentes Básicos:**

Los componentes de interfaz (botones, etiquetas, campos de texto) son visibles en la GUI.

Los componentes básicos (JFrame, JPanel) proporcionan estructura y organización.

**Resumen**:

Swing es una poderosa librería de interfaz gráfica en Java que permite la creación de aplicaciones con interfaces de usuario intuitivas. Al proporcionar componentes flexibles y avanzados, Swing facilita el desarrollo de aplicaciones visualmente atractivas y funcionales en entornos Java.

**Objeto JFrame y JOptionPanel**

**JFrame:**

El objeto JFrame es una clase en la biblioteca Swing de Java que representa una ventana gráfica. Se utiliza como el marco principal para aplicaciones GUI en Java. Aquí hay una introducción y un ejemplo básico:

**Definición y Creación:**

```java
import javax.swing.JFrame;

public class MiVentana extends JFrame {
    public MiVentana() {
        // Configuración de la ventana
        setTitle("Mi Aplicación");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Otros componentes y lógica aquí
    }

    public static void main(String[] args) {
        // Creación y visualización de la ventana
        MiVentana ventana = new MiVentana();
        ventana.setVisible(true);
    }
}
```

**Configuraciones Comunes:**

setTitle(String title): Establece el título de la ventana.

setSize(int width, int height): Define el tamaño de la ventana.

setDefaultCloseOperation(int operation): Configura la operación de cierre al hacer clic en el botón de cierre.

**JOptionPane:**

JOptionPane es una clase que proporciona un conjunto de métodos estáticos para crear y mostrar ventanas de diálogo modales (cuadros de diálogo) que contienen un mensaje o solicitan entrada del usuario. Se utiliza para interactuar con el usuario de manera eficiente.

**Mensajes Simples:**

import javax.swing.JOptionPane;

public class MiVentana {

    public static void main(String[] args) {

```
        // Mensaje de información
        JOptionPane.showMessageDialog(null, "¡Hola, Mundo!", "Saludo",
JOptionPane.INFORMATION_MESSAGE);

        // Mensaje de advertencia
        JOptionPane.showMessageDialog(null, "Advertencia: Algo ocurrió", "Alerta",
JOptionPane.WARNING_MESSAGE);

        // Mensaje de error
        JOptionPane.showMessageDialog(null, "Error: Operación fallida", "Error",
JOptionPane.ERROR_MESSAGE);
    }
}
```

**Solicitar Entrada:**

```
import javax.swing.JOptionPane;

public class MiVentana {
    public static void main(String[] args) {
        // Solicitar entrada de texto
        String nombre = JOptionPane.showInputDialog("Ingrese su nombre:");
        System.out.println("Nombre ingresado: " + nombre);

        // Solicitar confirmación
```

```
        int respuesta = JOptionPane.showConfirmDialog(null, "¿Está seguro?",
"Confirmación", JOptionPane.YES_NO_OPTION);

        if (respuesta == JOptionPane.YES_OPTION) {

            System.out.println("Confirmado");

        } else {

            System.out.println("Cancelado");

        }

    }

}
```

**Comparación y Contraste:**

**JFrame vs. JOptionPane:**

JFrame se utiliza para crear la ventana principal de la aplicación.

JOptionPane se utiliza para mostrar ventanas de diálogo con mensajes o solicitudes de entrada.

**Ventanas de Diálogo vs. Ventana Principal:**

Las ventanas de diálogo son modales y requieren interacción del usuario antes de continuar.

La ventana principal (JFrame) es la ventana principal de la aplicación.

**Resumen**:

JFrame se utiliza para crear la ventana principal de una aplicación Swing, mientras que JOptionPane proporciona una forma conveniente de mostrar mensajes y solicitar entrada del usuario. Estos componentes son esenciales para construir interfaces de usuario interactivas y amigables en aplicaciones Java.

**JPanel, JLabel, JTextField**

**JPanel:**

JPanel es una clase en la biblioteca Swing de Java que se utiliza para organizar y agrupar componentes dentro de una interfaz gráfica de usuario (GUI). Funciona como un contenedor flexible y se puede incorporar en un JFrame u otros contenedores.

**Creación y Configuración Básica:**

```
import javax.swing.JPanel;

public class MiPanel extends JPanel {
    public MiPanel() {
        // Configuración del panel
        // Otros componentes y lógica aquí
    }
}
```

Uso en un JFrame:

```
import javax.swing.JFrame;

public class MiVentana extends JFrame {
    public MiVentana() {
        // Configuración de la ventana
        setTitle("Ventana con Panel");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Creación y adición de un panel
        MiPanel miPanel = new MiPanel();
        add(miPanel);

        // Otros componentes y lógica aquí
    }
}
```

**JLabel:**

JLabel es un componente Swing utilizado para mostrar texto o imágenes en una interfaz gráfica. Proporciona una forma sencilla de presentar información estática.

