INSTITUTO SUPERIOR TECNOLÓGICO QUITO

Formamos tu PROPÓSITO DE VIDA

Guía general de FUNDAMENTOS DE PROGRAMACIÓN

INSTITUTO SUPERIOR TECNOLÓGICO QUITO - DESDE 1984 - SABER HACER Bien

INSTITUTO SUPERIOR TECNOLÓGICO QUITO
Excelencia en Educación Superior

GUÍA GENERAL DE FUNDAMENTOS DE PROGRAMACIÓN

AUTOR: MÓNICA ALEXANDRA RAMÍREZ VELASTEGUI

EDICIÓN: PRIMERA EDICIÓN

AÑO: 2024

TRABAJO EN EDICIÓN:

DOXA EDITION

DIRECCIÓN EDITORIAL: DIEGO JAVIER BASTIDAS LOGROÑO

EDITOR EXTERNO : DAVID FABIAN CEVALLOS SALAS

ISBN:

SOBRE EL AUTOR

Mónica Ramírez es una profesional con más de 10 años de experiencia en el apasionante mundo de las Tecnologías de la Información (TI). A lo largo de su carrera, se ha destacado por liderar proyectos innovadores y fomentar una comunicación efectiva en equipos multidisciplinarios. Su pasión por la educación la llevó a obtener una Maestría en Gerencia y Liderazgo Educativo, y actualmente se encuentra cursando una Maestría en Inteligencia Artificial Aplicada, lo que refleja su compromiso con el aprendizaje continuo.

Con más de 8 años de experiencia como docente universitaria, Mónica ha dejado una huella significativa en el desarrollo académico y personal de sus estudiantes. Su enfoque pedagógico busca formar profesionales íntegros, combinando conocimientos técnicos con habilidades interpersonales esenciales para el ámbito laboral.

La sólida trayectoria de Mónica en TI, junto con su interés en áreas emergentes como la Inteligencia Artificial, le permite ofrecer una educación actualizada y relevante. Ella cree firmemente que la educación tiene el poder de transformar vidas, preparándolos para enfrentar los desafíos del futuro con confianza, conocimiento y creatividad.

Su carrera comenzó como Ingeniera de Software, donde adquirió habilidades técnicas esenciales. Luego, se desempeñó como Analista de Software, Administradora de Base de Datos y Analista de Ciencia de Datos, aplicando técnicas de análisis para extraer información valiosa que respalda la toma de decisiones estratégicas. Su capacidad de liderazgo la llevó a convertirse en Gerente de Desarrollo de Software, donde impulsó proyectos innovadores y mejoró la eficiencia de los equipos. Además, ha realizado importantes contribuciones en el ámbito educativo, trabajando como Analista Técnica en evaluación educativa a nivel nacional y Docente Investigador.

Gracias a su experiencia en el sector público y privado, así como a sus valiosas contribuciones a la educación y su liderazgo académico, Mónica se ha consolidado como una referente en el ámbito de las Tecnologías de la Información y la Educación. Su compromiso con la excelencia y su anhelo de influir positivamente en las futuras generaciones de profesionales en tecnología son palpables en su trayectoria y en los logros que ha conseguido a lo largo de su carrera

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

ÍNDICE

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

ÍNDICE DE FIGURAS

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

ÍNDICE DE TABLAS

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

GUÍA GENERAL DE FUNDAMENTOS DE PROGRAMACIÓN

1. DESCRIPCIÓN DE LA ASIGNATURA

La asignatura de Fundamentos de Programación es fundamental para los estudiantes de la carrera de Desarrollo de Software. Con una duración de 3 créditos, su objetivo principal es introducir a los alumnos en el emocionante mundo de la programación. A lo largo del curso, los estudiantes aprenderán a comunicarse con las computadoras mediante la escritura de instrucciones que estas pueden ejecutar para realizar tareas específicas.

En esta asignatura los alumnos se familiarizarán con conceptos clave como variables y constantes, así como con operadores aritméticos, relacionales y lógicos. Estos elementos les permitirán realizar operaciones matemáticas, comparar datos y combinar condiciones. También se les enseñará a utilizar pseudocódigos, que son herramientas útiles para planificar y describir algoritmos de manera informal, facilitando así la comprensión y el diseño de soluciones antes de la codificación.

Al finalizar la asignatura, los estudiantes estarán preparados para diseñar e implementar soluciones básicas a problemas cotidianos. Habrán desarrollado habilidades en la resolución de problemas a través de la codificación y podrán aplicar los conceptos aprendidos en el lenguaje de programación Java. Mediante trabajos prácticos, estudios de caso y proyectos en entornos como PSeInt y NetBeans (versión 21), los alumnos tendrán la oportunidad de crear programas ejecutables que reflejen su creatividad y capacidad en el área de la programación.

2. BIBLIOGRAFÍA

2.1. Básica

- Ramírez, J. (2019). *Fundamentos iniciales de lógica de programación I. Algoritmos en PseInt y Python*: (1 ed.). D - Institución Universitaria de Envigado.

Este texto es un recurso esencial para entender los conceptos básicos de programación, organizado en tres unidades que cubren fundamentos como algoritmos, pseudocódigos, estructuras de decisión y la sintaxis básica de programación. También ofrece una guía

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

práctica para resolver ejercicios utilizando pseudocódigos y diagramas de flujo con la herramienta PseInt, facilitando así la comprensión de la programación. Está diseñado para ayudar a los lectores, especialmente a los principiantes, a progresar desde algoritmos simples hasta el desarrollo de aplicaciones más complejas en diversas plataformas, incluyendo aplicaciones móviles, sitios web, configuración de periféricos y creación de videojuegos.

- Vegas, J. (2020). *Java: curso práctico*: (1ed.).RA-MA Editorial.

Este texto es una guía completa para desarrollar aplicaciones profesionales en Java, comenzando desde la versión 8 del JDK. Presenta de manera práctica y didáctica los fundamentos de la programación y el diseño orientado a objetos, además de un enfoque moderno en estructuras de datos y diseño de algoritmos. También abarca temas avanzados como la programación funcional y concurrente, aprovechando las nuevas características de Java. Adicional ofrece acceso a códigos completo de un proyecto Maven a través de un sitio web específico, lo que ayuda en la experiencia de aprendizaje.

2.2. Complementaria

- Garrido, A. (2005). *Fundamentos de programación en C++:* (ed.). Delta Publicaciones.

El texto presenta muestra los conceptos fundamentales de programación en el contexto del lenguaje C++. A diferencia de otros textos, no se centra en detallar cada aspecto del lenguaje, sino en formar programadores. Cada tema se acompaña de ejemplos ilustrativos y ejercicios prácticos.

- Moreno, J. (2015). *Programación orientada a objetos:* (ed.). RA-MA Editorial.

Este libro está dirigida a los estudiantes que deseen aprender Programación con Lenguajes Orientados a Objetos y Bases de Datos Relacionales en concreto la programación orientada a objetos con Java.

- Zohonero, I. & Joyanes, L. (2008). *Estructuras de datos en Java:* (ed.). McGraw-Hill España.

Este texto se centra en la enseñanza de los conceptos fundamentales de análisis y diseño de algoritmos, junto con los principios de abstracción y las estructuras de datos en el

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

contexto del lenguaje de programación Java.

- HillPrieto, N. (2013). *Empezar a programar usando java:* (ed.). Editorial de la Universidad Politécnica de Valencia.

Este texto presenta una introducción al diseño metodológico de programas con un enfoque en la Programación Orientada a Objetos (POO) utilizando Java.

- Blasco, F. (2019). *Programación orientada a objetos en Java:* (ed.). Ediciones de la U.

Este texto introduce los fundamentos de la Programación Orientada a Objetos (POO), centrándose en conceptos clave como el encapsulamiento, la herencia y el polimorfismo, utilizando Java. Se incluyen ejemplos de aplicaciones Java.

3. COMPETENCIAS GENÉRICAS Y ESPECÍFICAS

Las competencias genéricas y específicas de la materia de Fundamentos de Programación en la carrera de desarrollo de software se resumen de la siguiente manera:

- Crear y desarrollar nuevos sistemas a través de la investigación de necesidades de los usuarios.
- Desarrollar aplicativos informáticos mediante el uso de plataformas actuales de programación.

Estas competencias permiten a los estudiantes no solo adquirir conocimientos técnicos en programación, sino también desarrollar habilidades críticas para trabajar de manera efectiva en equipos y adaptarse a las demandas cambiantes del sector tecnológico.

4. OBJETIVO GENERAL

Ofrecer una comprensión sólida de los principios básicos de la lógica de programación, abarcando temas como estructuras de datos, algoritmos y programación orientada a objetos. a través del análisis crítico y la práctica, los estudiantes aprenderán a diseñar, implementar y depurar programas básicos en un lenguaje de programación específico, lo que no solo mejora sus habilidades para analizar y resolver problemas, sino que también refuerza su perfil profesional, preparándolos para enfrentar y resolver desafíos reales en el campo del desarrollo de software.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

5. FORMACIÓN CIUDADANA, VALORES Y HABILIDADES BLANDAS

La asignatura cumple una función fundamental en el desarrollo integral de los alumnos al proporcionarles herramientas esenciales para su crecimiento personal y social, enfocándose en:

1. Educación ambiental: biodiversidad
2. Valores & habilidades blandas: amor: comunicación asertiva y escucha activa
3. Valores & habilidades blandas: compromiso social: adaptabilidad
4. Valores & habilidades blandas: cultura: creatividad
5. Valores & habilidades blandas: optimismo: planificación y gestión del tiempo
6. Valores & habilidades blandas: orgullo nacional: pensamiento crítico
7. Valores & habilidades blandas: solidaridad: trabajo en equipo
8. Valores & habilidades blandas: tolerancia: flexibilidad
9. Valores & habilidades blandas: verdad y honradez: proactividad

6. NORMAS DE CLASE

Es importante señalar que la evaluación de los componentes de gestión académica se compone de tres notas sumativas, cada una con una puntuación máxima de 6.60/6.60, así como un proyecto práctico, como evaluación formativa que se valora con 3.40/3.40, lo que da un total de 10/10 para la calificación del módulo. Los parciales se califican en una escala de hasta 6.60 puntos, representando cada uno el 2.22 de la calificación total de 6.6 puntos. Para presentarse al proyecto final, el estudiante debe haber obtenido al menos 4.50 puntos sumando las tres primeras notas. En caso de no alcanzar este mínimo en el proyecto, se otorga una oportunidad de recuperación dentro de las 48 horas laborables siguientes, según el calendario académico oficial. La nota mínima acumulada requerida para aprobar la asignatura es 7/10, y es esencial mantener al menos un 70% de asistencia a las clases. Los docentes deben informar a los estudiantes sobre sus notas individuales antes de registrarlas en el sistema, y se espera que los alumnos confirmen su aceptación y conformidad con estas calificaciones. Además, los docentes deben entregar un reporte de notas y asistencia a través del SGA y notificado a la coordinación de carrera y registrar las calificaciones en el sistema en un plazo máximo de 5 días posteriores a la recepción del proyecto final.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

7. SISTEMA DE EVALUACIÓN

- La asignatura está conformado por tres parciales, cada uno con una ponderación del 22% es decir, 66% más un 34% que proviene del proyecto final. Para proceder a realizar el proyecto, debes tener cómo requisito, una calificación mínima entre parciales del 45%.
- Para aprobar la asignatura se debe tener un promedio de 7/10 como mínimo.
- Para aprobar la asignatura debe tener un mínimo del 70% de asistencia.

8. UNIDADES

1. Fundamentos teóricos de programación
2. Elementos de los Algoritmos y Pseudocódigo
3. Programación Orientada a Objetos

Los logros obtenidos en su desarrollo personal y profesional dependen de cuánto esté dispuesto a invertir en esfuerzo y responsabilidad. Le invitamos a que, juntos, avancemos en el estudio de la asignatura. Tanto el equipo físico como el humano del TQ estamos aquí para acompañarlo en su proceso de enseñanza y aprendizaje.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

UNIDAD 1 (FUNDAMENTOS TEORICOS DE PROGRAMACIÓN)

"Estudia mientras otros están durmiendo; trabaja mientras otros están holgazaneando; prepárate mientras otros están jugando; y sueña mientras otros están deseando."

William Arthur Ward

Temas y Subtemas

- **Conceptos básicos de programación**
- **Lenguajes de programación**
- **Algoritmo**
- **Pseudocódigo**

Conceptos básicos a la programación

La programación es una de las bases más importantes en el mundo de la informática y la tecnología de la información. Conocer y saber programar es crucial si se desea desarrollar el software y las aplicaciones que se usan en nuestro día a día.

En términos simples, la programación consiste en desarrollar una serie de instrucciones precisas y detalladas que una computadora puede ejecutar para realizar tareas determinadas. Estas instrucciones se escriben en lenguajes de programación, que son como puentes que permiten a los programadores expresar sus ideas y hacer que la máquina las entienda. (Rodríguez, 2003)

En resumen, la programación no solo es una habilidad técnica, sino también una forma de comunicación con las computadoras que da vida a las herramientas digitales.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Los fundamentos de programación son esenciales para todos los programas y representan el primer paso que se debe aprender antes de elegir un lenguaje específico. Es crucial entender que un ordenador es una máquina eléctrica que solo comprende el código binario, es decir, 1 y 0, donde 1 indica corriente y 0 indica ausencia de corriente.

Lenguajes de programación

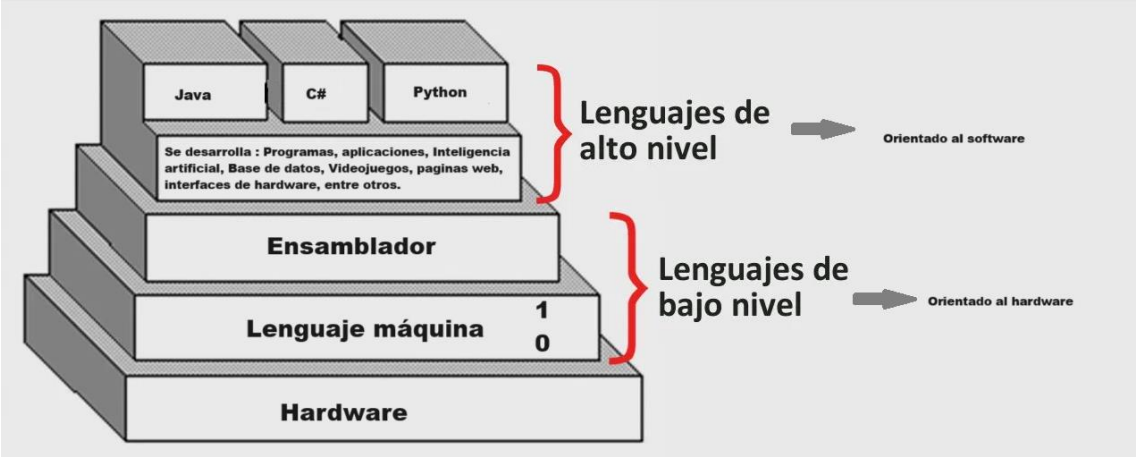
Un lenguaje de programación es un sistema formal de comunicación compuesto por un conjunto de reglas sintácticas y semánticas que permiten a los programadores escribir código fuente. Este código se traduce posteriormente en instrucciones que una computadora puede ejecutar directamente. Según Sebesta (2018), los lenguajes de programación son "vehículos para expresar algoritmos y estructuras de datos de manera que puedan ser entendidos y ejecutados por las máquinas".

Los lenguajes de programación se dividen en dos categorías (Tejera, 2020): de bajo nivel, que son más complejos y se asemejan al código del ordenador, y de alto nivel, que permiten interactuar con el ordenador en un lenguaje más comprensible, similar al del ser humano (como Java, Visual Basic, C++, JavaScript, Python, entre otros).

Algunos lenguajes de alto nivel son interpretados, lo que significa que el programa ejecuta las instrucciones directamente, mientras que otros requieren un compilador que traduce el código a un formato que el ordenador pueda entender.

Los lenguajes de programación facilitan la implementación de algoritmos y la resolución de problemas específicos en diversos contextos.

Ilustración 1:
Clasificación de los lenguajes de programación



Nota: En la ilustración se puede observar la clasificación de los lenguajes de programación según su utilidad y funcionalidad. Fuente: Ingeniería de Software. Generaciones del software. Recuperado de https://bit.ly/4fn1oHq

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Importancia y Utilidad de los Lenguajes de Programación

Los lenguajes de programación son fundamentales para la implementación de algoritmos y la resolución de problemas específicos en una amplia variedad de dominios, desde aplicaciones científicas hasta el desarrollo de videojuegos, o aplicaciones utilizando IA. Cada lenguaje tiene sus propias fortalezas y está optimizado para tareas específicas, lo que permite a los programadores elegir el más adecuado para sus requerimientos.

Algoritmos

Los algoritmos y pseudocódigos son elementos clave en el aprendizaje de la programación, ya que ofrecen las bases necesarias para resolver problemas mediante la creación de secuencias lógicas de pasos bien definidos (Navarrete, 2023). Los algoritmos actúan como el corazón de cualquier programa, permitiendo descomponer tareas complejas en acciones más simples y manejables. Por otro lado, los pseudocódigos facilitan la planificación y comprensión de estos algoritmos, utilizando una notación que se asemejaba al lenguaje natural, pero estructurada de manera que se puede traducir fácilmente a un lenguaje de programación. En esta unidad introductoria, se introduce a los estudiantes para establecer las bases teóricas y conceptuales de la programación.

Un algoritmo fue definido como una secuencia finita de pasos bien definidos que, al ser ejecutados, llevan a la resolución de un problema específico. Los algoritmos fueron la base de la programación y estuvieron presentes en casi todas las tareas computacionales. La precisión y la claridad en la definición de un algoritmo fueron cruciales para garantizar que se obtenga el resultado deseado.

Se puede decir que un algoritmo es como una receta que consiste en una serie de pasos ordenados, conocidos como instrucciones. Estas instrucciones sirven de guía para llevar a cabo una tarea específica, con el objetivo de lograr un resultado deseado o encontrar la solución a un problema. En esencia, un algoritmo ayuda a transformar una serie de acciones en un resultado claro y concreto.

Pasos para Diseñar Algoritmos

El diseño de algoritmos efectivos es fundamental en el mundo de la programación y en la resolución de problemas computacionales ya que es el conjunto de pasos que se debe seguir secuencialmente para poder resolver el problema planteado. Un buen algoritmo no solo se encarga de resolver el problema para el cual fue creado, sino que lo hace de manera eficiente, optimizando el uso del tiempo y los recursos disponibles.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

A continuación, se presentan varias estrategias clave para diseñar algoritmos:

- **Comprensión del Problema**: Antes de diseñar un algoritmo, es crucial comprender a fondo el problema a resolver. Esto incluye: realizar un análisis del problema, identificar las entradas y salidas y clarificar los requisitos.
- **Dividir y Conquistar**: La técnica de dividir y conquistar es una estrategia poderosa para el diseño de algoritmos. Implica dividir el problema en subproblemas más pequeños, resolver cada uno de forma independiente y luego integrar las soluciones para obtener el resultado final.ppseudoc

Este enfoque de resolución de problemas que implica descomponer un problema complejo en partes más manejables, identificar patrones, abstraer soluciones generales, y definir los pasos que puedan ser implementados por una computadora. A continuación, se describen los puntos fundamentales que todo programador debe seguir para resolver un problema usando el pensamiento computacional:

1. Descomposición

Consiste en dividir un problema grande y complejo en partes más pequeñas y manejables. Esto permite abordar cada subproblema por separado, facilitando su resolución. Un ejemplo si se quiere crear un videojuego, se descompone en subproblemas como el diseño de personajes, las reglas del juego, la creación de niveles, etc.

2. Reconocimiento de Patrones

Implica identificar similitudes o patrones dentro de los problemas o datos. Reconocer patrones ayuda a simplificar problemas porque se pueden usar soluciones previas para resolver problemas similares. Un ejemplo en un conjunto de datos de ventas, se pueden identificar patrones como el aumento de ventas durante ciertas épocas del año, lo que puede ayudar en la planificación futura.

3. Abstracción

Consiste en reducir la complejidad al enfocarse solo en los detalles relevantes y eliminar aquellos que no son esenciales para la solución del problema. Esto permite crear una representación general del problema. Un ejemplo en la creación de una aplicación de mapas, no se necesitan detalles como el color exacto de cada edificio; en cambio, se abstrae la información a puntos de interés y rutas.

4. Definir los pasos a seguir

Es el desarrollo de un conjunto de instrucciones paso a paso para resolver un problema o realizar una tarea. Estos pasos deben ser claros, precisos, y ejecutables.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

5. Prueba y Refinamiento

Implica evaluar y realizar las pruebas de la solución y los pasos para asegurarse de que resuelve el problema de manera eficiente y efectiva. Si se encuentra algún error o ineficiencia, se debe reordenar los pasos o ajustar y mejorar este flujo que se define. Un ejemplo una vez desarrollado un algoritmo de búsqueda, se puede evaluar su rendimiento en diferentes escenarios y ajustar su lógica para mejorar la velocidad o el uso de recursos.