**Creación y Configuración:**

```
import javax.swing.JLabel;

public class MiEtiqueta extends JLabel {
    public MiEtiqueta() {
        // Configuración de la etiqueta
        setText("¡Hola, Mundo!");
        // Otros ajustes y lógica aquí
    }
}
```

Uso en un JPanel:

```
import javax.swing.JPanel;
import javax.swing.JLabel;

public class MiPanel extends JPanel {
    public MiPanel() {
        // Creación y adición de una etiqueta al panel
        MiEtiqueta miEtiqueta = new MiEtiqueta();
        add(miEtiqueta);

        // Otros componentes y lógica aquí
    }
}
```

**JTextField:**

JTextField es un componente Swing que permite al usuario ingresar texto. Se utiliza para recopilar información de entrada del usuario.

**Creación y Configuración Básica:**

```
import javax.swing.JTextField;

public class MiCampoTexto extends JTextField {
    public MiCampoTexto() {
        // Configuración del campo de texto
        setColumns(20);
        // Otros ajustes y lógica aquí
    }
}
```

Uso en un JPanel:

```
import javax.swing.JPanel;
import javax.swing.JTextField;

public class MiPanel extends JPanel {
    public MiPanel() {
        // Creación y adición de un campo de texto al panel
        MiCampoTexto miCampoTexto = new MiCampoTexto();
        add(miCampoTexto);

        // Otros componentes y lógica aquí
    }
}
```

**Comparación y Contraste:**

**JPanel vs. JLabel vs. JTextField:**

JPanel es un contenedor general utilizado para organizar componentes.

JLabel se usa para mostrar texto o imágenes.

JTextField es un campo de texto interactivo para la entrada del usuario.

**Uso en Jerarquía de Componentes**:

JPanel y JLabel se pueden agregar a un JFrame u otros contenedores.

JTextField generalmente se utiliza en combinación con otros componentes dentro de un contenedor.

**Resumen**:

JPanel, JLabel y JTextField son componentes fundamentales en la construcción de interfaces de usuario en Java Swing. Mientras JPanel sirve como contenedor para organizar componentes, JLabel presenta información estática y JTextField permite la entrada de texto por parte del usuario. Su combinación proporciona flexibilidad y versatilidad en el diseño de aplicaciones GUI.

## ACTIVIDADES PRÁCTICAS

**Tabla**

Detalle de la Práctica

| Practicas | Instrucciones | Duración de la Practica | Mecanismo de Evaluación |
|---|---|---|---|
| **Foro de debate** | Revisar Actividades Prácticas | 0:30 mm | Por rúbricas |
| **Práctica investiga: Taller/Ensayo/Investigación** | Revisar Actividades Prácticas | 3:30 h | Por rúbricas |
| **Tutorías Prácticas** | Revisar Actividades Prácticas | 2:00 mm | Por participación en clase |
| **Test evaluativo** | Revisar Actividades Prácticas | 0:30 mm | Formativa |

*Nota.* Elaboración propia

**Consideraciones Generales**

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

**FORO DE DEBATE**

**Tema:** Describe tu opinión sobre el uso y aplicación de la librería de Swing.

**Instrucciones para Participar:**

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

**PRÁCTICA INVESTIGATIVA**

**Taller:** Ejercicios con librería Swing.

Desarrollar la lógica y adaptarse a la sintaxis de las estructuras de control del lenguaje de programación de JAVA, mediante la resolución de casos de estudio.

**Consigna:**

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

**PRÁCTICA TEST**

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

**Consideraciones para realizar el test**

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

**Tema 2: Objetos de listas en Java**

En Java, las listas son estructuras de datos que permiten almacenar y manipular colecciones de elementos. Una de las implementaciones más comunes es la interfaz List, que pertenece al framework de colecciones en Java. Esta interfaz define un conjunto de operaciones que se pueden realizar en listas, como agregar elementos, eliminar elementos, y acceder a elementos por posición.