Pseudocódigos

El pseudocódigo como su nombre lo indica, es un código falso que se asemeja a un lenguaje de programación, pero mucho más flexible, y fácil de entender, y obviamente no puede ser interpretado directamente por la máquina. (Del Prado, 2014)

Se puede decir que el pseudocódigo es una representación intermedia entre el lenguaje natural y el lenguaje de programación. Se utiliza para diseñar y visualizar algoritmos de manera más comprensible antes de implementarlos en un lenguaje de programación específico. El pseudocódigo ayuda a los programadores a planificar y estructurar sus soluciones sin preocuparse por la sintaxis específica del lenguaje de programación.

Características de los Pseudocódigos

Los pseudocódigos al ser lo suficientemente estructurado como para reflejar la lógica y el flujo de un algoritmo, pero sin la necesidad de adherirse a las reglas estrictas de sintaxis de un lenguaje de programación específico, tiene las siguientes características:

- **Fácil de Leer y Escribir:** El pseudocódigo utiliza una sintaxis que es fácil de entender para las personas, utilizando palabras y frases del lenguaje natural.
- **Independiente del Lenguaje de Programación:** No depende de ningún lenguaje de programación específico, lo que lo hace útil para cualquier programador, independientemente del lenguaje que utilice.
- **Estructura Similar a los Algoritmos Formales:** A pesar de su flexibilidad, sigue una estructura que refleja los componentes de un algoritmo: entrada, procesamiento y salida.
- **Facilita la Comprensión y Comunicación:** Ayuda a los programadores a conceptualizar y comunicar ideas sobre cómo resolver un problema antes de codificarlo.
- **Soporta la Modificación y Refinamiento:** Permite realizar cambios y mejoras en la lógica del algoritmo de manera rápida sin preocuparse por los errores de sintaxis.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- **No puede ser interpretado directamente por una máquina,** ya que está diseñado en un lenguaje natural entendible por los humanos.

Estructura Básica del Pseudocódigo

El pseudocódigo se organiza de manera similar a un programa real, con declaraciones de inicio y fin, así como instrucciones que definen entradas, procesos, y salidas. A continuación, se muestran algunos elementos comunes de pseudocódigo:

- Entrada: Especifica qué datos necesita el algoritmo para funcionar yn ejemplo podría ser: Leer número
- Procesamiento: Describe los pasos necesarios para procesar los datos de entrada., por ejemplo operaciones como

  suma = número1 + número2
- Salida: Especifica qué información generada debe ser presentada al usuario, por ejemplo

  Escribir El resultado es: suma.
- Condicionales: Indican decisiones que se realizan para el procesamiento de datos dentro del algoritmo, por ejemplo usar una condición:

  Si x > 0 Entonces Escribir "Positivo"

  Sino Escribir "Negativo"
- Bucles: Permiten repetir bloques de código hasta cumplir ciertas condiciones, por ejemplo:

  Mientras i <= 10 Hacer….

Relación entre Lenguajes de Programación, Algoritmos y Pseudocódigos

La relación entre los lenguajes de programación, los algoritmos y los pseudocódigos es tanto bidireccional como complementaria.

Cuando los programadores abordan un problema, a menudo comienzan creando un algoritmo en pseudocódigo. Esta etapa les permite enfocarse en la lógica del problema sin distraerse con los detalles específicos de la sintaxis de un lenguaje de programación. El pseudocódigo actúa como un borrador que facilita la visualización y planificación del algoritmo.

Una vez que el algoritmo está claramente definido en pseudocódigo, el siguiente paso es traducirlo a un lenguaje de programación. En esta fase, los programadores deben tener en cuenta las características y capacidades del lenguaje elegido para asegurarse de que el algoritmo se implemente de manera eficiente y correcta. Así, el pseudocódigo y los lenguajes de

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

programación trabajan juntos, permitiendo a los desarrolladores crear soluciones efectivas y comprensibles.

Para temas de estudio de esta materia se trabaja con PSeInt que es una herramienta diseñada para ayudar a los estudiantes a aprender los conceptos básicos de la programación sin la complejidad de la sintaxis estricta de los lenguajes de programación tradicionales. Su principal objetivo es permitir a los usuarios concentrarse en la lógica de los algoritmos, utilizando una notación más cercana al lenguaje natural. Según García (2016), PSeInt es una aplicación educativa cuyo objetivo es brindar una herramienta que permita el aprendizaje de la programación a través del uso de pseudocódigos.

A continuación, se proporciona el enlace para descargar PSeInt, disponible para los sistemas operativos GNU/Linux, Windows y MacOS: Descargar PSeInt.

Por ejemplo, si se desea hacer un algoritmo en pseudocódigo utilizando la herramienta PSeInt para calcular el área de un triángulo, sería como se muestra en la siguiente ilustración.

Ilustración 2:
Algoritmo en pseudocódigo usando PSeInt

```
1  Algoritmo CalcularAreaTriangulo
2      Definir base, altura, area Como Real
3      Escribir "Ingrese la base del triángulo: "
4      Leer base
5      Escribir "Ingrese la altura del triángulo: "
6      Leer altura
7      area = (base * altura) / 2
8      Escribir "El área del triángulo es: ", area
9  FinAlgoritmo
10
```

Nota: En la ilustración se puede observar el algoritmo que permite realizar el cálculo del área de un triángulo.

Fuente: Elaboración propia

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Autoevaluación 1

Lea detenidamente la pregunta y marque la respuesta correcta entre las opciones dadas.

1. **¿Qué es un algoritmo?**

   a) Una serie de instrucciones específicas para resolver un problema.

   b) Un lenguaje de programación que permite resolver un problema.

   c) Una pieza de hardware del computador.

   d) Un tipo de software ejecutable en el computador.

2. **¿Cuál es el principal objetivo de PSeInt?**

   a) Ejecutar código de alto rendimiento.

   b) Compilar código en diferentes lenguajes de programación.

   c) Diseñar interfaces gráficas que resuelvan un problema en concreto.

   d) Facilitar el aprendizaje de la programación utilizando pseudocódigo.

3. **¿ Qué es el pseudocódigo?**

   a) Un lenguaje de programación interpretado.

   b) Un lenguaje de programación compilado.

   c) Un código false que representa el lenguaje natural y el lenguaje de programación.

   d) Un conjunto de instrucciones ejecutadas directamente por la computadora.

4. **¿Cuál de los siguientes es un lenguaje de programación de alto nivel?**

   a) Assembly Language

   b) Machine Language.

   c) Binario.

   d) Java.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

5. **¿Qué herramienta se utiliza para aprender programación utilizando pseudocódigo?**

   a) Java

   b) PSeInt

   c) Python

   d) C#

6. **¿Útil para diseñar y visualizar algoritmos antes de implementarlos en un lenguaje de programación?**

   a) Compilador

   b) Debugger

   c) Pseudocódigo

   d) Interprete

7. **¿Cuál es el objetivo principal de un algoritmo?**

   a) Proporcionar una solución clara y eficiente a un problema específico.

   b) Convertir el código fuente en lenguaje máquina.

   c) Diseñar interfaces de usuario atractivas.

   d) Mejorar el diseño de interfaces del software.

8. **¿Qué es la programación?**

   a) La creación de hardware para que la computadora funcione exitosamente.

   b) El diseño gráfico de interfaces de usuario.

   c) El conjunto de instrucciones que una computadora puede ejecutar.

   d) La organización de archivos en una computadora.

9. **¿Los lenguajes de programación como Java, Visual Basic, C++ y Python son considerados lenguajes de bajo nivel?**

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

a) Verdadero

b) Falso

1. **PSeInt es una herramienta que facilita el aprendizaje de la programación utilizando pseudocódigo, eliminando la complejidad de la sintaxis de los lenguajes de programación tradicionales.**

a) Verdadero

b) Falso

Si ha concluido la autoevaluación:

Verifique sus respuestas en el solucionario que se encuentra al final de la presente Guía.

Si alcanzó un porcentaje alto de respuestas correctas en las preguntas de evaluación, es hora de continuar con la Unidad 2, caso contrario, vuelva a revisar los temas en los que ha tenido dificultad.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Resumen de la Unidad 1

La Unidad 1 trata sobre los fundamentos teóricos de la programación, abordando conceptos esenciales como algoritmos y pseudocódigos, que son vitales para resolver problemas a través de secuencias lógicas. Los algoritmos son fundamentales, ya que permiten descomponer tareas complejas en pasos manejables, mientras que los pseudocódigos ayudan a planificar estos algoritmos utilizando una notación similar al lenguaje natural. La programación implica crear instrucciones claras para que una computadora realice tareas específicas, utilizando lenguajes como Java, Visual Basic, C++ y Python. Estos lenguajes se dividen en dos categorías: de bajo nivel, que se asemejan al código máquina y están orientados al hardware, y de alto nivel, que son más comprensibles para los humanos y se enfocan en el software. Además, se destaca la relación complementaria entre lenguajes de programación, algoritmos y pseudocódigos, mencionando a PSeInt como una herramienta clave para aprender estos conceptos sin la complejidad de la sintaxis estricta de los lenguajes de programación.

¡Siga adelante! ¡Muchos éxitos!

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

UNIDAD 2 (ELEMENTOS DE UN ALGORITMO)

"La diferencia entre aquellos que tuvieron éxito y los que no lo tuvieron, no fue la falta de fuerza o de inteligencia; fue la falta de voluntad"

Vicent Lombardi.

Temas y Subtemas

- **Estructura de algoritmos**
- **Variables y constantes**
- **Operadores**
- **Estrutura de control**

La programación se ha convertido en una habilidad fundamental para el desarrollo de soluciones innovadoras y eficientes. En esta unidad se verá una base sólida en los fundamentos de la programación, cubriendo conceptos esenciales que son aplicables tanto en el entorno de PSeInt como en el lenguaje de programación Java que se verá en esta asignatura. Se explorará la estructura de un algoritmo, enfatizando su importancia en la resolución de problemas y en la creación de software funcional. Además, se analizará cómo se representan los algoritmos en PSeInt, y de esta manera se facilita la comprensión de la lógica de programación, y se abordará los conceptos básicos de Java, un lenguaje ampliamente utilizado en la industria.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Estructura de un algoritmo

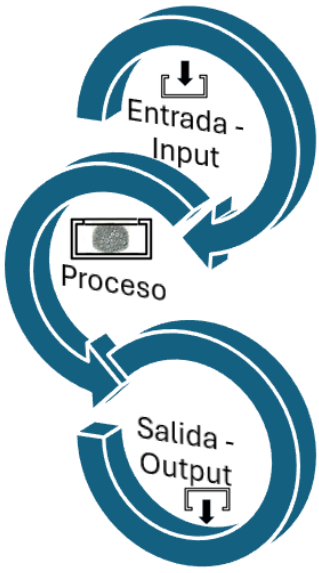
Como ya se ha visto un algoritmo es un conjunto de instrucciones definidas y ordenadas que permiten resolver un problema o realizar una tarea.

En un algoritmo (y en un programa) se puede identificar tres acciones clave:

- **Entrada**: La información inicial que el algoritmo necesita para comenzar.
- **Proceso**: Las operaciones que se llevan a cabo sobre esa información.
- **Salida**: Los resultados que obtenemos al final del proceso.

Estas acciones son fundamentales para entender cómo funcionan los algoritmos y su papel en la programación.

Ilustración 3:
Acciones de un algoritmo



Nota: En la ilustración se puede observar las tres acciones fundamentales de un algoritmo: Entrada, Proceso y Salida. Fuente: Elaboración propia

Elementos de un algoritmo

Para diseñar algoritmos efectivos, se debe considerar varios elementos clave:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Estructura de datos

Las estructuras de datos son formas de organizar y almacenar datos para que puedan ser accedidos y modificados de manera eficiente. Los tipos de datos más comunes incluyen:

Datos numéricos que pueden ser enteros o reales los mismos que permiten realizar cálculos.

Datos de carácter o cadenas que son útiles para manejar información de tipo texto.

Las operaciones y operadores

En programación, las **operaciones** son las acciones que se realizan sobre los datos, y los **operadores** son los símbolos que indican dichas acciones. Los operadores son fundamentales en la mayoría de los lenguajes de programación, ya que permiten manipular los datos, realizar cálculos, tomar decisiones, y controlar el flujo de un programa. Dependiendo de su función, los operadores se dividen en diferentes categorías. (Moreno, 2015).

Tipos de Operadores

A continuación, se detallan los tipos de operadores más comunes en los lenguajes de programación:

Operadores Aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas básicas sobre valores numéricos. Son los más utilizados para la manipulación de datos numéricos y pueden aplicarse a enteros, flotantes, dobles, entre otros tipos de datos numéricos.

- **Suma (+):** Añade dos operandos.
    - Ejemplo: resultado = 5 + 3 (El resultado es 8).
- **Resta (-):** Resta el segundo operando del primero.
    - Ejemplo: resultado = 10 - 7 (El resultado es 3).

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- **Multiplicación (*):** Multiplica dos operandos.
    - Ejemplo: resultado = 4 * 6 (El resultado es 24).
- **División (/):** Divide el primer operando por el segundo.
    - Ejemplo: resultado = 20 / 4 (El resultado es 5).
    - Nota: La división entre cero no es válida y generalmente causa un error.
- **Módulo (%):** Devuelve el resto de la división del primer operando por el segundo.
    - Ejemplo: resultado = 10 % 3 (El resultado es 1).

Los operadores aritméticos son fundamentales en cálculos matemáticos y son utilizados en algoritmos que implican operaciones numéricas, como los algoritmos de ordenación, búsqueda, y otros.

Operadores Relacionales

Los operadores relacionales se utilizan para comparar dos valores y devuelven un resultado booleano (true o false). Estos operadores son esenciales en la toma de decisiones y control de flujo dentro de un programa, como en las estructuras condicionales (if, else) y bucles (while, for).

- **Igual a (==):** Verifica si dos operandos son iguales.
    - Ejemplo: 5 == 5 (Devuelve true).
- **Diferente de (!=):** Verifica si dos operandos no son iguales.
    - Ejemplo: 5 != 3 (Devuelve true).
- **Mayor que (>):** Verifica si el primer operando es mayor que el segundo.
    - Ejemplo: 10 > 7 (Devuelve true).
- **Menor que (<):** Verifica si el primer operando es menor que el segundo.
    - Ejemplo: 3 < 8 (Devuelve true).

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- **Mayor o igual que (>=):** Verifica si el primer operando es mayor o igual al segundo.
    - Ejemplo: 5 >= 5 (Devuelve true).
- **Menor o igual que (<=):** Verifica si el primer operando es menor o igual al segundo.
    - Ejemplo: 4 <= 6 (Devuelve true).

Los operadores relacionales son cruciales en la implementación de lógica condicional y estructuras de control que determinan el flujo de ejecución de un programa.

Operadores Lógicos

Los operadores lógicos se utilizan para realizar operaciones lógicas sobre valores booleanos (true o false). Se combinan comúnmente con operadores relacionales para evaluar múltiples condiciones dentro de estructuras de control como if, while, y for.

- **Y Lógico (&&):** Devuelve true si ambos operandos son true.
    - Ejemplo: (5 > 3) && (8 < 10) (Devuelve true).
- **O Lógico (||):** Devuelve true si al menos uno de los operandos es true.
    - Ejemplo: (5 > 3) || (8 > 10) (Devuelve true).
- **No Lógico (!):** Devuelve true si el operando es false, y viceversa.
    - Ejemplo: !(5 > 3) (Devuelve false).

Los operadores lógicos son utilizados principalmente para construir expresiones condicionales complejas y para controlar el flujo de ejecución en programas de software.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Operadores de Asignación

Los operadores de asignación se utilizan para asignar valores a las variables. El operador de asignación más básico es el = (igual), pero existen otros operadores de asignación combinados que realizan una operación aritmética y asignan el resultado al mismo tiempo.

- **Asignación Simple (=):** Asigna un valor a una variable.
    - Ejemplo: x = 5
- **Asignación Suma (+=):** Suma el valor de la derecha al de la variable.
    - Ejemplo: x += 3 es equivalente a x = x + 3.
- **Asignación Resta (-=):** Resta el valor de la derecha al de la variable.
    - Ejemplo: x -= 2 es equivalente a x = x - 2.
- **Asignación Multiplicación (*=):** Multiplica el valor de la derecha al de la variable.
    - Ejemplo: x *= 4 es equivalente a x = x * 4.
- **Asignación División (/=):** Divide el valor de la variable por el de la derecha.
    - Ejemplo: x /= 5 es equivalente a x = x / 5.

Operadores Unarios

Los operadores unarios se aplican a un solo operando y realizan operaciones como incrementar o decrementar su valor.

- **Incremento (++):** Aumenta el valor de una variable en uno.
    - Ejemplo: x++ es equivalente a x = x + 1.
- **Decremento (--):** Disminuye el valor de una variable en uno.
    - Ejemplo: x-- es equivalente a x = x - 1.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Variables y constantes

Las variables son contenedores que almacenan datos que pueden cambiar durante la ejecución del programa, es importante mencionar que toda variable tiene un nombre que sirve para identificarla en la ejecución del algoritmo. Las constantes son contenedores cuyo valor no cambia durante la ejecución. (Juganaru, 2015).

Es crucial declarar adecuadamente las variables y constantes dentro de un algoritmo con sus respectivos tipos.

Declarar una variable en PSeInt significa reservar un espacio en la memoria para guardar datos de un tipo específico. En PSeInt, los nombres de las variables deben comenzar con una letra y no pueden contener espacios ni caracteres especiales, excepto el guion bajo (_).

Definir nombre_variable Como tipo_de_dato

Donde: **nombre_variable**: Este es el nombre que se le asignará a la variable.

tipo_de_dato: El tipo de dato que la variable almacenará, como entero, real, carácter o lógico.

Las variables como contadores e incrementales juegan un papel crucial en la programación. Un contador es una variable que se utiliza para llevar un registro de la cantidad de veces que ocurre un evento, mientras que un incremental es una variable que se utiliza para aumentar o decrementar su valor en un patrón regular, como en bucles o iteraciones.

Tipos de Datos

Especifican el tipo de dato que una variable o constante puede almacenar. En PSeInt, hay diferentes tipos de datos que definen el tipo de información que puede guardarse en una variable.. A continuación, se describen algunos de los tipos de datos más comunes en PseInt:

- **Entero**: Este tipo de dato se emplea para guardar números enteros, es decir, aquellos que no tienen una parte decimal.
- **Real**: Se usa para guardar números que incluyen decimales, es decir una parte entera y una parte decimal.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- **Carácter**: Se utiliza para almacenar una secuencia de caracteres, como una palabra o una frase.
- **Lógico**: Se emplea para almacenar valores lógicos, como Verdadero o Falso.

Representación de algoritmo

Existen principalmente dos maneras en las que se suelen describir los algoritmos. Estas formas son esenciales para entender y comunicar cómo funcionan los algoritmos de manera efectiva.

Diagramas de flujo

Los diagramas de flujo son herramientas visuales que, a través de símbolos gráficos, ilustran cómo se va desarrollando el control en un proceso o algoritmo.. Utilizan símbolos gráficos para ilustrar las diferentes etapas, decisiones y secuencias de pasos necesarios para llegar a un resultado específico. (Acosta, 2009). Son especialmente útiles para planificar, documentar y comunicar procesos complejos de manera clara y comprensible, Existen algunos símbolos los más utilizados se encuentran en la tabla siguiente, aunque existen muchos más.

Tabla 1:
Símbolos, nombres y funciones de los elementos de un diagrama de flujo

Símbolo	Nombre	Función
	Inicio / Fin	Representa el Inicio y el Fin del algoritmo.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida.
	Proceso / instrucciones	Representa las operaciones que realiza el algoritmo

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

	Decisión	Representa la situación de decisión entre verdadero y falso.
	Flujo	Representa el orden de ejecución de cada operación o instrucción.

Nota: En la tabla se visualiza cada símbolo comúnmente utilizado en los diagramas de flujo, junto con su nombre y función.

Pseudocódigos

Los algoritmos se pueden describir de una manera que se asemeja a un lenguaje de programación, pero sin la rigidez de este. En lugar de seguir estrictas reglas sintácticas, se utilizan expresiones más cercanas al lenguaje natural, lo que facilita su comprensión y comunicación.