**Ejemplo de Uso de Listas:**

```java
import java.util.ArrayList;
import java.util.List;

public class EjemploListas {
    public static void main(String[] args) {
        // Creación de una lista de cadenas
        List<String> listaCadenas = new ArrayList<>();

        // Agregar elementos a la lista
        listaCadenas.add("Elemento 1");
        listaCadenas.add("Elemento 2");
        listaCadenas.add("Elemento 3");

        // Acceder a elementos por posición
        String elemento2 = listaCadenas.get(1);
        System.out.println("Elemento en la posición 2: " + elemento2);

        // Imprimir todos los elementos de la lista
        System.out.println("Elementos en la lista:");
        for (String elemento : listaCadenas) {
            System.out.println(elemento);
        }
    }
}
```

**Métodos Comunes de Listas**

**add(E element):** Agrega un elemento al final de la lista.

**get(int index):** Devuelve el elemento en la posición especificada.

**remove(int index):** Elimina el elemento en la posición especificada.

**size():** Devuelve el número de elementos en la lista.

**contains(Object o):** Devuelve true si la lista contiene el elemento especificado.

**Comparación con Otros Tipos de Listas**

**ArrayList vs. LinkedList**

ArrayList es más eficiente para el acceso aleatorio y la lectura.

LinkedList es más eficiente para la inserción y eliminación en el medio de la lista.

**List vs. Set vs. Map**

List permite duplicados y mantiene el orden.

Set no permite duplicados y no garantiza un orden específico.

Map es una colección de pares clave-valor.

Las listas en Java, implementadas a través de la interfaz List, ofrecen una forma flexible y poderosa de gestionar colecciones de elementos. Pueden almacenar cualquier tipo de objeto, permiten el acceso por posición y ofrecen una variedad de métodos para manipular y consultar la información. La elección entre ArrayList y LinkedList depende de los requisitos específicos de la aplicación.

## JButton, JList, JComboBox

### JButton

JButton es un componente Swing utilizado para crear botones en una interfaz gráfica de usuario (GUI). Los botones pueden contener texto, imágenes o ambas. Su propósito principal es capturar eventos de acción, como clics, para ejecutar acciones específicas en la aplicación.

**Ejemplo de Uso:**

```java
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class EjemploBoton {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Ejemplo JButton");
        JButton boton = new JButton("Haz clic");

        // Agregar un ActionListener para capturar eventos de clic
        boton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("Botón clickeado");
            }
        });

        frame.getContentPane().add(boton);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

### JList

JList es un componente Swing que muestra una lista de elementos seleccionables. Puede contener elementos simples o complejos y permite a los usuarios seleccionar uno o varios elementos de la lista.

### Ejemplo de Uso:

```java
import javax.swing.DefaultListModel;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;

public class EjemploLista {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Ejemplo JList");

        // Crear una lista de elementos
        String[] elementos = {"Elemento 1", "Elemento 2", "Elemento 3", "Elemento 4"};
        JList<String> lista = new JList<>(elementos);

        // Permitir selección simple o múltiple
        lista.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);

        // Agregar la lista a un JScrollPane para permitir desplazamiento
        JScrollPane scrollPane = new JScrollPane(lista);
        frame.getContentPane().add(scrollPane);

        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

**JComboBox**

JComboBox es un componente Swing que proporciona una lista desplegable de elementos. Los usuarios pueden seleccionar un elemento de la lista desplegable.

**Ejemplo de Uso:**

```
import javax.swing.DefaultComboBoxModel;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class EjemploComboBox {

    public static void main(String[] args) {

        JFrame frame = new JFrame("Ejemplo JComboBox");

        // Crear una lista de elementos

        String[] elementos = {"Elemento 1", "Elemento 2", "Elemento 3", "Elemento 4"};

        JComboBox<String> comboBox = new JComboBox<>(elementos);


        // Agregar un ActionListener para capturar eventos de selección

        comboBox.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                System.out.println("Elemento seleccionado: " +
comboBox.getSelectedItem());

            }
```

```
        });

        frame.getContentPane().add(comboBox);

        frame.setSize(300, 200);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setVisible(true);

    }

}
```

**Comparación y Contraste**

**JButton vs. JList vs. JComboBox**

JButton se utiliza para acciones específicas, como clics.

JList muestra una lista de elementos seleccionables.

JComboBox proporciona una lista desplegable de elementos.

**Acciones y Eventos**

JButton captura eventos de clic.

JList y JComboBox capturan eventos de selección.

JButton, JList y JComboBox son componentes esenciales en el desarrollo de interfaces de usuario en Java Swing. Los botones (JButton) son utilizados para acciones específicas, mientras que las listas (JList) y las listas desplegables (JComboBox) son ideales para presentar y seleccionar elementos de una lista. Estos componentes permiten una interacción efectiva y amigable con el usuario en aplicaciones Java.