Ilustración 4:
Pseudocódigo de un algoritmo

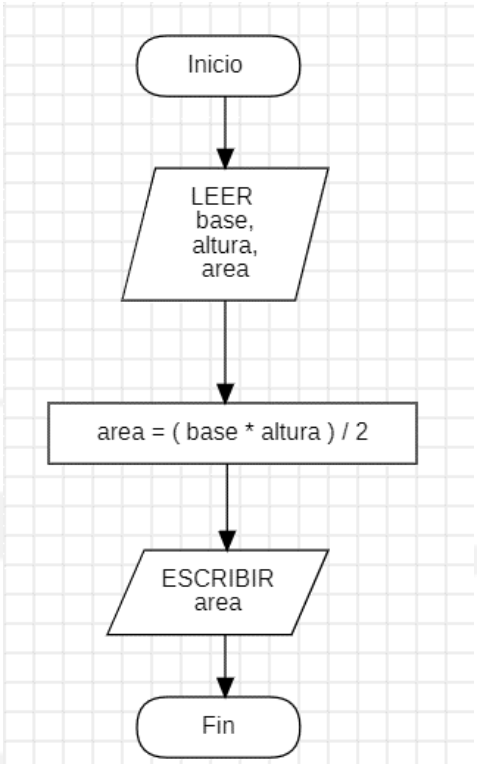
```
Algoritmo CalcularAreaTriangulo
    Definir base, altura, area Como Real
    Escribir "Ingrese la base del triángulo: "
    Leer base
    Escribir "Ingrese la altura del triángulo: "
    Leer altura
    area = (base * altura) / 2
    Escribir "El área del triángulo es: ", area
FinAlgoritmo
```

Nota: En la ilustración se puede observar un pseudocódigo de un algoritmo que realiza el cálculo del área de un triángulo, este pseudocódigo esta realizado en PSeInt. Fuente: Elaboración propia

Este algoritmo se puede ser representado en un diagrama de flujo utilizando herramientas en línea o instalables. En este caso, se utilizará UMLStart, que es una herramienta que se instala en el computador. Existen algunas herramientas en línea tales como Lucidchart, Draw.io, Creately, que son herramienta en línea que permite crear diagramas de flujo de forma colaborativa.
A continuación, se proporciona el enlace para descargar de UMLStart, disponible para los sistemas operativos GNU/Linux, Windows y MacOS: Descargar UMLStart.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 5:
Diagrama de flujo de un algoritmo



Nota: En la ilustración se puede observar el diagrama de flujo del algoritmo que realiza el cálculo del área de un triángulo, este diagrama esta realizado en UMLStart. Fuente: Elaboración propia

Estructuras de control

Las estructuras de control son esenciales en la programación, ya que permiten dirigir el flujo de ejecución del programa. Existen tres tipos principales de estructuras de control:

- **Secuenciales**: Ejecutan un bloque de instrucciones de manera lineal, una tras otra tal como están escritas en el algoritmo.
- **Condicionales o Selectivas**: Permiten ejecutar diferentes conjuntos de instrucciones según se cumpla o no una condición específica.
- **Bucles o Repeticiones**: Habilitan la ejecución repetida de un conjunto de instrucciones, ya sea un número determinado de veces o hasta que se cumpla una condición.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Estas estructuras son fundamentales para crear programas más dinámicos y adaptativos.

Estructuras de control secuenciales

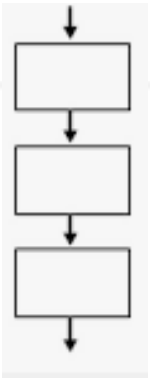
La estructura de control secuencial organiza las instrucciones de un programa de manera que se ejecutan una tras otra, en el orden en que aparecen. (Oviedo, 2015). Es la forma más básica y fundamental en programación, donde cada instrucción se ejecuta una sola vez, a menos que se modifique el flujo con otra estructura de control.

Características de la Estructura Secuencial:

- Orden Lineal: Las instrucciones se procesan de arriba hacia abajo, siguiendo el orden escrito.
- Simplicidad: Son fáciles de entender, ya que no hay bifurcaciones ni repeticiones.
- Flujo Directo: No hay saltos ni bucles, lo que hace que el control del flujo sea claro y predecible.

Esta estructura es esencial para construir la base de cualquier programa.

Ilustración 6:
Representación de estructura de control secuencial



Nota: En la ilustración se puede observar un diagrama de flujo que representa una estructura de control secuencial. En una estructura de control secuencial, las instrucciones se ejecutan una tras otra, en el orden en que aparecen. Fuente: Elaboración propia.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 7:
Estructura de control secuencial

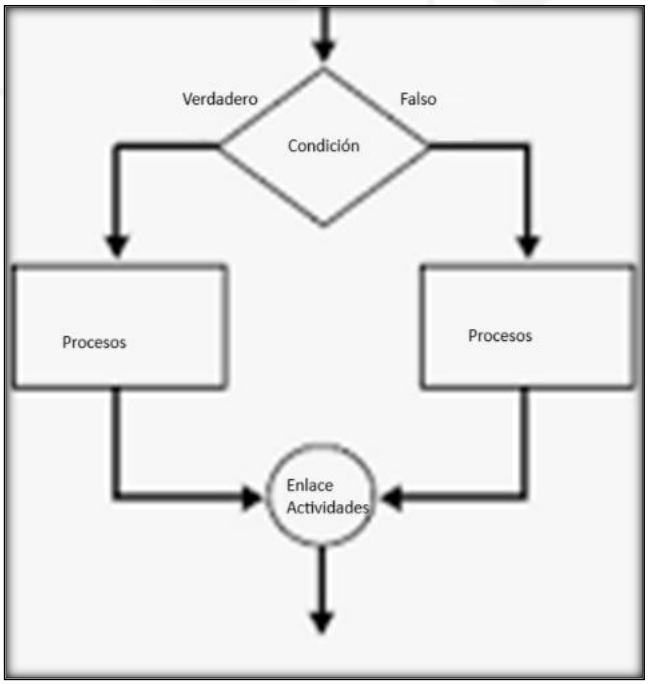
```
Algoritmo CalcularAreaRectangulo
    Definir long, ancho, area Como Real // Definición de variables
    Escribir "Ingrese la longitud del rectángulo:"  // Solicitar al usuario la longitud del rectángulo
    Leer long
    Escribir "Ingrese el ancho del rectángulo:"   // Solicitar al usuario el ancho del rectángulo
    Leer ancho
    area = long * ancho  // Calcular el área del rectángulo
    Escribir "El área del rectángulo es: ", area   // Mostrar el área del rectángulo
FinAlgoritmo

```

Nota: En la ilustración se puede observar el pseudocódigo de un algoritmo que calcula el área de un rectángulo a partir de la longitud y el ancho proporcionados por el usuario, usando una estructura de control secuencial. Fuente: Elaboración propia.

Estructuras de control condicional

La estructura de control condicional o selectiva permite que un programa ejecute diferentes bloques de código según si una condición específica es verdadera o falsa. (Farrell, 2013). Estas estructuras son cruciales para tomar decisiones dentro del programa, ya que permiten que el flujo de ejecución siga distintos caminos basados en condiciones lógicas.

Ilustración 8:
Representación de estructura de control secuencial



(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Nota: En la ilustración se puede observar un diagrama de flujo que representa una estructura de control condicional. Se utiliza para tomar decisiones en un programa, ejecutando diferentes bloques de código en función de si una condición específica se evalúa como T (Verdadera o True) o F (Falsa o False). Fuente: Elaboración propia.

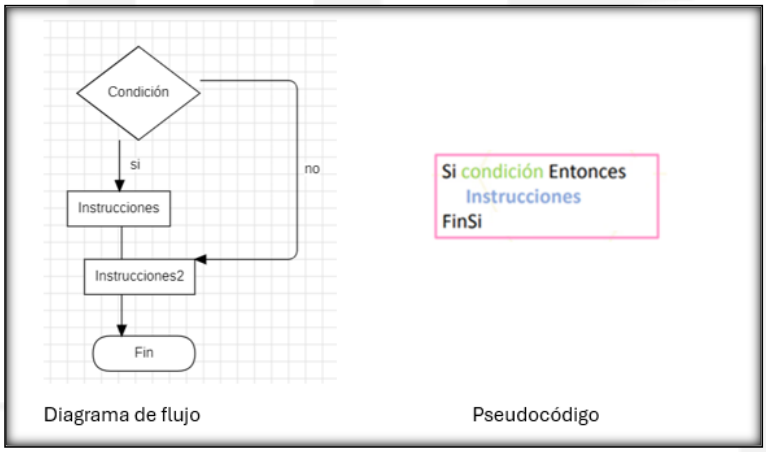
Tipos de estructuras de control condicionales

Existen distintas estructuras de control entre las e se puede mencionar:

If-Else o Si-Entonces

La instrucción *If* o *Si* evalúa una condición; si es verdadera, se ejecuta un bloque de código. Si es falsa, se puede ejecutar un bloque alternativo usando *Else* o *SiNo*. (Arteaga, 2023).

Ilustración 9:
Pseudocódigo y Diagrama de flujo de la estructura de control Si-Entonces

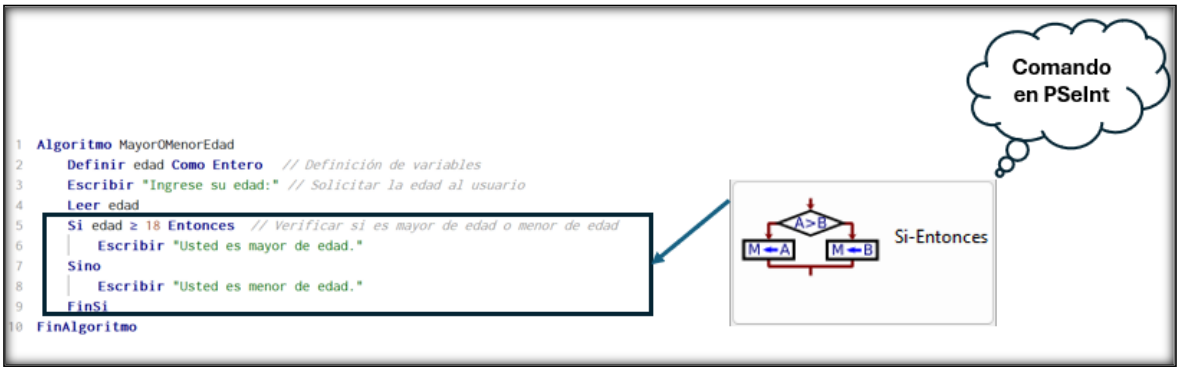


Nota: En la ilustración se puede observar el diagrama de flujo y el pseudocódigo de la estructura de control condicional Si-Entonces. Fuente: Elaboración propia.

En este algoritmo se visualiza un pseudocódigo en PSeInt usando el comando Si-Entonces

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 10:
Pseudocódigo usando la estructura de control Si-Entonces



Nota: En la ilustración se puede observar el pseudocódigo de un algoritmo que determina si la edad ingresada corresponde a un mayor o menor de edad, utilizando una estructura de control condicional Si-Entonces. Fuente: Elaboración propia.

Ejemplo: Diseñar un algoritmo que determine si un número ingresado por el usuario es par o impar. El programa debe solicitar al usuario que ingrese un número entero y, luego, verificar si el número es divisible por 2. Si es divisible por 2, el programa mostrará el mensaje "El número es par". Si no lo es, mostrará "El número es impar".

Solución del ejemplo: Para resolver este problema empleando la herramienta PSeInt, se usa una estructura de control condicional if, una de las posibles soluciones podría seguir los siguientes pasos:

1. **Entrada:** Pedir al usuario que ingrese un número entero.
2. **Proceso:**
    - Definir variable para asignar la edad ingresada.
    - Evaluar si el número ingresado es divisible por 2 (usando el operador módulo %).
    - Si el resultado del módulo es 0 (es decir, no hay residuo), el número es par y mostrar el mensaje "El número es par".
    - Si el resultado del módulo no es 0, el número es impar y mostrar el mensaje "El número es impar".
3. **Salida:** El mensaje que identifica si el número es par o impar.

Algoritmo en PSeInt:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 11:
Algoritmo en Pseint ejemplo de estructura de control Si-Entonces

```
Algoritmo Par_Impar
    Definir numero Como Entero

    // Paso 1: Solicitar al usuario que ingrese un número entero
    Escribir "Ingrese un número entero:"
    Leer numero

    // Paso 2: Verificar si el número es par o impar usando el operador módulo
    Si numero % 2 = 0 Entonces
        // Paso 3a: Si la condición es verdadera (el módulo es 0), el número es par
        Escribir "El número es par."
    Sino
        // Paso 3b: Si la condición es falsa (el módulo no es 0), el número es impar
        Escribir "El número es impar."
    FinSi
FinAlgoritmo
```

Nota: En la ilustración se puede observar el algoritmo que determina si el numero ingresado es par o impar, utilizando una estructura de control condicional *Si-Entonces*. Fuente: Elaboración propia.

Switch-Case-Según

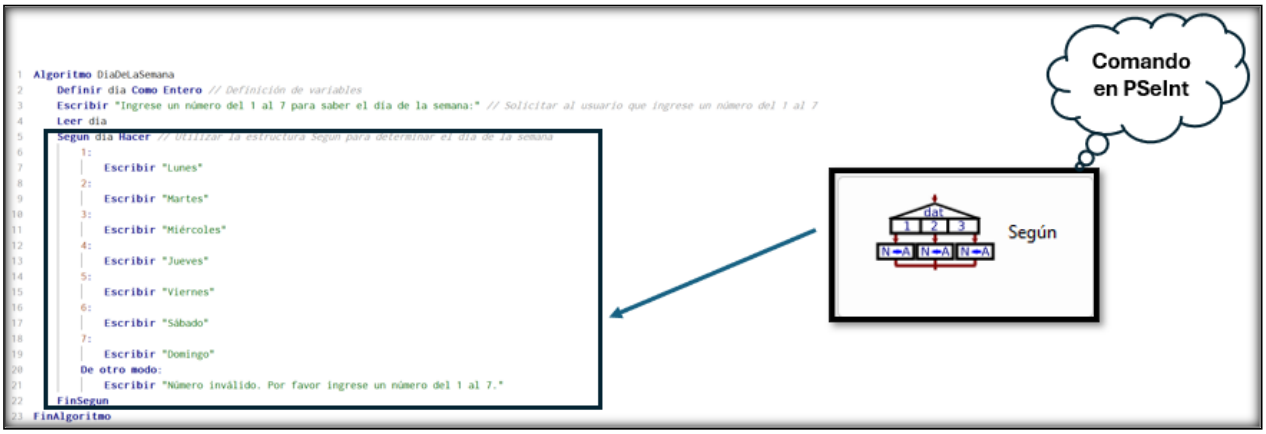
La instrucción switch o también llamada Case o Según evalúa el valor de una variable y ejecuta uno de varios bloques de código según ese valor. Es especialmente útil cuando hay múltiples condiciones que dependen de una sola variable. (Ramírez, 2019).

Estas estructuras son fundamentales para crear programas que respondan de manera dinámica a diferentes situaciones.

En este algoritmo se visualiza un pseudocódigo en PSeInt usando el comando Según.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 12:
Pseudocódigo usando la estructura de control Switch – Case - Según



Nota: En la ilustración se puede observar el pseudocódigo de un algoritmo que determina el día de la semana basado en un número ingresado por el usuario, utilizando una estructura de control condicional *Según*. Fuente: Elaboración propia.

Ejemplo: Diseña un algoritmo para un programa que permita al usuario seleccionar una operación aritmética básica (suma, resta, multiplicación o división) e ingrese dos números. El programa debe realizar la operación seleccionada y mostrar el resultado. Si el usuario selecciona una opción inválida, el programa debe mostrar un mensaje de error.

Solución del ejemplo: Para resolver este problema empleando la herramienta PSeInt, se usa una estructura de control condicional *Según* una de las posibles soluciones podría seguir los siguientes pasos:

1. **Entrada:**
    - Solicitar al usuario que ingrese un número que represente la operación deseada, por ejemplo:
        - 1 para Suma
        - 2 para Resta
        - 3 para Multiplicación
        - 4 para División
    - Luego, solicitar al usuario que ingrese dos números sobre los cuales se realizará la operación.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

2. **Proceso:**
    - Definir variable para asignar la operación, los números ingresados y el resultado de la operación.
    - Usar la estructura según para determinar qué operación realizar según la opción ingresada por el usuario.
    - Realizar la operación correspondiente entre los dos números ingresados.
    - Si la opción es 4 (División) y el segundo número es 0, mostrar un mensaje de error indicando que no se puede dividir por cero.
3. **Salida**: El resultado de la operación o un mensaje de error si la opción es inválida o hay un intento de división por cero.

Algoritmo en PSeInt:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 13:
Algoritmo en Pseint ejemplo de estructura de control Segun

```
Algoritmo OperacionesBasicas
    Definir opcion, numero1, numero2 Como Real
    Definir resultado Como Real

    // Paso 1: Solicitar al usuario que seleccione una operación
    Escribir "Seleccione una operación:"
    Escribir "1. Suma"
    Escribir "2. Resta"
    Escribir "3. Multiplicación"
    Escribir "4. División"
    Leer opcion

    // Paso 2: Solicitar al usuario que ingrese dos números
    Escribir "Ingrese el primer número:"
    Leer numero1
    Escribir "Ingrese el segundo número:"
    Leer numero2

    // Paso 3: Utilizar la estructura 'según' para determinar la operación
    Según opcion Hacer
1:
    // Suma
    resultado = numero1 + numero2
    Escribir "El resultado de la suma es: ", resultado
2:
    // Resta
    resultado = numero1 - numero2
    Escribir "El resultado de la resta es: ", resultado
3:
    // Multiplicación
    resultado = numero1 * numero2
    Escribir "El resultado de la multiplicación es: ", resultado
4:
    // División
    Si numero2 ≠ 0 Entonces
        resultado = numero1 / numero2
        Escribir "El resultado de la división es: ", resultado
    Sino
        Escribir "Error: No se puede dividir por cero."
    FinSi
De otro modo:
    // Opción inválida
    Escribir "Opción inválida. Por favor, seleccione una opción del 1 al 4."
FinSegún
FinAlgoritmo
```

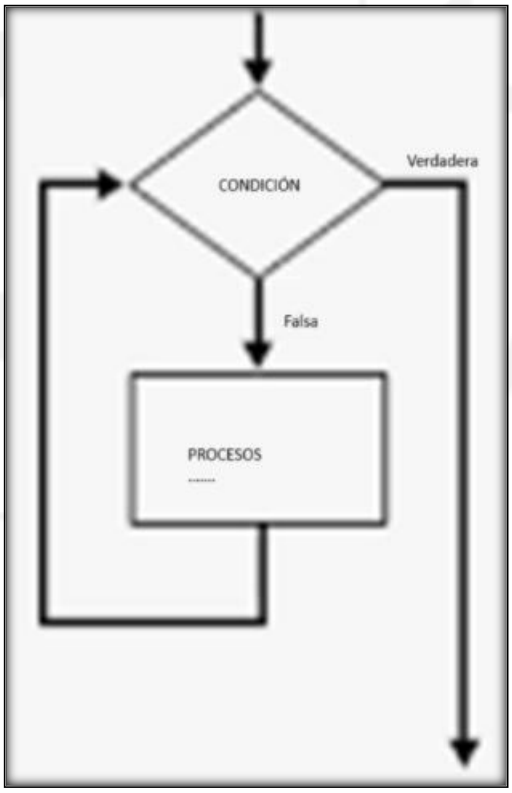
Nota: En la ilustración se puede observar el algoritmo que realiza las operaciones básicas de suma, resta, multiplicación y división entre dos números, utilizando una estructura de control condicional *Según* y para la validación de la división la estructura *If./Else* Fuente: Elaboración propia.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Estructuras de control repetitiva

Las estructuras de control repetitivas, comúnmente conocidas como bucles o iteraciones, son esenciales en programación. Permiten ejecutar un bloque de código varias veces, lo que resulta fundamental para tareas que requieren repetición, como procesar elementos de una lista, realizar cálculos iterativos o verificar condiciones hasta que se cumpla un criterio específico. (Aristizábal, 2023). Se analizará los diferentes tipos de estructuras de control repetitivas que se encuentran centrándose en PSeInt.

Ilustración 14:
Representación de estructura de control repetitiva



Nota: En la ilustración se puede observar un diagrama de flujo que diagrama de flujo que representa una estructura de control repetitiva. Fuente: Elaboración propia.

For-Para

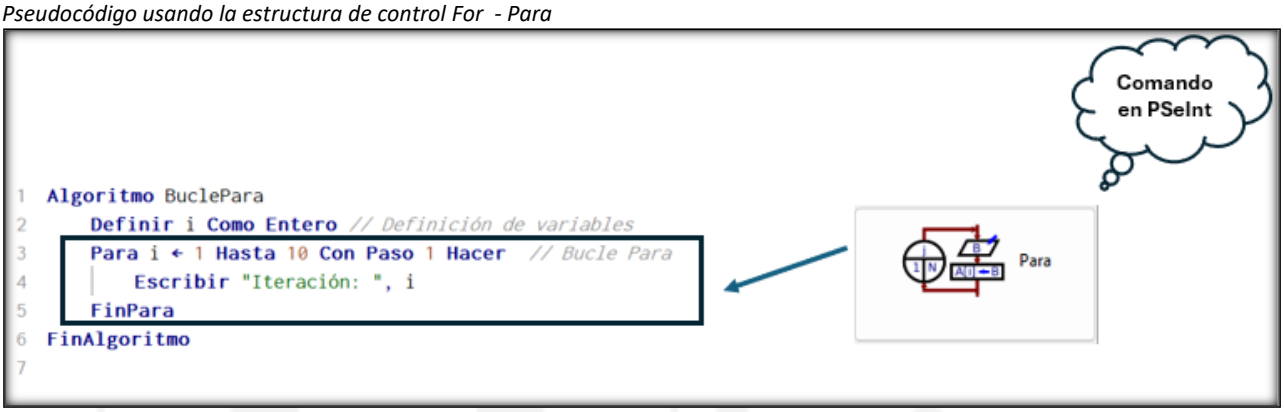
La instrucción *For* o también llamada *Para* se utiliza cuando se conoce de antemano el número de iteraciones que se desea realizar. (Becerra, 2020). Es ideal para situaciones en las que se necesita iterar un número fijo de veces.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Para ello, se utiliza una variable de control, conocida también como índice, que recorre un conjunto predefinido de valores en un orden específico. Además, se emplea un contador que acumula el número de veces que se han repetido las instrucciones y ayuda a determinar el valor final que debe alcanzar el índice.

En este algoritmo se visualiza un pseudocódigo en PSeInt usando el comando Para.

Ilustración 15:
Pseudocódigo usando la estructura de control For - Para



Nota: En la ilustración se puede observar el pseudocódigo de un algoritmo que utiliza la estructura de control repetitiva Para en PSeInt que imprime el número de veces que se repite en cada interacción. Fuente: Elaboración propia.

Ejemplo: Diseña un algoritmo para un programa que calcule la suma de los primeros 10 números naturales utilizando un bucle. El programa debe mostrar el resultado al finalizar la suma.

Solución del ejemplo: Para resolver este problema empleando la herramienta PSeInt, se usa una estructura de control condicional *Para,* una de las posibles soluciones podría seguir los siguientes pasos:

1. **Inicialización:**
    - Definir variable para asignar el resultado de la suma que comenzará en 0 y servirá para acumular la suma de los números.
2. **Proceso:**
    - Utilizar la estructura Para para iterar del 1 al 10.
    - En cada iteración, sumar el valor del contador i a la variable suma.
3. **Salida**: Mostrar el resultado de la suma de los primeros 10 números naturales.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Algoritmo en PSeInt:

Ilustración 16:
Algoritmo en Pseint ejemplo de estructura de control Para

```
Algoritmo SumaPrimerosDiezNumeros
    Definir suma Como Entero
    suma = 0  // Inicializar la variable suma en 0

    // Paso 2: Utilizar la estructura 'Para' para iterar de 1 a 10
    Para i = 1 Hasta 10 Hacer
        // En cada iteración, sumamos el valor de i a la variable suma
        suma = suma + i
    FinPara

    // Paso 3: Mostrar el resultado de la suma
    Escribir "La suma de los primeros 10 números naturales es: ", suma
FinAlgoritmo
```

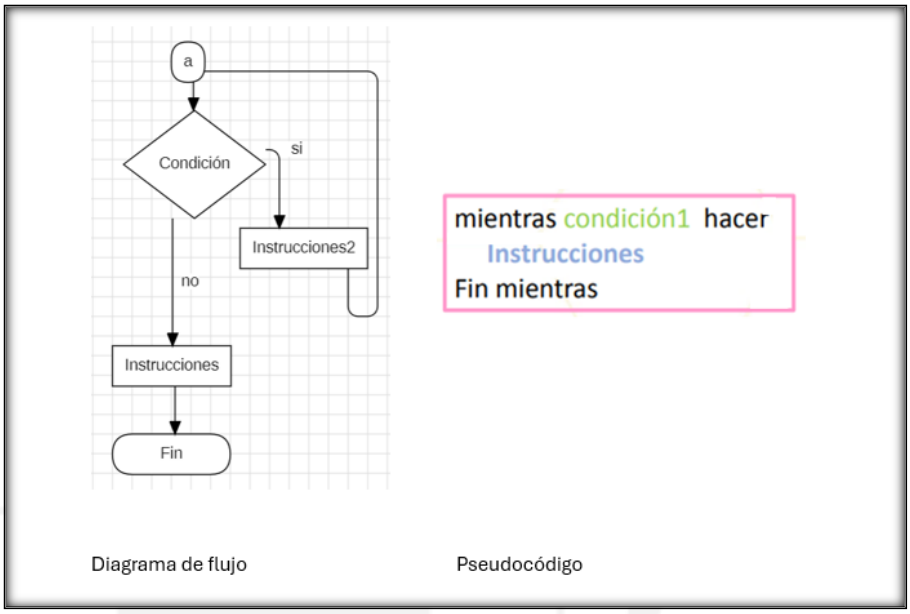
Nota: En la ilustración se puede observar el algoritmo que utilizar la estructura de repetición *Para* para iterar desde 1 hasta 10, acumulando el total de la suma. Fuente: Elaboración propia.

Mientras-While

La instrucción *Mientras* o también llamada *While*, se utiliza cuando no se conoce el número exacto de iteraciones con antelación. (Duran, 2007). En su lugar, el bucle sigue ejecutándose mientras una condición específica sea verdadera.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

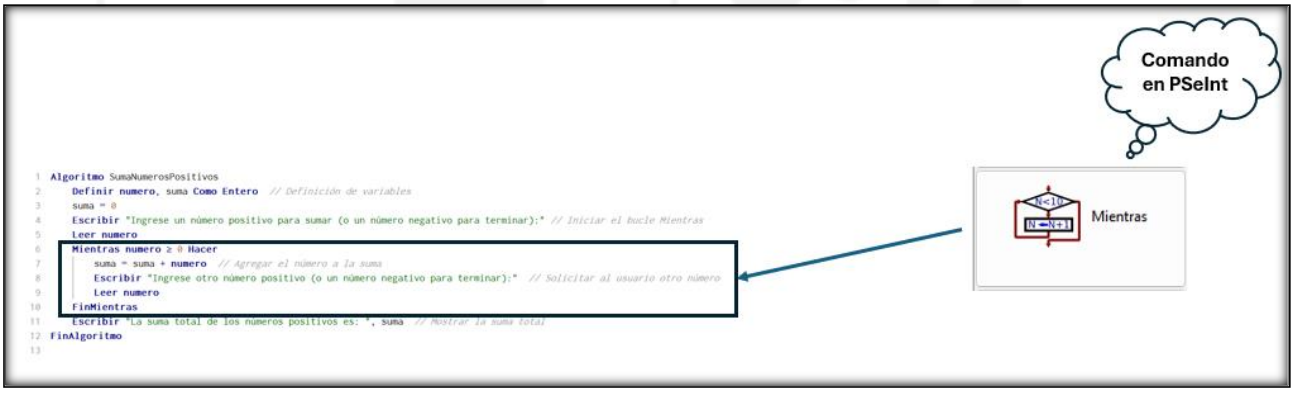
Ilustración 17:
Diagrama de flujo y Pseudocódigo de la estructura de control Mientras - While



Nota: En la ilustración se puede observar el diagrama de flujo y el pseudocódigo de la estructura de control repetitiva *Mientras – While*. Fuente: Elaboración propia.

En este algoritmo se visualiza un pseudocódigo en PSeInt usando el comando *Mientras*.

Ilustración 18:
Pseudocódigo usando la estructura de control Mientras - While



Nota: En la ilustración se puede observar el pseudocódigo de un algoritmo que utiliza la estructura de control repetitiva Mientras en PSeInt que suma una serie de números ingresados por el usuario hasta que el usuario ingrese un número negativo, momento en el cual el algoritmo termina y muestra la suma total. Fuente: Elaboración propia.

Ejemplo: Diseña un algoritmo para un programa que permita al usuario adivinar un número secreto entre 1 y 10. El programa debe solicitar al usuario que ingrese un número y comparar ese número con el número secreto. Si el usuario no acierta, el programa debe indicarlo y

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

permitirle intentarlo de nuevo. Este proceso se repetirá hasta que el usuario adivine correctamente el número secreto. Al final, el programa debe mostrar un mensaje de felicitación.

Solución del ejemplo: Para resolver este problema empleando la herramienta PSeInt, se usa una estructura de control condicional *Mientras,* una de las posibles soluciones podría seguir los siguientes pasos:

1. **Inicialización:**
    - Definir una variable que almacene un número secreto (en este ejemplo en número 3).
    - Definir una variable adivinanza para almacenar el número que el usuario ingresa.
2. **Proceso:**
    - Usar la estructura Mientras para permitir que el usuario siga intentando adivinar mientras su número no sea igual al número secreto.
    - Dentro del bucle, pedir al usuario que ingrese un número.
    - Usar la estructura de control If-Else. Si el número ingresado no es correcto, indicar que intente de nuevo.
3. **Salida**: Cuando el usuario acierta el número secreto, mostrar un mensaje de felicitación y el bucle termina.

Algoritmo en PSeInt:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 19:
Algoritmo en PseInt ejemplo de estructura de control Mientras

```
Algoritmo AdivinaElNumero
    Definir numeroSecreto, adivinanza Como Entero
    numeroSecreto = 3  // Paso 1: Establecer el número secreto
    adivinanza = 0  // Inicializar la adivinanza en un valor que no sea el número secreto

    // Paso 2: Utilizar la estructura 'Mientras' para permitir que el usuario adivine
    Mientras adivinanza ≠ numeroSecreto Hacer
        // Solicitar al usuario que ingrese un número
        Escribir "Adivina el número entre 1 y 10:"
        Leer adivinanza

        // Verificar si el número es incorrecto
        Si adivinanza ≠ numeroSecreto Entonces
            Escribir "Incorrecto. Intenta de nuevo."
        FinSi
    FinMientras

    // Paso 3: Cuando se adivina correctamente, mostrar mensaje de felicitación
    Escribir "¡Felicidades! Adivinaste el número secreto."
FinAlgoritmo
```

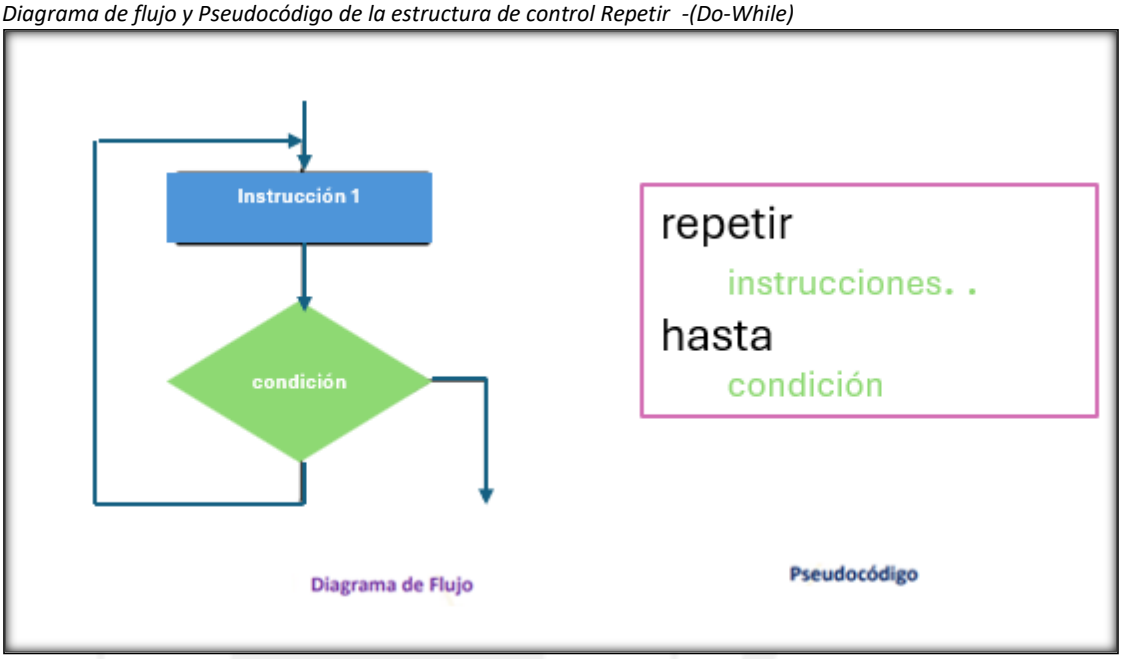
Nota: En la ilustración se puede observar el algoritmo que utilizar la estructura de repetición *Mientras*, hasta que el usuario adivine correctamente el número secreto, también se usa la estructura de control *Si-Else* para determinar si es diferente o igual el numero ingresado al número secreto. Fuente: Elaboración propia.

Repetir – (Do-While)

El bucle Repetir - Hasta Que, conocido también como Do-While, es similar al bucle Mientras, pero con una diferencia clave: el bloque de código se ejecuta al menos una vez antes de que se evalúe la condición. (Duran, 2007). Esto es útil cuando se desea garantizar que el bucle se ejecute al menos una vez.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

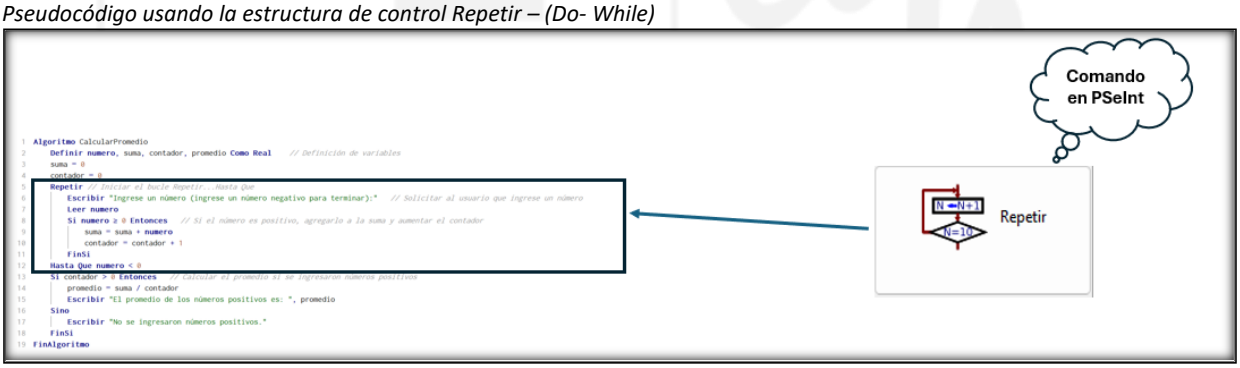
Ilustración 20:
Diagrama de flujo y Pseudocódigo de la estructura de control Repetir -(Do-While)



Nota: En la ilustración se puede observar el diagrama de flujo y el pseudocódigo de la estructura de control repetitiva *Repetir – (Do-While).*Fuente: Elaboración propia.

En este algoritmo se visualiza un pseudocódigo en PSeInt usando el *comando Repetir – (Do-While.*

Ilustración 21:
Pseudocódigo usando la estructura de control Repetir – (Do- While)



Nota: En la ilustración se puede observar el pseudocódigo de un algoritmo que utiliza la estructura de control repetitiva Repetir en PSeInt que calcula el promedio de una serie de números ingresados por el usuario hasta que el usuario ingrese un número negativo. Fuente: Elaboración propia.

Ejemplo: Diseña un algoritmo para un programa un programa que permita a un cajero registrar varias transacciones de depósito en una cuenta bancaria. El programa debe solicitar al cajero que ingrese un monto de depósito repetidamente y, al final, mostrar el total acumulado de los depósitos realizados. El programa debe detenerse cuando el cajero ingrese un monto igual a 0.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Solución del ejemplo: Para resolver este problema empleando la herramienta PSeInt, se usa una estructura de control condicional *Repetir,* una de las posibles soluciones podría seguir los siguientes pasos:

1. **Inicialización:**
    - Definir una variable totalDepositos para almacenar la suma acumulada de todos los depósitos. Inicialmente a esta variable se le asigna 0.
    - Definir una variable deposito para almacenar el monto de cada depósito ingresado por el cajero.
2. **Proceso:**
    - Usar la estructura Repetir para solicitar al cajero que ingrese un monto de depósito.
    - Si el monto ingresado es diferente de 0, el monto se suma a la variable totalDepositos.
    - El bucle Repetir continúa hasta que el cajero ingrese 0, momento en el cual el bucle se detiene.
3. **Salida**: Al finalizar el bucle, el programa muestra la variable totalDepositos, que es la suma de todos los depósitos ingresados.

Algoritmo en PSeInt:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 22:
Algoritmo en Pseint ejemplo de estructura de control Repetir

```
Algoritmo RegistroDeDepositos
    Definir totalDepositos, deposito Como Real
    totalDepositos = 0  // Inicializar el total de depósitos en 0

    // Paso 2: Utilizar la estructura 'Repetir' para solicitar montos de depósito
    Repetir
        // Solicitar al cajero que ingrese el monto del depósito
        Escribir "Ingrese el monto del depósito (0 para terminar):"
        Leer deposito

        // Verificar que el depósito no sea 0 antes de sumarlo
        Si deposito ≠ 0 Entonces
            totalDepositos = totalDepositos + deposito
        FinSi
    Hasta Que deposito = 0

    // Paso 3: Mostrar el total acumulado de los depósitos
    Escribir "El total de los depósitos es: ", totalDepositos
FinAlgoritmo
```

Nota: En la ilustración se puede observar el algoritmo que utilizar la estructura de Repetir, se ingresa repetidamente montos de depósito y, al final, se muestra el total acumulado. Fuente: Elaboración propia.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Autoevaluación 2

Lea detenidamente la pregunta y marque la respuesta correcta entre las opciones dadas.

1. **¿Cuáles son las tres acciones clave en un algoritmo?**

   a) Entrada, Salida, Error.

   b) Entrada, Proceso, Salida.

   c) Entrada, Bucle, Salida.

   d) Entrada, Decisión, Salida.

2. **¿Qué tipo de dato se utiliza en PSeInt para almacenar el numero 89.41?**

   a) Real.

   b) Carácter.

   c) Lógico.

   d) Entero.

3. **¿Cuál es la función del símbolo de proceso en un diagrama de flujo?**

   a) Representar el inicio y el fin del algoritmo.

   b) Representar la lectura y escritura de datos.

   c) Representar las operaciones que realiza el algoritmo.

   d) Representar la toma de decisiones.

4. **¿Cuál de los siguientes operadores es un operador aritmético en PSeInt?**

   a) &&

   b) ==

   c) +

   d) !=

5. **¿Qué tipo de dato se utiliza en PSeInt para almacenar valores booleanos?**

   a) Entero.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

b) Real.

c) Carácter.

d) Lógico.

6. ¿Cuál es la estructura de control que ejecuta un bloque de instrucciones de manera lineal?

a) Estructura condicional.

b) Estructura secuencial.

c) Estructura repetitiva.

d) Estructura selectiva.

7. ¿Qué instrucción se utiliza en PSeInt para declarar una variable?

a) Leer.

b) Escribir.

c) Definir.

d) Proceso.

8. ¿Cuál es la función de la estructura de control "Si-Entonces" en PSeInt?

a) Tomar decisiones basadas en condiciones

b) Ejecutar un bloque de código una cantidad específica de veces.

c) Declarar variables y constantes.

d) Repetir un bloque de código hasta que una condición sea falsa.

9. ¿Cuál de las siguientes estructuras repetitivas garantiza que el bloque de código se ejecutará al menos una vez?

a) Para.

b) Mientras.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

c) Repetir...Hasta Que

d) Si-Entonces

10. ¿Qué hace la estructura de control Mientras en programación?

a) Evalúa una condición y, si es verdadera, ejecuta un bloque de código, repitiendo esta evaluación antes de cada iteración.

b) Ejecuta un bloque de código un número fijo de veces considerando un índice y un contador.

c) Evalúa una condición y, si es verdadera, ejecuta un bloque de código solo una vez

d) Ejecuta un bloque de código al menos una vez y luego evalúa una condición para repetirlo

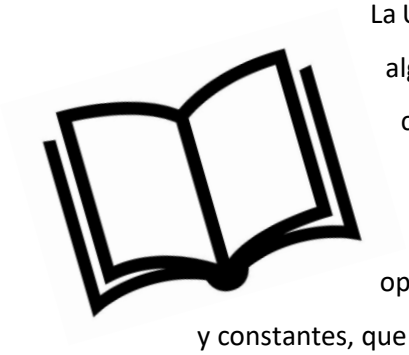
Si ha concluido la autoevaluación:

> Verifique sus respuestas en el solucionario que se encuentra al final de la presente Guía.

Si alcanzó un porcentaje alto de respuestas correctas en las preguntas de evaluación, es hora de continuar con la Unidad 3, caso contrario, vuelva a revisar los temas en los que ha tenido dificultad.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Resumen de la Unidad 2



La Unidad 2 se centra en los elementos esenciales que componen un algoritmo y su representación en herramientas de programación como PSeInt y Java. A lo largo de esta unidad, se abordan las acciones clave de un algoritmo, que son: Entrada, Proceso y Salida. También se describen las estructuras de datos, las operaciones y los operadores, así como la declaración de variables y constantes, que son fundamentales para construir algoritmos efectivos.

La estructura de un algoritmo se puede definir como:

- Entrada: Es la información inicial necesaria para comenzar el algoritmo.
- Proceso: Son las operaciones que se llevan a cabo sobre la información.
- Salida: Son los resultados obtenidos al final del proceso.

Entre los elementos de un Algoritmo se puede citar:

- Estructuras de Datos: Son las maneras de organizar y almacenar datos, que pueden ser numéricos, caracteres o cadenas.
- Operaciones y Operadores: Se refieren a las acciones que se realizan sobre los datos, utilizando operadores aritméticos, relacionales y lógicos.
- Declaración de Variables y Constantes: Esto implica asignar nombres y tipos de datos para almacenar valores que pueden cambiar (variables) o que permanecen constantes (constantes) durante la ejecución del programa.
- Tipos de Datos en PSeInt y Java

  PSeInt: Ofrece tipos de datos como Entero, Real, Carácter y Lógico.