**JMenu, JMenuBar, JScrollBar**

**JMenu y JMenuBar**

JMenu y JMenuBar son componentes Swing utilizados para crear menús en una interfaz gráfica de usuario (GUI). JMenuBar actúa como un contenedor para JMenu y otros componentes relacionados con menús.

**Ejemplo de Uso:**

```
import javax.swing.JFrame;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

public class EjemploMenu {

    public static void main(String[] args) {

        JFrame frame = new JFrame("Ejemplo JMenu y JMenuBar");

        // Crear una barra de menú

        JMenuBar menuBar = new JMenuBar();

        // Crear menús

        JMenu archivoMenu = new JMenu("Archivo");

        JMenu editarMenu = new JMenu("Editar");

        // Crear elementos de menú

        JMenuItem abrirItem = new JMenuItem("Abrir");

        JMenuItem guardarItem = new JMenuItem("Guardar");

        JMenuItem cortarItem = new JMenuItem("Cortar");

        JMenuItem copiarItem = new JMenuItem("Copiar");
```

```
        JMenuItem pegarItem = new JMenuItem("Pegar");
        // Agregar elementos de menú a los menús
        archivoMenu.add(abrirItem);
        archivoMenu.add(guardarItem);
        editarMenu.add(cortarItem);
        editarMenu.add(copiarItem);
        editarMenu.add(pegarItem);
        // Agregar menús a la barra de menú
        menuBar.add(archivoMenu);
        menuBar.add(editarMenu);
        // Establecer la barra de menú en el marco
        frame.setJMenuBar(menuBar);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

**JScrollBar**

JScrollBar es un componente Swing que proporciona una barra de desplazamiento que permite a los usuarios ajustar valores numéricos.

**Ejemplo de Uso:**

```
import javax.swing.JFrame;
import javax.swing.JPanel;
```

```
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;

public class EjemploScrollBar {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Ejemplo JScrollBar");

        // Crear una barra de desplazamiento vertical
        JScrollBar scrollBar = new JScrollBar(JScrollBar.VERTICAL, 30, 20, 0, 100);

        // Crear un panel con la barra de desplazamiento
        JPanel panel = new JPanel();
        panel.add(scrollBar);

        // Agregar el panel a un JScrollPane para permitir desplazamiento
        JScrollPane scrollPane = new JScrollPane(panel);

        // Establecer el JScrollPane en el marco
        frame.getContentPane().add(scrollPane);

        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
```

```
    }
}
```

**Comparación y Contraste**

**JMenu vs. JMenuBar**

JMenuBar es la barra que contiene uno o más menús (JMenu).

JMenu representa un menú desplegable que puede contener elementos de menú (JMenuItem) y submenús (JMenu).

**JScrollBar vs. JSlider**

JScrollBar proporciona una barra de desplazamiento que permite ajustar valores numéricos.

JSlider es similar, pero más adecuado cuando se desea una representación visual continua de valores y no solo un ajuste preciso.

JMenu y JMenuBar son componentes esenciales para la creación de menús en aplicaciones Java Swing, facilitando la organización y accesibilidad de funciones. JScrollBar proporciona una forma eficiente para que los usuarios ajusten valores numéricos. Estos componentes son esenciales para mejorar la usabilidad y la experiencia del usuario en aplicaciones GUI.

## ACTIVIDADES PRÁCTICAS

**Tabla**

Detalle de la Práctica

| Practicas | Instrucciones | Duración de la Practica | Mecanismo de Evaluación |
|---|---|---|---|
| **Foro de debate** | Revisar Actividades Prácticas | 0:30 mm | Por rúbricas |
| **Práctica investiga: Taller/Ensayo/Investigación** | Revisar Actividades Prácticas | 3:30 h | Por rúbricas |
| **Tutorías Prácticas** | Revisar Actividades Prácticas | 2:00 mm | Por participación en clase |
| **Test evaluativo** | Revisar Actividades Prácticas | 0:30 mm | Formativa |

*Nota.* Elaboración propia

**Consideraciones Generales**

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

**FORO DE DEBATE**

**Tema:** Aplicación de listas y menú en Programación Orientada a Objetos en JAVA.

**Instrucciones para Participar:**

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

**PRÁCTICA INVESTIGATIVA**

**Taller:** Creación de formulario básico de datos.

Desarrollar la lógica y adaptarse a la sintaxis de las estructuras de control del lenguaje de programación de JAVA, mediante la resolución de casos de estudio.

**Consigna:**

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

**PRÁCTICA TEST**

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

**Consideraciones para realizar el test**

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.