  Java: Tiene tipos de datos Primitivos (int, float, char, boolean) y No Primitivos (String, Arrays, Clases).

Los algoritmos se pueden representar mediante:

- Diagramas de Flujo: Son herramientas visuales que muestran el flujo de control de un proceso o algoritmo utilizando símbolos gráficos.
- Pseudocódigos: Son descripciones que se asemejan a un lenguaje de programación y facilitan la comprensión y comunicación de algoritmos.

Existen estructuras de control:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Secuenciales: Ejecutan instrucciones de forma lineal.
- Condicionales: Permiten ejecutar diferentes conjuntos de instrucciones basadas en una condición.
    - If-Else: Evalúa una condición y ejecuta un bloque de código basado en el resultado.
    - Switch-Case: Evalúa el valor de una variable y ejecuta uno de varios bloques de código.
- Repetitivas: Ejecutan un bloque de código varias veces.
    - Para (For): Itera un número fijo de veces.
    - Mientras (While): Itera mientras una condición sea verdadera.
    - Repetir-Hasta Que (Do-While): Itera al menos una vez antes de evaluar la condición.

La unidad enfatiza la importancia de las estructuras de control en programación, presentando ejemplos prácticos y diagramas que ilustran su funcionamiento. Además, se destacan herramientas y recursos como UMLStart, Lucidchart, Draw.io y Creately, que son ideales para la creación de diagramas de flujo.

¡Continúe así! ¡Le deseo mucho éxito!

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

UNIDAD 3. (PROGRAMACION ORIENTADA A OBJETOS)

"Nunca consideres el estudio como una obligación, sino como una oportunidad para ingresar en el bello y maravilloso mundo del saber."

Albert Einstein

Temas y Subtemas

- Introduccion a la programación orientada a objetos
- IDEs y Frameworks
- Estructura y funcionamiento de un Programa en Java
- Manejo de tipos de datos en Java

Introducción a la programación orientada a objetos

La programación orientada a objetos (POO) es un enfoque de programación que utiliza "objetos" para diseñar aplicaciones y programas. (Spigariol, 2021). Este paradigma facilita la creación de software que es más modular, reutilizable y escalable.

En esta unidad, se tratará los pilares fundamentales de la POO, se explorará el entorno de desarrollo integrado (IDE) NetBeans para temas de estudio de esta materia, y se aprenderá sobre Java como lenguaje de programación, donde se revisarán los tipos de datos en Java, el uso de arrays y las operaciones de entrada/salida en este lenguaje.

La POO nos permite organizar el código de manera más intuitiva, lo que resulta en aplicaciones más robustas y fáciles de mantener. A medida que avanzamos, descubriremos cómo estos conceptos se aplican en la práctica y cómo pueden mejorar nuestra capacidad para desarrollar soluciones efectivas en el mundo del software.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Hoy en día, hay muchos lenguajes de programación orientada a objetos, como Java, Python, C++, PHP y Ruby. Java es uno de los lenguajes más populares y utilizado en el desarrollo web y de aplicaciones móviles.

Ventajas de la programación orientada a objetos

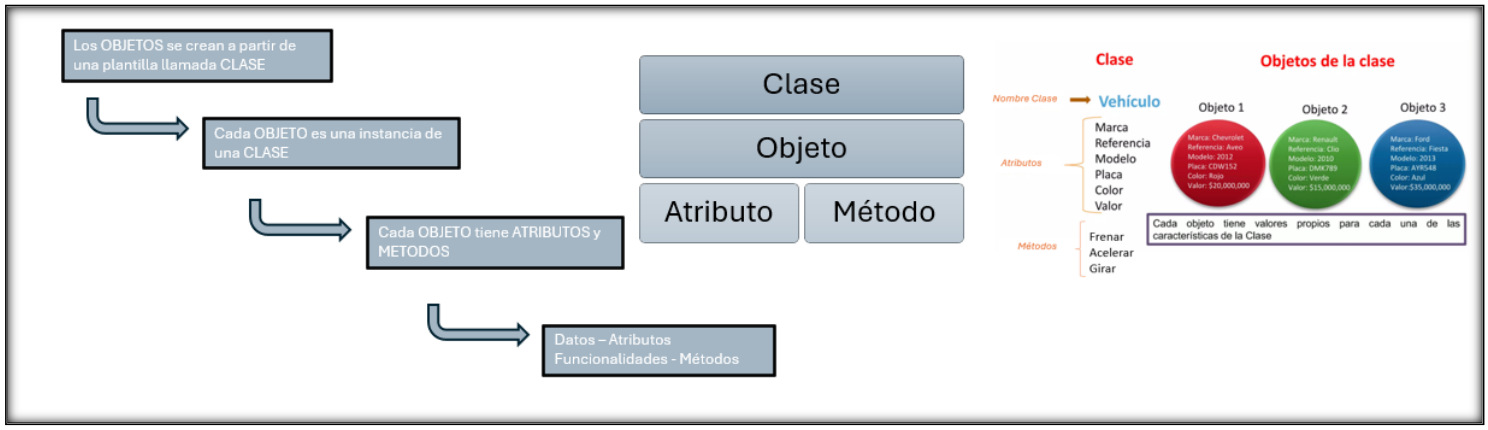
La programación orientada a objetos (POO) ofrece varias ventajas para los desarrolladores:

- **Reutilización del código**: Una de las grandes ventajas de la POO es la posibilidad de reutilizar código heredado. Al diseñar clases de manera adecuada, se puede utilizarlas en diferentes partes de un programa o incluso en distintos proyectos. Gracias a la herencia, se puede crear una clase base y luego definir subclases que hereden sus características, lo que ahorra tiempo y esfuerzo, ya que no se necesita reescribir funciones. Además, si se realiza un cambio en la clase base, todas las subclases lo adoptarán automáticamente.
- **Flexibilidad**: La POO facilita la modificación de programas, se puede añadir, cambiar o eliminar objetos y funciones de manera sencilla, lo que ahorra tiempo y esfuerzo a los programadores al actualizar el software.
- **Detección de errores simplificada**: En la programación orientada a objetos, no es necesario revisar cada línea de código para encontrar errores. Gracias a la encapsulación, los objetos son autónomos, lo que permite abstraer problemas y localizar errores más fácilmente cuando algo no funciona como debería.
- **Modularidad**: El modularidad es otra característica de la POO. Permite que diferentes miembros de un equipo trabajen en múltiples objetos al mismo tiempo, reduciendo la posibilidad de duplicar funcionalidades. Además, dividir problemas en partes más pequeñas facilita las pruebas independientes de cada módulo.

Por último, en la programación orientada a objetos es común crear y compartir librerías, lo que puede ahorrar horas de desarrollo, especialmente en proyectos grandes, contribuyendo así a la reducción de costos.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 23:
Fundamentos de la Programación Orientada a Objetos (POO)



Nota: En la ilustración se puede observar los conceptos básicos de la Programación Orientada a Objetos (POO), detallando un ejemplo de la clase Vehículo que tiene atributos como marca, referencia, modelo, placa, color y valor, y Métodos como frenar, acelerar y girar. Fuente: Elaboración propia.

La programación orientada a objetos (POO) es una forma de diseñar y escribir programas de software utilizando *objetos*, donde los objetos son como bloques de construcción que representan cosas del mundo real o conceptos abstractos en una aplicación.

Es importante el concepto de Clase donde se define que es una plantilla o molde a partir de la cual se crean los objetos. Se puede imaginar a una clase como un plano para construir una casa; cada casa construida a partir de ese plano sería un objeto. Otro concepto importante en la POO es la instanciación que es el proceso de crear un objeto a partir de una clase. Cuando se instancia una clase, se crea una copia de ese plano con valores específicos, como si se construyera una casa con características particulares tales como color, tamaño, etc.

En la POO los objetos tienen atributos, que son los datos que les pertenecen. Por ejemplo, si se tiene una clase llamada Usuario, los atributos podrían incluir el nombre, apellido, correo y contraseña. Estos atributos se definen como las características que describen al objeto y son fundamentales para su funcionamiento.

También los objetos tienen los métodos que son las funciones o acciones que puede realizar un objeto. Siguiendo con el ejemplo de la clase Usuario, algunos de los métodos podrían ser editarPerfil(), iniciarSesion() y cerrarSesion(). Estos métodos representan las capacidades o comportamientos del objeto, permitiendo que interactúe y realice tareas específicas.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Pilares de la programación orientada a Objetos

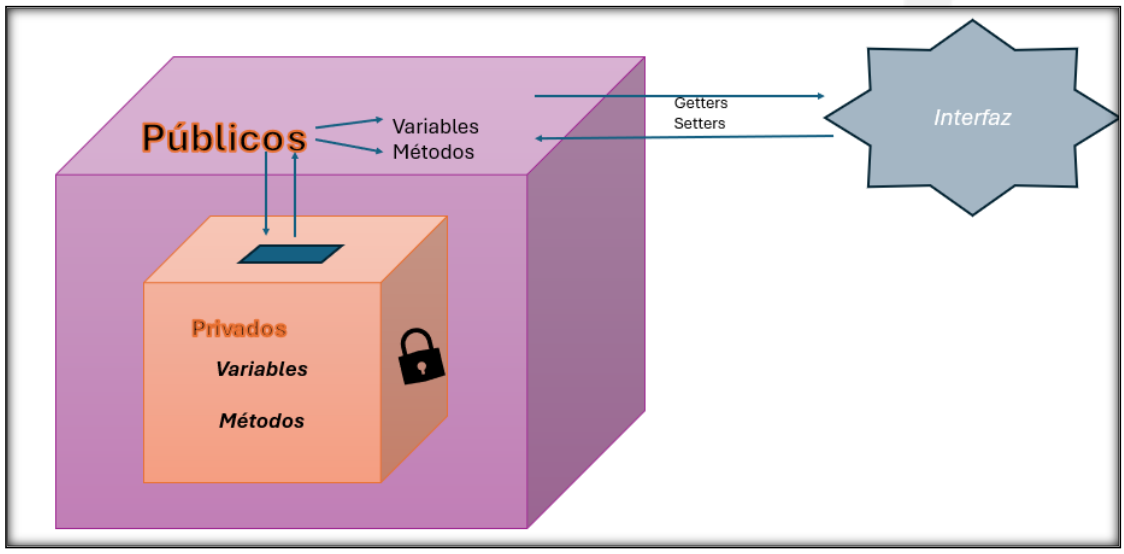
Encapsulamiento

El encapsulamiento es un principio clave de la POO que consiste en agrupar datos con los métodos que los manipulan y restringir el acceso directo a ciertos componentes de un objeto. (Sanchez, 2015). Esto significa que la representación interna de un objeto está oculta al exterior, ofreciendo una interfaz pública para interactuar con él y protegiendo los detalles de su implementación interna.

Se puede decir que el encapsulamiento es una técnica de ocultar los detalles internos o de implementación de un objeto, protegiendo los datos y las funciones internas del acceso externo no autorizado, esta característica ayuda a reducir la complejidad y aumentar la reusabilidad del código, permitiendo que los objetos oculten sus estados internos y expongan solo métodos de acceso definidos.

Se puede afirmar que algunos métodos o atributos de un objeto pueden no ser visibles desde el exterior. En la POO, aquellos atributos o métodos que no pueden ser accedidos desde fuera del objeto se denominan "privados", mientras que los que sí son accesibles se llaman "públicos". Es importante mencionar que, aunque un atributo esté encapsulado, se puede leer o modificar su valor a través de *Getters* y *Setters*.

Ilustración 24:
Visualización del Encapsulamiento en Programación Orientada a Objetos



Nota: En la ilustración se describe el concepto de encapsulamiento en la programación orientada a objetos.
Fuente: Elaboración propia.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

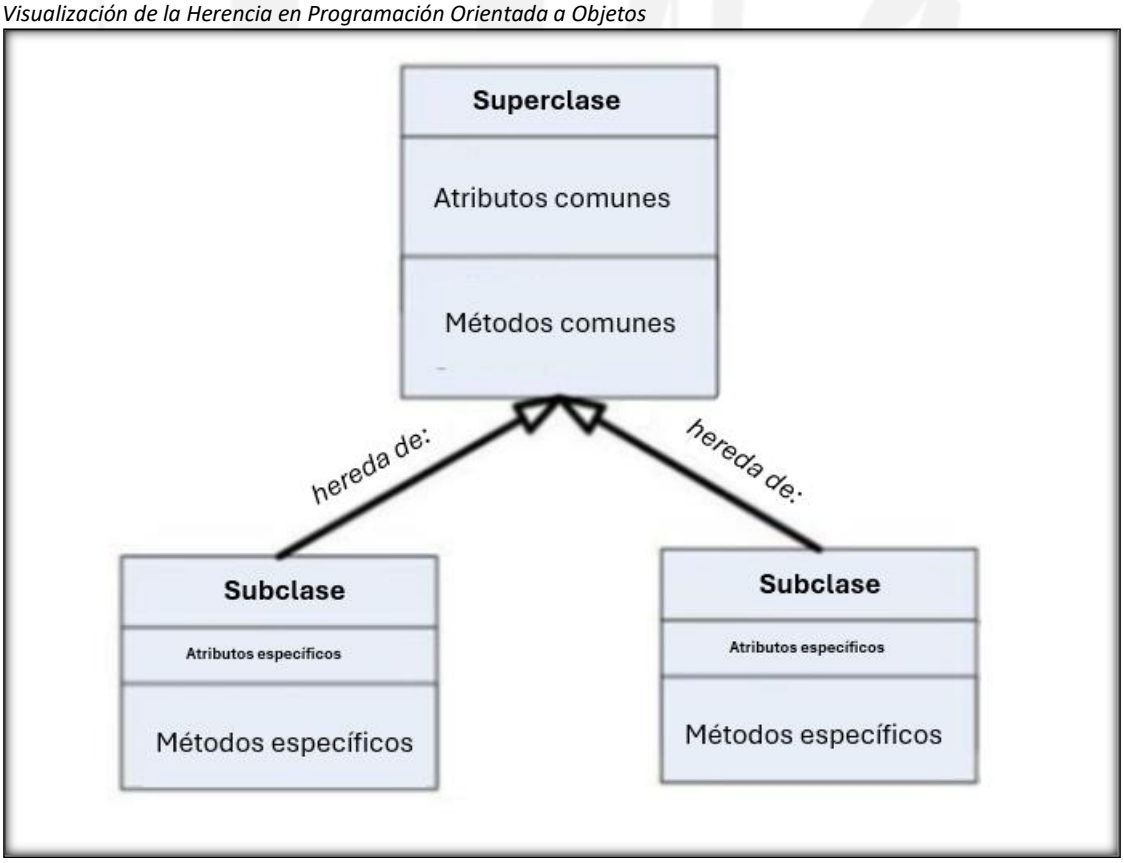
Herencia

Es un mecanismo que permite compartir y reutilizar código ya existente. Esto se logra permitiendo que nuevas clases adopten características de clases ya existentes. Según (Booch, 2007) la clase que hereda se llama subclase o clase derivada, mientras que la clase de la que se hereda se llama superclase o clase base. Este concepto permite que una clase, conocida como clase derivada, herede atributos y métodos de otra clase, conocida como clase base. Existen dos tipos de herencia:

- Herencia Simple: Que se produce cuando una clase derivada hereda de una sola clase base.
- Herencia Múltiple: Que se produce cuando una clase derivada hereda de más de una clase base.

Se puede decir que la herencia permite a una nueva clase adoptar las propiedades y métodos de una clase existente y facilita la reutilización del código y establece una relación jerárquica entre clases, permitiendo la creación de clases más especializadas sobre bases generales.

Ilustración 25:
Visualización de la Herencia en Programación Orientada a Objetos



Nota: En la ilustración se describe el concepto de herencia en la programación orientada a objetos. Fuente: Elaboración propia.

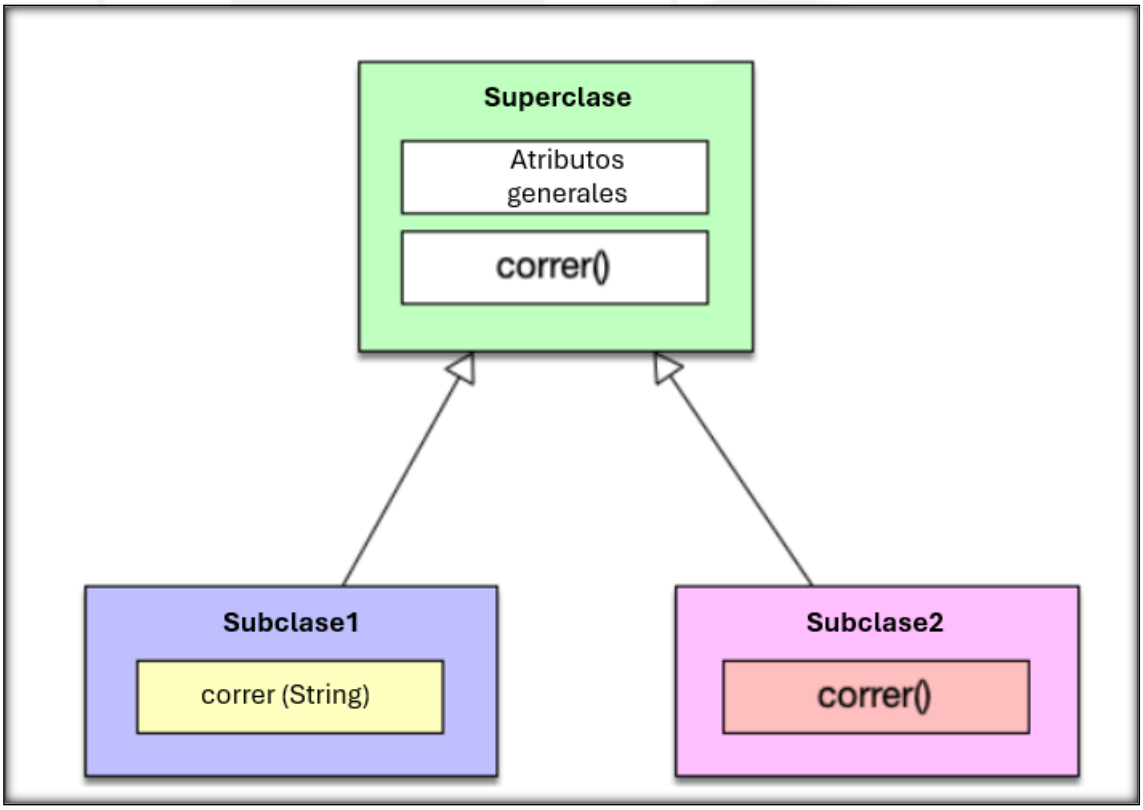
(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Polimorfismo

Es la capacidad de un objeto para asumir diferentes formas. Más técnicamente, Según (Stroustrup, 1994) es la habilidad de diferentes clases de objetos para responder a la misma interfaz, o el mismo conjunto de mensajes, de maneras que son específicas a cada clase. Esta característica permite que se escriba código más general y flexible.

Se puede decir que el polimorfismo permite a las clases derivadas definir métodos que tienen el mismo nombre que los métodos en la clase base, pero con comportamientos distintos, lo que mejora la flexibilidad y la interacción entre objetos, permitiendo que se traten los objetos derivados como objetos de su clase base, facilitando así la generalización de funciones.

Ilustración 26:
Visualización del Polimorfismo en Programación Orientada a Objetos



Nota: En la ilustración se describe el concepto de polimorfismo en la programación orientada a objetos. Fuente: Arquitectura Java. *Java, polimorfismo, herencia y simplicidad*. Arquitectura Java. Recuperado de https://bit.ly/48Ws8fx

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Abstracción

Según (Booch, 2007) la abstracción se refiere al proceso de ocultar los detalles complejos de implementación detrás de una interfaz simplificada. Permite a los desarrolladores concentrarse en las interacciones a un nivel más alto, ignorando los detalles de las operaciones de bajo nivel que ocurren en el fondo.

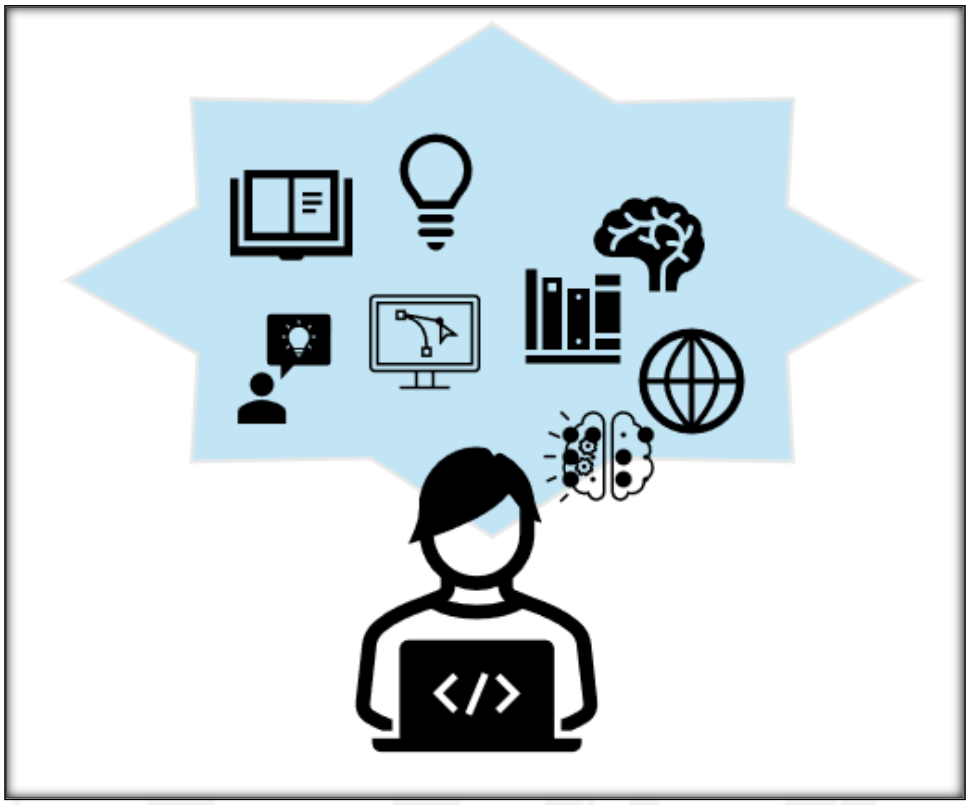
Existen dos tipos de abstracción:

- A**bstracción de Datos**: Que se enfoca en identificar conjuntos de datos esenciales y las operaciones que se pueden realizar sobre estos datos. En muchos lenguajes de programación, esto se implementa a través de clases.
- **Abstracción Procesal**: Que se refiere a encapsular las operaciones complejas en métodos que proporcionan funcionalidades sin revelar su implementación interna.

Se puede decir que la abstracción se enfoca en identificar y separar los aspectos esenciales de una entidad, definiendo los atributos y métodos de una clase, ignorando las características menos importantes o accidentales y permite a los programadores centrarse en 'qué' hace un objeto en lugar de 'cómo' lo hace, facilitando así el manejo de sistemas complejos y la reducción de la interdependencia.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 27:
Visualización de la Abstracción en Programación Orientada a Objetos



Nota: En la ilustración se describe el concepto de abstracción donde se definen, identifican y separan los aspectos esenciales de una entidad. Fuente: Elaboración propia.

Estos cuatro pilares son esenciales para el desarrollo de software robusto y eficiente bajo el paradigma de la POO y proporcionan un marco que no solo mejora la calidad del software, sino que también facilita la gestión de proyectos de software más grandes y complejos.

Introducción a Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems en 1995. La plataforma Java se diseñó con la filosofía de WORA ("write once, run anywhere"), lo que significa que el código Java compilado (bytecode) puede ejecutarse en cualquier dispositivo que disponga de una Máquina Virtual Java (JVM) sin necesidad de recopilación. (Vidal, 2014).

Entre las características mas relevantes de Java se puede citar:

- Orientado a Objetos: Java es estrictamente orientado a objetos, lo que facilita la creación de programas modulares y reutilizables.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Independiente de la Plataforma: Gracias a la JVM, Java es independiente de la plataforma, permitiendo la portabilidad entre diferentes sistemas operativos.
- Seguridad: Incorpora características que facilitan la escritura de programas seguros, incluyendo la eliminación de punteros y la gestión automática de memoria.
- Multihilo: Soporta la programación multihilo nativamente, permitiendo el desarrollo de aplicaciones con múltiples hilos de ejecución.

Arquitectura de la Plataforma Java

La plataforma Java se compone de varias tecnologías y componentes, que incluyen:

- **Java Development Kit (JDK)**: El kit de desarrollo que incluye el compilador Java y herramientas de desarrollo.
- **Java Runtime Environment (JRE)**: El entorno de ejecución que incluye la Máquina Virtual Java (JVM), bibliotecas de clases y otros componentes necesarios para ejecutar aplicaciones Java.
- **Java Virtual Machine (JVM)**: Una "máquina dentro de una máquina" que ejecuta el bytecode Java en una plataforma específica.

Java es una opción popular y poderosa para el desarrollo de software en diversas industrias. Su **robustez**, **portabilidad** y **enfoque orientado a objetos** lo convierten en una herramienta ideal para enseñar a los nuevos programadores los fundamentos de la programación. Aprender Java no solo abre las puertas a múltiples oportunidades en el ámbito tecnológico, sino que también proporciona una base sólida para adentrarse en otros lenguajes de programación y tecnologías.

IDEs y Frameworks

El uso de Entornos de Desarrollo Integrados (IDEs) y frameworks es esencial para facilitar el aprendizaje y la implementación de conceptos teóricos, con un enfoque particular en Java por temas de estudio de esta asignatura se utilizará el IDE NetBeans 21.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Los IDEs o también llamados **Entornos de Desarrollo Integrados (IDEs)** son aplicaciones de software que proporcionan herramientas comprehensivas para programadores para el desarrollo de software. Según (Harold, 2008) un IDE típicamente incluye:

- **Editor de texto**: Para escribir y editar código con características como resaltado de sintaxis y autocompletado.
- **Compilador y/o intérprete**: Para transformar el código escrito en un programa ejecutable o para ejecutarlo directamente.
- **Depurador**: Para probar y depurar el código, permitiendo a los desarrolladores seguir la ejecución y examinar variables.
- **Gestor de proyectos**: Para manejar varios aspectos del proyecto de software, incluyendo la construcción, dependencias, y configuración de versiones.

Aquí hay varios ejemplos de IDEs populares que son ampliamente utilizados en diferentes lenguajes de programación:

- **Eclipse**: Es un IDE muy utilizado para Java, aunque soporta otros lenguajes como C, C++, PHP, y JavaScript mediante plugins, ofrece una gran cantidad de plugins y configuraciones, lo que lo hace extremadamente versátil.

- **Visual Studio**: Es un IDE predominantemente utilizado para desarrollo en C#, VB.NET, y C++, pero soporta muchos otros lenguajes a través de extensiones, se integra con Microsoft Azure, herramientas de diagnóstico avanzadas y posee un potente depurador.

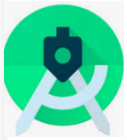
- **IntelliJ IDEA**: Principalmente utilizado para desarrollo en Java, Kotlin, y Scala, es muy reconocido por su eficiencia en la autocompletación de código y refactoring, además de su soporte para desarrollo web y móvil.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- **NetBeans:** Es muy utilizado para Java, PHP, JavaScript, HTML5, entre otros, incluye un entorno de desarrollo robusto con un conjunto de herramientas para GUI de escritorio, aplicaciones web y móviles, para temas de estudio de esta materia se sugiere trabajar con este IDE.

- **PyCharm**: IDE específico para Python, aunque también soporta frameworks de desarrollo web como Django, ofrece análisis de código, un depurador gráfico y herramientas para desarrollo científico.

- **Xcode**: Es un IDE oficial para el desarrollo de software para macOS, iOS, watchOS y tvOS, se integra con todos los frameworks de Apple, simulador para dispositivos Apple y herramientas de perfil.



- **Android Studio:** Específico para el desarrollo de aplicaciones Android, posee un emulador integrado, editor de layout visual, soporte para programación en Kotlin y herramientas específicas para el desarrollo Android.

- **Atom:** Es un IDE ligero que puede ser utilizado para programar en una multitud de lenguajes como JavaScript, HTML, CSS, PHP, Python, y más, es altamente personalizable y extensible con plugins, además de integrarse con Git y GitHub directamente.

Existen en el merado muchos más pero cada uno de estos IDEs ofrece herramientas y características únicas que pueden adaptarse a diferentes estilos de desarrollo y necesidades de proyecto. Elegir el IDE correcto puede depender del lenguaje de programación utilizado, la complejidad del proyecto y las preferencias personales del desarrollador.

Los **Frameworks**, en el contexto de programación, son conjuntos de herramientas y bibliotecas estandarizadas que proporcionan una forma estructurada de desarrollar aplicaciones, facilitan

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

patrones de diseño comunes y conectividad con bases de datos, y manejan aspectos del software como la seguridad y la gestión de sesiones.Típicamente, un framework incluye:

- **Bibliotecas de código**: Son el conjunto de herramientas y funciones predefinidas para tareas comunes.
- **Estructuras de control de flujo**: Son los patrones predefinidos para la arquitectura del software, como MVC (Modelo-Vista-Controlador).
- **Hooks y callbacks**: Son los puntos de integración para que los desarrolladores enganchen su propio código sin modificar el framework mismo.

Existen algunos Ffameworks en el mercado, dependiendo del desarrollo, por ejemplo:

- Desarrollo web: Frameworks como Django (Python), Laravel (PHP), Rails (Ruby), y Spring (Java), Angular, ofrecen estructuras robustas para la construcción de aplicaciones web.
- Desarrollo de aplicaciones móviles: Frameworks como React Native y Flutter permiten el desarrollo de aplicaciones nativas utilizando enfoques basados en JavaScript y Dart, respectivamente, existe también Xamarin, Ionic, entre otros.

Para temas de estudio de esta asignatura se analizará el IDE NetBeans 21 que es uno de los más populares para el desarrollo en Java, conocido por su interfaz intuitiva y su capacidad para simplificar el desarrollo de aplicaciones Java estándar, Java EE, y JavaFX. Ofrece características como autocompletado de código, plantillas de código, y herramientas gráficas para el diseño de aplicaciones, facilitando a los principiantes la comprensión y aplicación de conceptos de programación.

Requisitos del Sistema: Para instalar NetBeans 21, es necesario contar con una computadora que cumpla con los siguientes requisitos mínimos:

Tabla 2:
Requisitos del sistema

Símbolo Operativo	Windows, macOS, o Linux
JDK (Java Development Kit)	Tener instalado Java JDK 8 o superior.
Memoria RAM	Mínimo 2GB, recomendado 4GB o más.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Espacio en disco	Al menos 750MB libres para instalación

Nota: En la tabla se visualiza los requisitos imnimos que debe tener el sistema para instalar NetBeans 21.

Pasos para instalar NetBeans:

- Descarga del Instalador, desde la página oficial de NetBeans Descargar NetBeans y descargar la versión más reciente en este caso de NetBeans 21.

- Instalación del JDK: Si aún no está instalado en el dispositivo que se desea instalar, descargar e instalar el JDK desde el sitio web de Oracle, escogiendo el sistema operativo que disponga, del siguiente enlace: Descargar JDK

- Ejecutar el Instalador de NetBeans: Abrir el archivo descargado y seguir las instrucciones en pantalla para completar la instalación.

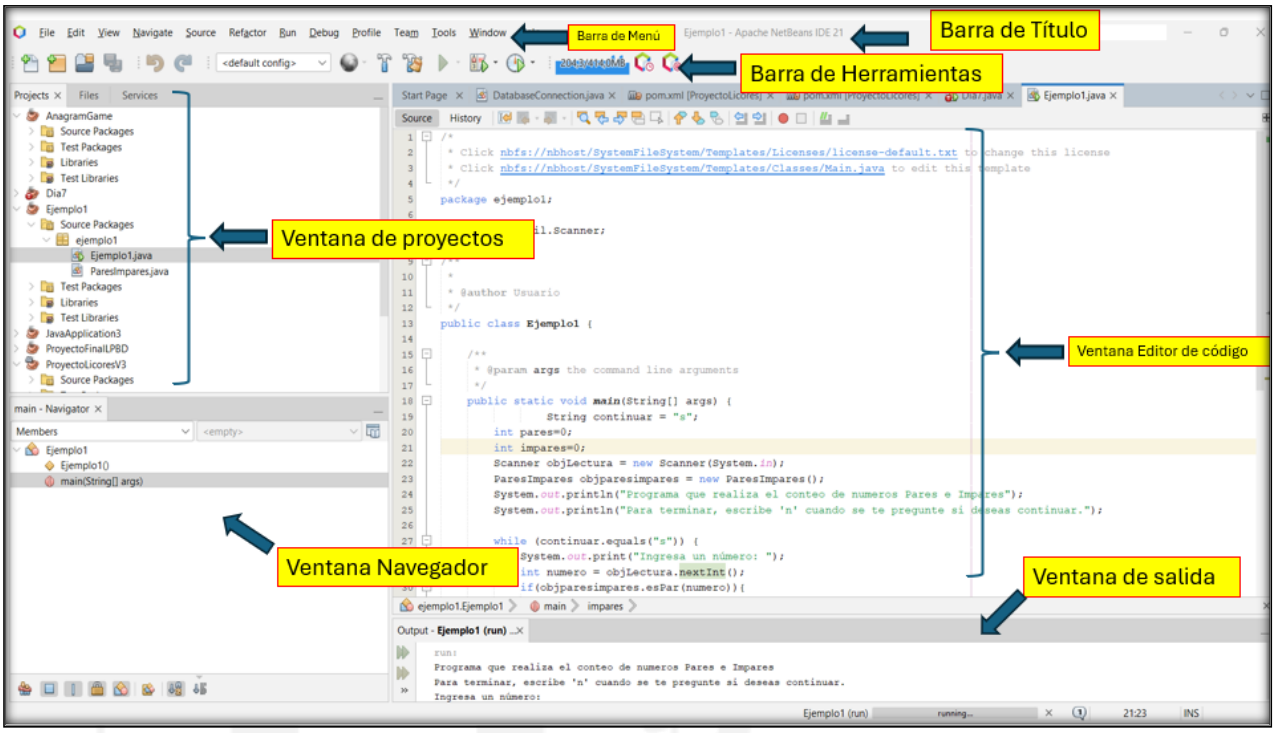
- Configuración del JDK: Durante la instalación, seleccionar el JDK instalado cuando se solicite configurar el JDK.

Explorando NetBeans

Una vez instalado, al abrir NetBeans. La primera pantalla que se observa es el entorno de trabajo principal, que incluye la barra de menú, la barra de herramientas, el explorador de proyectos, el editor de código y la consola de salida.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 28:
Visualización de la pantalla principal de NetBeans



Nota: En la ilustración se visualiza la pantalla principal del IDE NetBeans con sus elementos principales. Fuente: Elaboración propia.

Creación de un Proyecto Nuevo

1. **Abrir NetBeans**: Iniciar el programa desde el menú de aplicaciones o el escritorio.

2. **Crear un Proyecto Nuevo**: Ir a File > New Project. Elegir Java with Ant y Java Application, luego clic en Next.

3. **Configurar el Proyecto**: Asignar un nombre al proyecto y especificar la ubicación del mismo. Asegurarse de que el JDK correcto esté seleccionado, luego clic en Finish.

Pasos para la creación de un proyecto

NetBeans es un IDE gratuito y de código abierto que soporta múltiples lenguajes de programación, pero está particularmente optimizado para Java. Facilita la gestión de proyectos, desde la creación hasta la depuración y el despliegue.

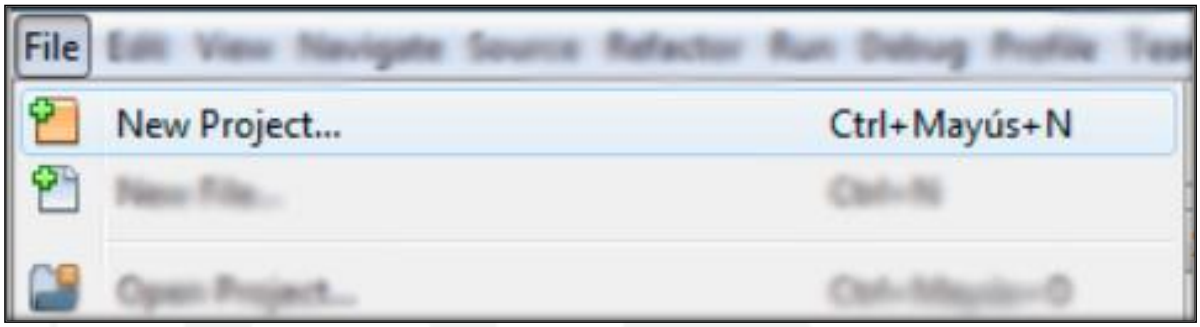
Una vez haya iniciado correctamente Netbeans, se puede crear un nuevo proyecto.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Para crear un nuevo proyecto, solo se necsita seguir unos pasos sencillos. En la parte superior de nuestra ventana principal, existen dos opcopnes.

Opcion1: Dar clic en: **Archivo » Nuevo Proyecto...**, tal como se muestra en la siguiente ilustración.

Ilustración 29:
Icono para la creación de nuevo proyecto



Nota: En la ilustración se visualiza las opciones para crear un nuevo proyecto en NetBeands. Fuente: Elaboración propia.

Opcion2: Dar clic en:Icono de nuevo proyecto de la barra de herramientas del IDE, tal como muestra la siguiente ilustración.

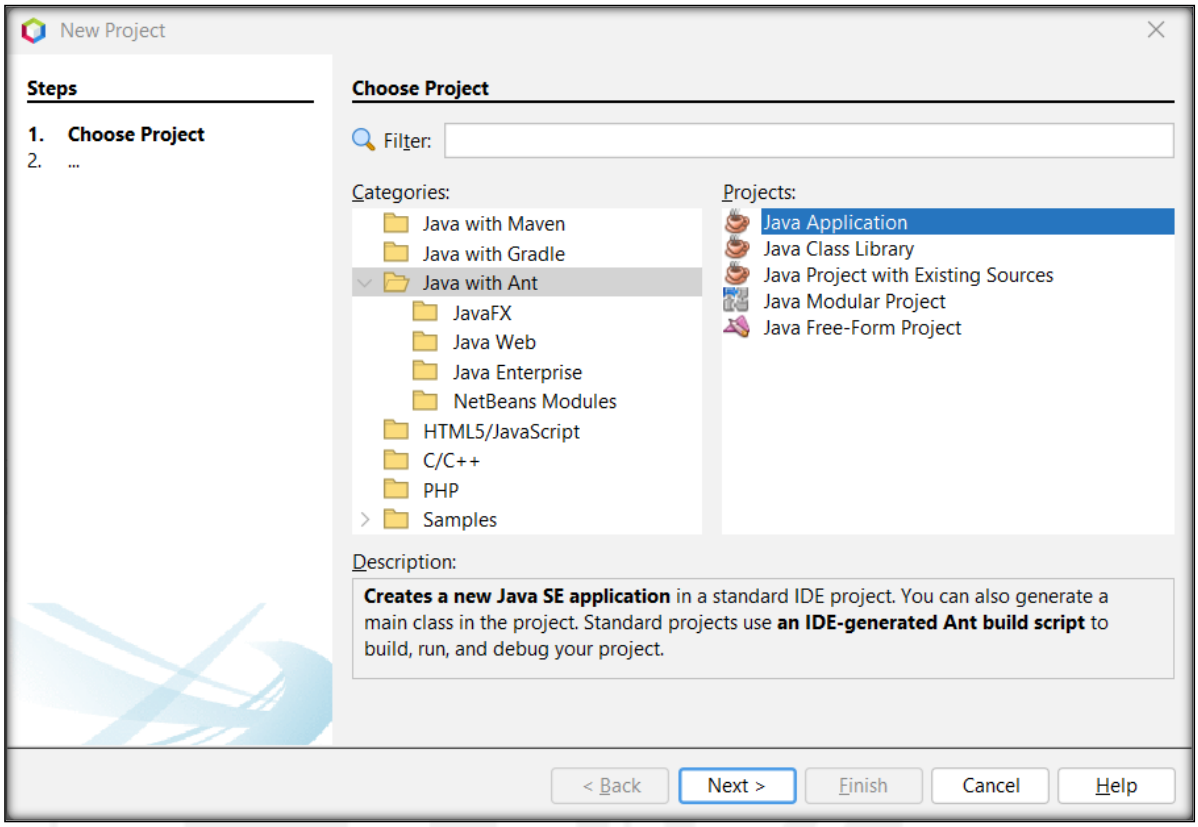
Ilustración 30:
Icono para Crear nuevo proyecto en la Barra de herramientas de NetBeans

Nota: En la ilustración se visualiza el icono de crear un nuevo proyecto en NetBeans en la barra de herramientas Fuente: Elaboración propia.

Se abre una nueva ventana donde se puede elegir el tipo de proyecto que se desea crear. Se tiene varias opciones disponibles, y de ellas para este tema de estudio se selecciona **Java with Ant » Java Application**. Luegose se da clic en **Next >** para continuar, tal como muestra la siguiente ilustración:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 31:
Asistente para la creación de un nuevo proyecto en NetBeans

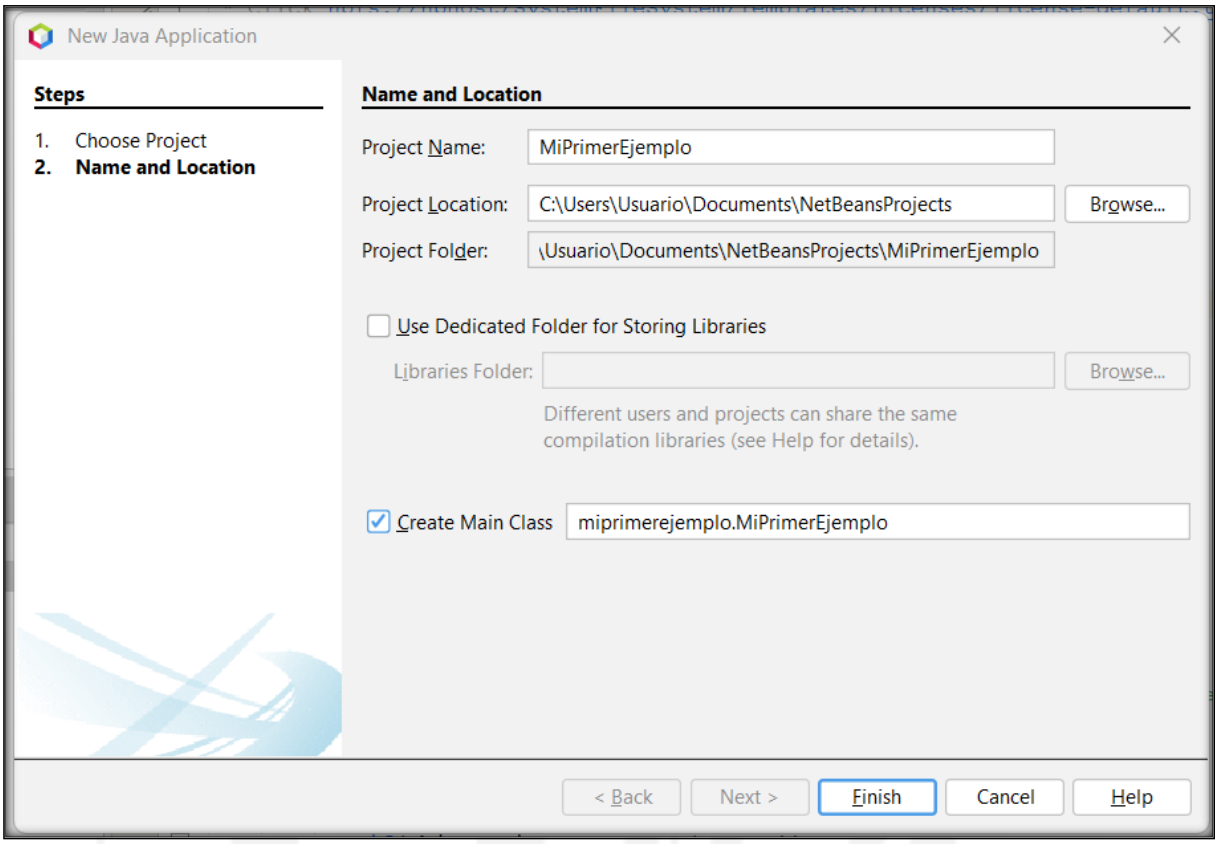


Nota: En la ilustración se visualiza las opciones del Asistente para la creación de un nuevo proyecto en NetBeans. Fuente: Elaboración propia.

A continuación, se abre una ventana donde se llena con información sobre el proyecto como Nombre del proyecto, ubicación donde se lozaliza localmente, si se puede cambiar la ubicación del proyecto haciendo clic en **Examinar**, por defecto se guarda en la carpeta **NetBeansProjects** que se crea al momento de instalar Netbeans y clic en **Create Main Class.** Finalmente, se da clic en **Finish**

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 32:
Asistente – Datos del nuevo proyecto



Nota: En la ilustración se visualiza las opciones para crear un nuevo proyecto en NetBeands donde se escoge el nombre la localizacon del proyecto. Fuente: Elaboración propia.

De esta manera se ha creado un nuevo proyecto para desarrollar en Java.

POO en java

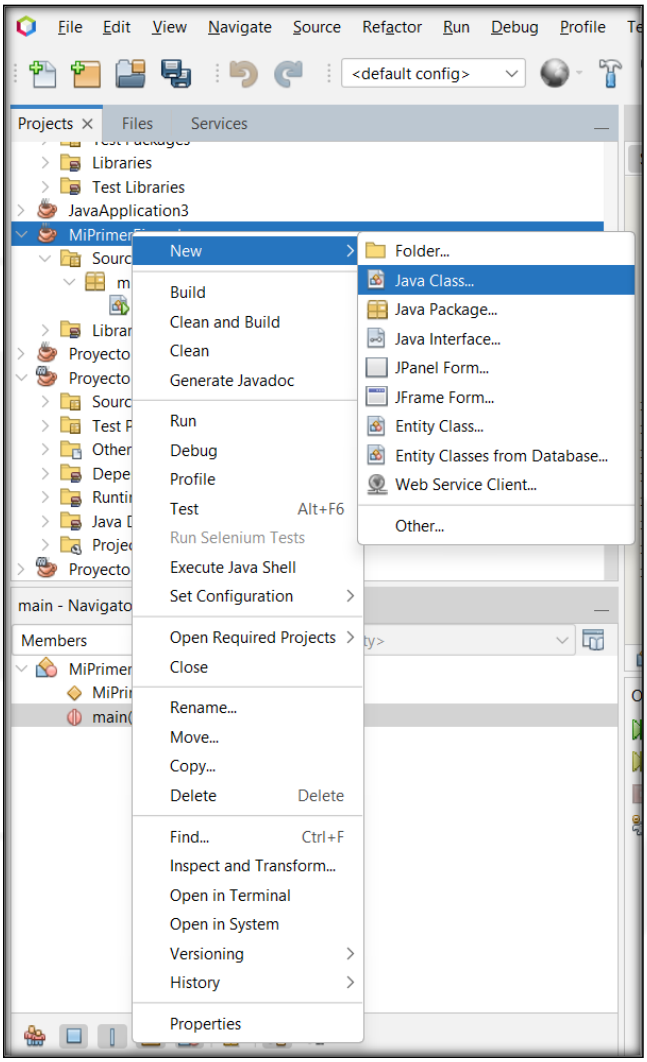
Clases

En Java, una clase se puede entender como un **prototipo** que define las variables y métodos comunes a un grupo específico de instancias. Las clases son la base a partir de la cual podemos crear múltiples instancias del mismo tipo. Antes de poder crear instancias de una clase, primero se debe definirla.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

En NetBeans se puede añadir elementos al proyecto, clases, interfaces, etc. En esta ocasión se va a crear una clase al interior del proyecto, para esto se da clic derecho sobre el proyecto, luego en New y finalmente en Java Class.

Ilustración 33:
Creación del nuevo proyecto



Nota: En la ilustración se visualiza la manera de crear clases en un proyecto en NetBeans. Fuente: Elaboración propia.

Aparecerá una ventana donde se coloca el nombre para la clase, el proyecto y el paquete, donde se encontrará la nueva clase de java.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Objetos

En Java, un **objeto** es simplemente una instancia de una clase. Lo más relevante de los objetos en Java es que nos permiten controlar quién puede acceder a sus miembros. Esto significa que un objeto puede tener miembros públicos, accesibles por otros objetos, o miembros privados, a los que solo él puede acceder. Estos miembros pueden ser tanto variables como funciones, y así se permite que el código de un objeto se mantenga de forma independiente de otros objetos en la aplicación, lo que facilita su mantenimiento y comprensión.

Estructura de un Programa Básico en Java

Un programa en Java sigue una estructura bien definida, y cuando se utiliza NetBeans, el entorno de desarrollo integrado (IDE) simplifica el proceso de creación y organización de esta estructura. A continuación, se detalla la estructura básica de un programa en Java utilizando NetBeans:

Paquete (package)

En NetBeans, los paquetes son creados automáticamente cuando se define un nuevo proyecto. El paquete organiza las clases relacionadas dentro de un mismo espacio de nombres. Ejemplo:

```
package miPrimerPrograma;
```

Importaciones (import)

NetBeans facilita la importación de clases y paquetes necesarios para el programa, sugiriendo importaciones automáticas mientras se escribe el código. Ejemplo:

```
import java.util.Scanner;
```

Declaración de la Clase

La clase principal del programa es automáticamente generada por NetBeans. El nombre de la clase generalmente coincide con el nombre del archivo.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ejemplo:

```
public class HolaMundo {
```

Método Principal (main)

NetBeans también genera el método **main**, que es el punto de entrada de cualquier programa Java.

Ejemplo:

```
public static void main(String[] args) {
```

Instrucciones o Código

Dentro del método **main**, se escribe el código que define las operaciones del programa. Ejemplo:

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

Cierre de la Clase

El código de la clase se cierra con una llave, concluyendo la definición de la clase. Ejemplo:

```
}
```

Comentarios en Java

Los **comentarios** en Java, al igual que en otros lenguajes de programación, son herramientas valiosas para documentar el código. Ayudan a que otras personas o incluso el mismo desarrollador comprendan mejor el funcionamiento del código. Los comentarios son líneas de código que el compilador ignora al ejecutar la aplicación, lo que significa que no están sujetos a restricciones de sintaxis. El uso principal de los comentarios es organizar el código y hacerlo más comprensible, especialmente cuando otras personas deben leerlo.

Comentarios de una sola línea: En Java se pueden colocar en cualquier parte del código. Se inician con //. Al agregar este símbolo en una línea, todo lo que esté a la derecha de estos en esa misma línea será considerado un comentario. Es importante destacar que las líneas que siguen no se verán afectadas. En resumen, este símbolo // solo impacta la línea donde se coloca.

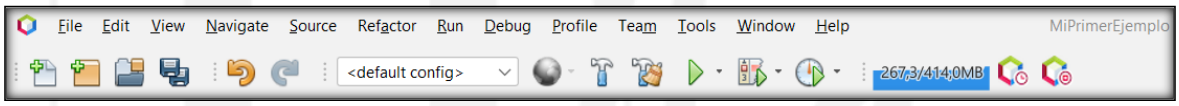
Comentarios de múltiples líneas: En Java tal como el nombre lo indica permite comentar varias líneas del código Java de manera mucho más sencilla en vez de esta añadiendo // a cada línea.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Estos comentarios van cerrados entre /* y */, es decir comienzan donde se ponga /* y terminan donde esté el */. Estos comentarios funcionan de manera similar a los comentarios de una sola línea, pero deben tener un comienzo y un final. A diferencia de los comentarios de una sola línea, al poner el símbolo "/*" todo el código que haya tanto en la misma línea, como en las líneas posteriores de este se convertirán en comentarios hasta que encuentre el */, de manera que si se inicia un comentario de múltiples líneas, se debe cerrarlo, tal como sucede con las llaves o los corchetes en Java.

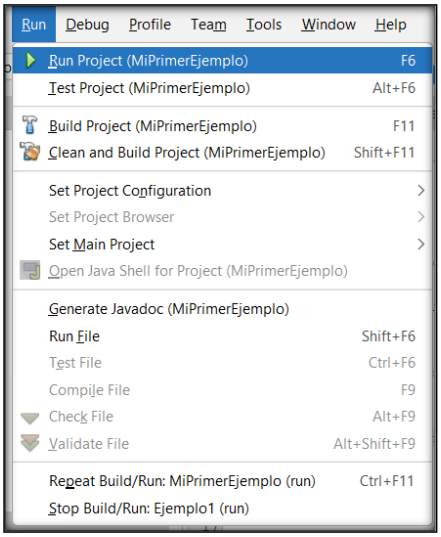
El entorno de desarrollo integrado facilita el proceso de compilación y ejecución de programas Java. NetBeans proporciona herramientas gráficas para escribir código, compilarlo y ejecutarlo con solo unos clics. Simplemente el desarrollador se enfoca en escribir el código en el editor del IDE y utilizar las opciones de 'Run' o 'Compile' para compilar el código.

Ilustración 34:
Barra de herramientas – icono RUN



Nota: En la ilustración se visualiza el icono que sirve para correr y compilar el programa en Java desde NetBeans. Fuente: Elaboración propia.

Ilustración 35:
Compiilación y Ejecución



Nota: En la ilustración se visualiza como se compila y ejecuta un programa en Java desde NetBeands. Fuente: Elaboración propia.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

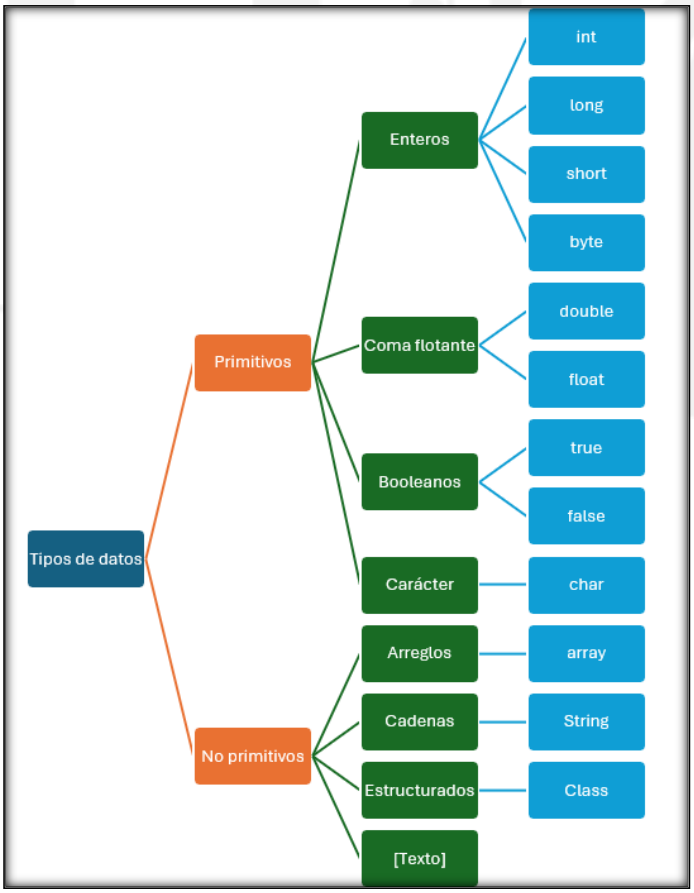
En Java, los tipos de datos se dividen en dos categorías principales: tipos de datos primitivos y tipos de datos no primitivos.

Tipos de datos

Comprender estos tipos de datos es fundamental para escribir programas eficientes y eficaces. Para temas de estudio de esta asignatura se usará Java, como lenguaje de programación de alto nivel, y este lenguaje de programación ofrece varios tipos de datos que permiten manejar diferentes tipos de información de manera adecuada. En Java, los tipos de datos se clasifican en dos categorías principales: tipos de datos primitivos y tipos de datos no primitivos..

- **Datos Primitivos:** Los datos primitivos son los tipos de datos básicos que ofrece Java. Estos incluyen: enteros, números de coma flotante, caracteres y valores booleanos. Java define ocho tipos de datos primitivos.
- **Datos No Primitivos:** Java también proporciona tipos de datos no primitivos. Estos incluyen cadenas de caracteres, arreglos y clases.

Ilustración 36:
Tipos de datos en java



Nota: En la ilustración se puede observar un mapa conceptual de los tipos de datos en Java, clasificados en dos categorías principales: primitivos y no primitivos. Fuente: Elaboración propia.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Diversas funcionalidades en Java

Java es un lenguaje de programación muy conocido por su versatilidad y facilidad de uso, lo que lo convierte en una excelente opción para desarrollar aplicaciones robustas y eficientes. Entre sus características más importantes se encuentran la capacidad de interactuar con el usuario a través de la entrada desde el teclado y la salida a la consola, así como el manejo de archivos. Por ejemplo, los programadores pueden utilizar la clase Scanner para capturar datos del usuario y la clase System.out para mostrar mensajes y resultados. También Java permite crear, leer y modificar archivos mediante clases como File y BufferedReader, lo que facilita la gestión de datos. La flexibilidad, junto con su amplia biblioteca de clases, hace que Java sea una herramienta útil en el mundo del desarrollo de software. En un entorno donde la manipulación de datos es crucial, Java se posiciona como uno de los lenguajes que no solo mejora las habilidades técnicas de los programadores, sino que también fomenta un enfoque analítico y crítico en la creación de soluciones efectivas.

Entrada desde el Teclado

En Java, la entrada desde el teclado se maneja utilizando la clase *Scanner*. Esta clase, parte del paquete *java.util*, que permite la lectura de datos desde diversas fuentes, incluyendo la entrada estándar por teclado. Con la clase *Scanner*, es posible capturar diferentes tipos de datos como enteros, cadenas de texto, y más que son ingresados por el usuario.

Uso de la Clase Scanner

Para utilizar la clase Scanner, se debe crear una instancia de la clase y emplear sus métodos para leer datos del usuario:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 37:
Uso de la clase Scanner

```java
package miprimerejemplo;

import java.util.Scanner;

public class MiPrimerEjemplo {


    public static void main(String[] args) {    //Se declara el método main, que es el punto de entrada de cualquier aplicación Java.
        Scanner scanner = new Scanner(System.in);   // Se crea un objeto de la clase Scanner para leer la entrada del teclado
        System.out.print("Ingrese su nombre: "); // Solicita al usuario que ingrese su nombre
        String nombre = scanner.nextLine();  // Se declara y asigna el valor ingresado en el teclado en la variable nombre
        System.out.println("Hola, " + nombre); // Imprime un saludo seguido del nombre ingresado por el usuario
    }

}
```

Nota: En la ilustración se puede observar el código usando la clase Scanner que captura una cadena de texto ingresada por el usuario y la imprime en la consola. Fuente: Elaboración propia.

Entre los principales métodos que tiene imp0lementada esta clase son:

- nextInt(): Lee el siguiente entero disponible.
- nextDouble(): Lee el siguiente número en punto flotante.
- nextLine(): Lee una línea completa de entrada, incluyendo espacios.
- next(): Lee el siguiente token (cadena sin espacios) del flujo de entrada.
- nextBoolean(): Lee el siguiente valor booleano (true o false).
- nextFloat(): Lee el siguiente número de punto flotante de precisión simple.

Al usar esta clase es importante considerar que se debe tener en cuenta estas sugerencias al momento de programar:

- Siempre se debe cerrar el objeto Scanner después de su uso para liberar los recursos.
- Implementar controles y manejo de excepciones para evitar que entradas no válidas detengan el programa.
- Al usar métodos como nextInt() y nextLine() en secuencia, manejar correctamente los saltos de línea pendientes para evitar errores de lectura.

Salida a la Consola

La salida a la consola en Java se realiza mediante los métodos *System.out.print()* y *System.out.println()*.

En Java, la salida en consola es una de las formas más fundamentales de mostrar información al usuario. Permite a los programadores enviar mensajes, resultados de cálculos, o cualquier otra

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

información que se necesite presentar al usuario durante la ejecución del programa. Para la salida en consola, Java utiliza la clase *System*, que es parte del paquete *java.lang*. Dentro de esta clase, se utilizan los métodos *System.out.print(), System.out.println(),* y *System.out.printf()* para imprimir datos en la consola.

Uso de System.out.print() y System.out.println()

Ejemplo de uso de estos métodos:

Ilustración 38:
Uso de de System.out.print() y System.out.println()

```
1  package miprimerejemplo;   // Se declara el paquete al que pertenece esta clase, en este caso, "miprimerejemplo".
2
3  public class MiPrimerEjemplo { // Se define la clase pública "MiPrimerEjemplo". El nombre del archivo debe coincidir con el nombre de esta clase.
4      public static void main(String[] args) {     //Se declara el método main, que es el punto de entrada de cualquier aplicación Java.
5          System.out.print("Hola, ");  // Imprime "Hola, " en la consola sin un salto de línea al final.
6          System.out.println("Mundo!");  // Imprime "Mundo!" en la consola y añade un salto de línea al final.
7      }
8  }
```

Nota: En la ilustración se puede observar el código del programa que imprime `Hola, ` sin un salto de línea, seguido de `Mundo!` con un salto de línea. Fuente: Elaboración propia.

Entre los principales métodos que tiene imp0lementada esta clase son:

- System.out.print(): Este método imprime texto en la consola sin agregar un salto de línea al final. Cualquier texto posterior que se imprima se mostrará en la misma línea.
- System.out.println(): Este método es similar a *print(),* pero agrega un salto de línea al final del texto impreso. Es útil para imprimir texto en líneas separadas o para finalizar una línea de salida antes de empezar la siguiente.
- System.out.printf(): Este método permite una salida formateada en la consola, similar a la función *printf* en el lenguaje C. Utiliza especificadores de formato para dar salida a variables y texto de manera estructurada.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Manipulación de Archivos

El manejo de archivos es una habilidad fundamental en programación, ya que permite a los programas almacenar, leer, escribir y manipular datos de manera persistente. En Java, el manejo de archivos se realiza a través de clases que forman parte del paquete java.io y, en versiones más recientes, el paquete *java.nio.file*. Estas clases proporcionan métodos para realizar operaciones comunes de entrada y salida (I/O) con archivos.

Entre las principales clases que se utilizan para el manejo de archivos en Java son:

- File: Esta clase proporciona métodos para realizar operaciones sobre los archivos y directorios del sistema de archivos, como comprobar si un archivo existe, crear nuevos archivos o directorios, eliminar archivos, y obtener propiedades de un archivo, como su tamaño y ruta.
- FileReader y FileWriter: Se utilizan para leer y escribir archivos de texto de forma simple. *FileReader* lee caracteres de un archivo, mientras que *FileWriter* escribe caracteres en un archivo.
- BufferedReader y BufferedWriter: Estas clases envuelven a *FileReader* y *FileWriter*, proporcionando una forma más eficiente de leer y escribir archivos de texto mediante el uso de un búfer, lo que mejora el rendimiento al reducir el número de operaciones de E/S.
- PrintWriter: Facilita la escritura de archivos con métodos adicionales como *println(), printf(), y format()* para una salida formateada.
- FileInputStream y FileOutputStream: Se utilizan para leer y escribir bytes desde y hacia archivos. Son adecuados para manejar archivos binarios como imágenes, videos o archivos de audio.
- Files (en java.nio.file package): Proporciona métodos para operaciones comunes con archivos, como copiar, mover, eliminar, leer y escribir, así como manipular archivos de manera más moderna y eficiente.

Crear archivos

Para crear un Archivo se usa la clase File que permite crear un nuevo archivo en el sistema.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 39:
Uso de la clase File

```
import java.io.File;
import java.io.IOException;

public class CrearArchivo {
    public static void main(String[] args) {
        File archivo = new File("miArchivo.txt");
        try {
            if (archivo.createNewFile()) {
                System.out.println("Archivo creado: " + archivo.getName());
            } else {
                System.out.println("El archivo ya existe.");
            }
        } catch (IOException e) {
            System.out.println("Ocurrió un error.");
            e.printStackTrace();
        }
    }
}
```

Nota: En la ilustración se puede observar el código del programa que permite crear un archivo llamado miArchivo.txt en la ruta donde se encuentra el proyecto. Fuente: Elaboración propia.

Lectura y Escritura en Archivos

En Java, la escritura en archivos es una operación fundamental que permite guardar datos de manera persistente. Dos clases importantes para realizar estas operaciones son FileWriter y BufferedWriter, las cuales pertenecen al paquete java.io. Estas clases permiten escribir datos en archivos de texto de manera eficiente y controlada.

La clase FileWriter se utiliza para escribir caracteres en archivos de texto. Es una clase de bajo nivel que permite escribir directamente en un archivo, y es adecuada para operaciones simples de escritura. Sin embargo, no es la opción más eficiente cuando se requiere escribir grandes cantidades de datos, debido a que cada operación de escritura accede al disco, lo que puede ser lento.

Para escribir en un archivo, se puede usar `FileWriter` junto con `BufferedWriter` para optimizar la escritura de datos:

La clase BufferedWriter es una clase de escritura de alto nivel que proporciona una forma más eficiente de escribir texto en archivos. Se utiliza junto con FileWriter para almacenar datos en un búfer antes de escribirlos en el archivo. Esto reduce la cantidad de accesos al disco y mejora el rendimiento, especialmente cuando se escriben grandes cantidades de datos.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ilustración 40:
Uso de la clase BufferedWriter y FileWriter)

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class EscribirArchivo {
  public static void main(String[] args) {
    try {
        BufferedWriter escritor = new BufferedWriter(new FileWriter("archivo.txt"));
        escritor.write("Hola, archivo!");
        escritor.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```

Nota: En la ilustración se puede observar el código del programa escribe en el archivo archivo.txt, utilizando BufferedWriter y FileWriter Fuente: Elaboración propia.

La clase FileReader es una clase de bajo nivel diseñada para leer flujos de caracteres de archivos de texto. Proporciona una forma sencilla de leer datos de archivos, caracter por caracter o en bloques de caracteres. FileReader es adecuado para operaciones simples de lectura, pero no es la opción más eficiente cuando se requiere leer grandes cantidades de datos debido a que cada operación de lectura accede al disco.
La clase BufferedReader es una clase de lectura de alto nivel que proporciona una forma más eficiente de leer texto de archivos. Se utiliza junto con FileReader para almacenar datos en un búfer antes de leerlos, lo que reduce la cantidad de accesos al disco y mejora el rendimiento, especialmente cuando se leen grandes cantidades de datos.

Ilustración 41:
Uso de la clase BufferedReade y FileReader

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class LeerArchivo {
    public static void main(String[] args) {
        try {
            BufferedReader lector = new BufferedReader(new FileReader("archivo.txt"));
            String linea;
            while ((linea = lector.readLine()) != null) {
                System.out.println(linea);
            }
            lector.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Nota: En la ilustración se puede observar el código del programa que lee cada línea del archivo `archivo.txt` y la imprime en la consola. Fuente: Elaboración propia.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Autoevaluación 3

Lea detenidamente la pregunta y marque la respuesta correcta entre las opciones dadas.

1. **¿Qué es la Programación Orientada a Objetos (POO)??**

   a) Un lenguaje de programación.

   b) Un paradigma de programación.

   c) Una técnica de depuración de software.

   d) Un tipo de sistema operativo.

2. **¿ Cuál es uno de los principales lenguajes de programación orientado a objetos?**

   a) HTML

   b) CSS

   c) Java

   d) SQL

3. **¿ Cuál es uno de los pilares fundamentales de la POO?**

   a) Herencia

   b) Algoritmo

   c) Pseudocódigo

   d) Memoria

4. **¿Qué permite el encapsulamiento en POO?**

   a) Compartir todos los atributos de un objeto.

   b) Eliminar métodos obsoletos de una clase.

   c) Crear múltiples objetos a partir de una clase.

   d) Esconder los detalles internos de un objeto.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

5. ¿ Qué es una clase en POO?

a) Un archivo de texto que guarda datos.

b) Un tipo de dato primitivo.

c) Una plantilla o molde a partir de la cual se crean objetos.

d) Un comando específico para el sistema operativo.

6. ¿Qué es la instanciación en POO?

a) El proceso de eliminar un objeto.

b) La creación de una copia de un objeto.

c) El proceso de crear un objeto a partir de una clase.

d) La conversión de un dato primitivo a un objeto.

7. ¿Qué pilar de POO permite que un objeto asuma diferentes formas?

a) Encapsulamiento

b) Herencia

c) Abstracción

d) Polimorfismo

8. ¿Qué permite la herencia en POO?

a) Crear una nueva clase a partir de múltiples objetos.

b) Permitir que una nueva clase adopte propiedades y métodos de otra clase.

c) Multiplicar la memoria asignada a una clase.

d) Crear interfaces para clases derivadas con sus propiedades y métodos.

9. ¿ Qué clase en Java se utiliza para leer entradas del teclado?

a) FileReader

b) BufferedReader

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

c) InputStream

d) Scanner

10. ¿ Qué término describe las funciones o acciones que un objeto puede realizar en POO?

a) Métodos

b) Variables

c) Interfaces

d) Atributos

Si ha concluido la autoevaluación:

Verifique sus respuestas en el solucionario que se encuentra al final de la presente Guía.

Si alcanzó un porcentaje alto de respuestas correctas en las preguntas de evaluación, ha finalizado este módulo de Introducción a la Programación Orientada a Objetos caso contrario, vuelva a revisar los temas en los que ha tenido dificultad.

.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Resumen de la Unidad 3

En la Unidad 3 se aborda la Introducción a la programación orientada a objetos, centrándose en que la Programación Orientada a Objetos (POO) es un paradigma de programación que se centra en el uso de objetos como elementos fundamentales para el desarrollo de software. Este enfoque permite crear aplicaciones de manera modular, reutilizable y escalable, lo que es especialmente valioso para estudiantes de tecnología y desarrollo de software. Existen 4 pilares fundamentales de la POO: **Encapsulación**: Este principio se refiere a la práctica de restringir el acceso a ciertos componentes de un objeto, exponiendo solo aquellos métodos y propiedades que son necesarios a través de una interfaz pública. Esto no solo mejora la seguridad del código, sino que también facilita su modularidad. **Herencia**: Permite que una clase (subclase) obtenga los atributos y métodos de otra clase (superclase), promoviendo la reutilización de código y creando relaciones jerárquicas que facilitan la especialización de clases. **Polimorfismo**: Este concepto permite que diferentes objetos respondan de manera única a la misma interfaz. Esto proporciona flexibilidad en el diseño del software, ya que se pueden utilizar diferentes implementaciones de un mismo método. **Abstracción**: Consiste en ocultar los detalles complejos de la implementación y mostrar solo las características esenciales de un objeto. Esto ayuda a simplificar la complejidad del sistema y a enfocarse en lo que realmente importa. Además, se revisa de manera general a los lenguajes de programación destacando que uno de los lenguajes más destacados en el ámbito de la POO es **Java**, conocido por su independencia de plataforma y seguridad. Utilizar un entorno de desarrollo integrado (IDE) como NetBeans que es objeto de estudio de esta materia por su facilidad de uso y permite la creación, depuración y gestión de proyectos en Java, permitiendo a los desarrolladores organizar su código de manera clara y eficiente. Además, Java proporciona clases estándar como **Scanner** para la entrada de datos y **BufferedWriter, FileWriter, BufferReader y FileReader** para la manipulación de archivos.

Esta unidad destaca que la POO no solo ofrece un marco robusto para el desarrollo de software, sino que también es fundamental para abordar los desafíos de programación complejos. Al adoptar este paradigma, los estudiantes de tecnología pueden crear aplicaciones más seguras, eficientes y fáciles de mantener, lo que los prepara para enfrentar las demandas del mundo real.

¡Sigua brillando! ¡Le deseo lo mejor en sus futuros proyectos!

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

9. REFERENCIAS

- Acosta, R. (2009). *Flujograma*: (ed.). El Cid Editor | apuntes. Recuperado el 14 de Noviembre de 2024 de https://elibro.net/es/lc/itq/titulos/28942

- Aristizábal, D. & Quiceno, S. (2023). *Lógica de programación básica orientada a objetos con ejercicios resueltos*: (1 ed.). Instituto Tecnológico Metropolitano. Recuperado el 12 de Agosto de 2024 de https://elibro.net/es/lc/itq/titulos/253798

- Arteaga, M. (2023). *Lógica de programación con Pseint: enfoque práctico: (1 ed.). Corporación Universitaria Remington.* Recuperado el 13 de Noviembre de 2024 de https://elibro.net/es/lc/itq/titulos/250650

- Astudillo, G. (2016). *Enfoque basado en gamificación para el aprendizaje de un lenguaje de programación.* Recuperado el 15 de Agosto de 2024 de https://bit.ly/4ebj8Vr

- Becerra, Y. (2020). *Una revisión de plataformas robóticas para el sector de la construcción.* Recuperado el 12 de Agosto de 2024 de http://www.scielo.org.co/scielo.php?pid=S0123-921X2020000100115&script=sci_arttext

- Casale, J. (2016). *Introducción a la programación: Aprenda a programar sin conocimientos previos*. RedUsers. Recuperado el 13 de Noviembre de 2024 de https://bit.ly/3zeukS5

- Del Prado, A., & Lamas, N. (2014). *Alternativas para la enseñanza de pseudocódigo y diagrama de flujo.* Rev. Electrónica Iberoam. Educ. en Ciencias y Tecnol, 5(3). Recuperado el 13 de Noviembre de 2024 de https://bit.ly/48ROajv

- Duran, et al. (2007). *Programación orientada a objetos con Java.* Recuperado el 12 de Agosto de 2024 de https://bit.ly/4eu4CYu

- Farrell, J. (2013). *Introducción a la programación lógica y diseño*: (7 ed.). Cengage Learning. Recuperado el 12 de Agosto de 2024 de https://elibro.net/es/lc/itq/titulos/93265

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Jimenez, et al. (2014). *Fundamentos de Programación: Diagramas de flujo, Diagramas N-S, Pseudocódigo*. Alfaomega Grupo Editor SA. Recuperado el 14 de Noviembre de 2024 de https://bit.ly/40S7xXX

- Joyanes, et al. (2003). *Fundamentos de programación.* McGraw-Hill: España. Recuperado el 20 de Agosto de 2024 de https://bit.ly/3ALHsP8

- Juganaru, M. (2015). *Introducción a la programación*: (ed.). Grupo Editorial Patria. Recuperado el 14 de Noviembre de 2024 de https://elibro.net/es/lc/itq/titulos/39449

- Mazón, et al (2015). *Fundamentos de Programación Orientada a objetos en java*. Ediciones Utmatch. Ecuador-Machala. Recuperado el 02 de Agosto de 2024 de https://bit.ly/3Za8jhP

- Martínez, J. (2005). *Fundamentos de programación en Java*. Recuperado el 12 de Agosto de 2024 de https://bit.ly/3MwUHpm

- Moreno, J. (2015). *Programación*: (ed.). RA-MA Editorial. Recuperado el 12 de Agosto de 2024 de https://elibro.net/es/lc/itq/titulos/62476

- Oviedo Regino, E. M. (2015). *Lógica de programación orientada a objetos*: (ed.). Ecoe Ediciones. Recuperado el 12 de Noviembre de 2024 de https://elibro.net/es/lc/itq/titulos/70431

- Ramírez, J. (2019). *Fundamentos iniciales de lógica de programación I. Algoritmos en PseInt y Python*: (1 ed.). D - Institución Universitaria de Envigado. Recuperado el 12 de Agosto de 2024 de https://elibro.net/es/lc/itq/titulos/226488

- Rodríguez, J. (2003). *Introducción a la programación. Teoría y práctica. Editorial Club Universitario*. Recuperado el 12 de Agosto de 2024 de https://bit.ly/3UPu6sl

- Sánchez, et. al. (2015). *Análisis de los problemas de aprendizaje de la programación orientada a objetos.* Recuperado el 12 de Noviembre de 2024 de https://dialnet.unirioja.es/descarga/articulo/7915479.pdf

- Sebesta, R. W. (2018). *Concepts of Programming Languages* (12th ed.). Pearson.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Spigariol, L. & Palumbo, N & Passerini, N. (2021). *Competencias en programación orientada a objetos.* Recuperado el 12 de Noviembre de 2024 de https://sedici.unlp.edu.ar/handle/10915/135020

- Tejera, et al . (2020*). Lenguajes de programación y desarrollo de competencias clave. Revisión sistemática.* Redie, 16. Recuperado el 12 de Agosto de 2024 de https://bit.ly/48QwXHg

- Vital, M. (2019). *Estructuras de control para la programación*. Vida Científica Boletín Científico De La Escuela Preparatoria No. 4, 7(13). Recuperado el 14 de Noviembre de 2024 de https://repository.uaeh.edu.mx/revistas/index.php/prepa4/article/view/3573

- Vidal, et al. (2014). *Revisión de un Enfoque Modular de Programación Orientada a Aspectos.* Recuperado el 14 de Noviembre de 2024 de https://www.scielo.cl/scielo.php?pid=S0718-07642014000400020&script=sci_arttext

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

10. SOLUCIONARIO

Autoevaluación 1		
Pregunta	Respuesta	Retroalimentación
1	a.	Un algoritmo es una secuencia de pasos finitos y específicos diseñados para resolver un problema o realizar una tarea. No es un lenguaje de programación, una pieza de hardware ni un software ejecutable.
2	d.	PSeInt es una herramienta educativa que ayuda a los principiantes a aprender los fundamentos de la programación mediante el uso de pseudocódigo, sin la complejidad de la sintaxis de los lenguajes de programación formales.
3	c.	El pseudocódigo es una forma de expresar algoritmos utilizando un lenguaje simplificado que se asemeja al lenguaje natural, lo que lo hace más comprensible antes de implementarlo en un lenguaje de programación formal.
4	d.	Java es un lenguaje de programación de alto nivel que permite a los desarrolladores escribir código más cercano al lenguaje humano. Assembly Language, Machine Language y el lenguaje Binario son ejemplos de lenguajes de bajo nivel.
5	b.	PSeInt es una herramienta diseñada específicamente para facilitar el aprendizaje de la programación utilizando pseudocódigo, proporcionando un entorno amigable para principiantes.
6	c.	El pseudocódigo es una herramienta valiosa para planificar y visualizar la lógica de un programa antes de traducirlo a un lenguaje de programación específico.
7	a.	Un algoritmo está diseñado para ofrecer una solución clara y eficiente para resolver un problema o realizar una tarea, no para diseñar interfaces de usuario o mejorar su diseño.
8	c.	La programación consiste en escribir un conjunto de instrucciones que una computadora puede ejecutar para realizar tareas específicas. No implica la creación de hardware, diseño gráfico de interfaces, ni organización de archivos.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

9	b.	Java, Visual Basic, C++, y Python son lenguajes de alto nivel, diseñados para ser fáciles de leer y escribir por los humanos, a diferencia de los lenguajes de bajo nivel como Assembly y Machine Language.
10	a.	PSeInt permite a los estudiantes aprender a programar utilizando pseudocódigo, lo que simplifica la lógica de programación y elimina la necesidad de dominar la sintaxis compleja de los lenguajes de programación desde el principio.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Autoevaluación 2		
Pregunta	Respuesta	Retroalimentación
1	b.	Todo algoritmo sigue tres pasos básicos: entrada (datos que se reciben), proceso (operaciones o cálculos realizados con los datos) y salida (resultados obtenidos). Los otros términos como "Error", "Bucle" y "Decisión" son acciones adicionales que pueden ocurrir, pero no son los tres pasos clave de un algoritmo.
2	a.	En PSeInt, el tipo de dato "Real" se utiliza para almacenar números con decimales. "Entero" se utiliza para números sin decimales, "Carácter" para caracteres individuales y "Lógico" para valores booleanos.
3	c.	En un diagrama de flujo, el símbolo de proceso (generalmente un rectángulo) se utiliza para representar operaciones o cálculos que realiza el algoritmo. El inicio y el fin del algoritmo se representan con un óvalo, la lectura/escritura con un paralelogramo, y las decisiones con un rombo.
4	c.	El operador + es un operador aritmético utilizado para la suma. Los otros operadores &&, ==, y != son operadores lógicos o relacionales.
5	d.	El tipo de dato "Lógico" se utiliza en PSeInt para almacenar valores booleanos (verdadero o falso). "Entero", "Real", y "Carácter" son tipos de datos para almacenar números y caracteres.
6	b.	La estructura secuencial ejecuta las instrucciones de manera lineal, de arriba hacia abajo, en el orden en que se encuentran. Las estructuras condicionales y repetitivas dependen de condiciones o repeticiones.
7	c.	En PSeInt, la palabra clave Definir se utiliza para declarar una variable y especificar su tipo de dato. "Leer" y "Escribir" son para entrada y salida de datos, mientras que "Proceso" no es una instrucción de declaración.
8	a.	La estructura "Si-Entonces" permite que un programa tome decisiones basadas en condiciones. Si la condición es verdadera,

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

		ejecuta un bloque de código; de lo contrario, puede ejecutar otra acción.
9	c.	La estructura Repetir...Hasta Que en PSeInt ejecuta un bloque de código al menos una vez y luego evalúa la condición. Las estructuras "Mientras" y "Para" pueden no ejecutarse si la condición no se cumple desde el inicio.
10	a.	La estructura "Mientras" evalúa una condición y ejecuta el bloque de código solo si la condición es verdadera, repitiendo esta evaluación antes de cada iteración. Si la condición es falsa desde el principio, el bloque no se ejecuta.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Autoevaluación 3		
Pregunta	Respuesta	Retroalimentación
1	b.	La Programación Orientada a Objetos (POO) es un paradigma de programación basado en el concepto de "objetos", que pueden contener datos (atributos) y código (métodos). No es un lenguaje, técnica de depuración, ni un sistema operativo.
2	c.	Java es uno de los lenguajes de programación más populares y ampliamente utilizados que sigue el paradigma de la Programación Orientada a Objetos (POO). HTML y CSS son lenguajes de marcado y de estilo, respectivamente, y SQL es un lenguaje de consulta para bases de datos.
3	a.	La herencia es uno de los pilares fundamentales de la POO, que permite a una clase derivar de otra y reutilizar código. Algoritmo, pseudocódigo y memoria no son pilares de la POO.
4	d.	El encapsulamiento en POO permite ocultar los detalles internos de un objeto, exponiendo solo lo necesario. Esto mejora la seguridad y la modularidad del código.
5	c.	Una clase es una plantilla o molde que define atributos y métodos comunes a los objetos creados a partir de ella. No es un archivo de texto, tipo de dato primitivo, ni comando del sistema operativo.
6	c.	La instanciación es el proceso de crear un objeto a partir de una clase en POO. No es eliminar un objeto, crear una copia, ni convertir un dato primitivo a un objeto.
7	d.	El polimorfismo es un pilar de la POO que permite que los objetos de diferentes clases se traten como objetos de una clase común, permitiendo múltiples formas. Encapsulamiento, herencia y abstracción son otros pilares, pero no permiten esta característica específica.
8	b.	La herencia permite que una clase (subclase) herede atributos y métodos de otra clase (superclase), promoviendo la reutilización de código y la organización jerárquica. No crea clases a partir de múltiples objetos, multiplica la memoria, ni crea interfaces.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

9	d.	La clase Scanner en Java se utiliza para leer la entrada del teclado y otros tipos de entrada. FileReader, BufferedReader y InputStream son utilizados para leer entradas de archivos y flujos.
10	a.	Los métodos son funciones o acciones que los objetos pueden realizar en POO. Las variables y atributos se utilizan para almacenar datos, mientras que las interfaces definen contratos de métodos que una clase debe implementar.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC