Coding Bootcamp

Core Java

TECHNOLOGY

Introduction to Java

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- Identify the steps required to install and configure Java on a PC
- Analyze the functions of bytecode and class files in Java
- Evaluate the compilation process of Java and its significance in programming
- Categorize the different data types, operators, and elements utilized in Java programming
- Implement conditional and looping statements to solve programming problems

What Is Java?

What Is Java?

Java is a high-level, object-oriented programming language that eliminates platform dependency with its well-known architecture.

It can run on multiple platforms using the Java Virtual Machine (JVM) and is known for its memory management capabilities.

Why Use Core Java?

Following are the key purposes of using Core Java:

Platform independence:

- Java programs can run on any device with a Java Virtual Machine (JVM), making it platform-independent.

Robustness:

- Its strong memory management, exception handling, and type-checking mechanisms make it a robust and reliable programming language.

Rich API:

- It provides a vast set of APIs for various tasks, including networking, data structures, graphical interfaces, and more.

Why Use Core Java?

Security:

- It provides built-in security features like bytecode verification; sandboxing and security APIs enhance safety.

Portability:

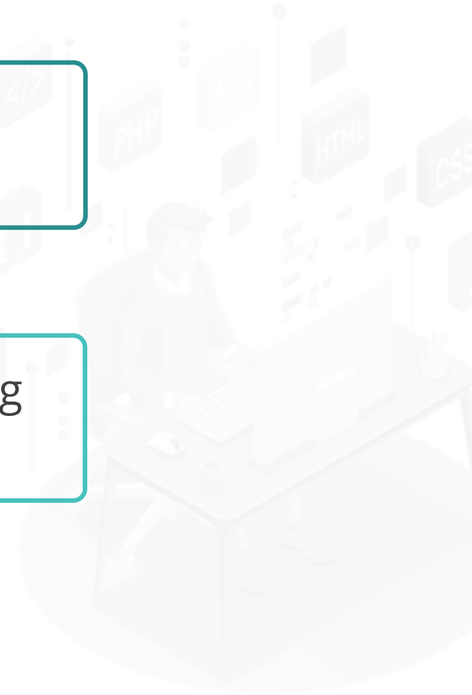
- It runs on any JVM-compatible platform and ensures high portability and flexibility.

Where to Use Core Java?

Core Java plays a pivotal role in diverse industries, driving innovation and efficiency in applications. Below are key applications where it is utilized:

01 **Mobile applications**: Core Java is the primary and one of the most widely used programming languages for developing Android apps.

02 **Financial services**: It is used for developing banking applications, including customer management and transaction processing systems.

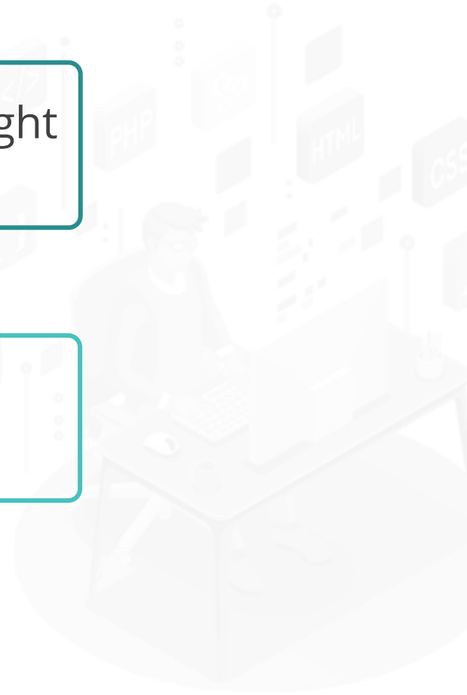


Where to Use Core Java?

Below are key applications where it is utilized:

03 **Gaming industry**: It is used particularly with libraries like LWJGL (Lightweight Java Game Library) for developing games and gaming engines.

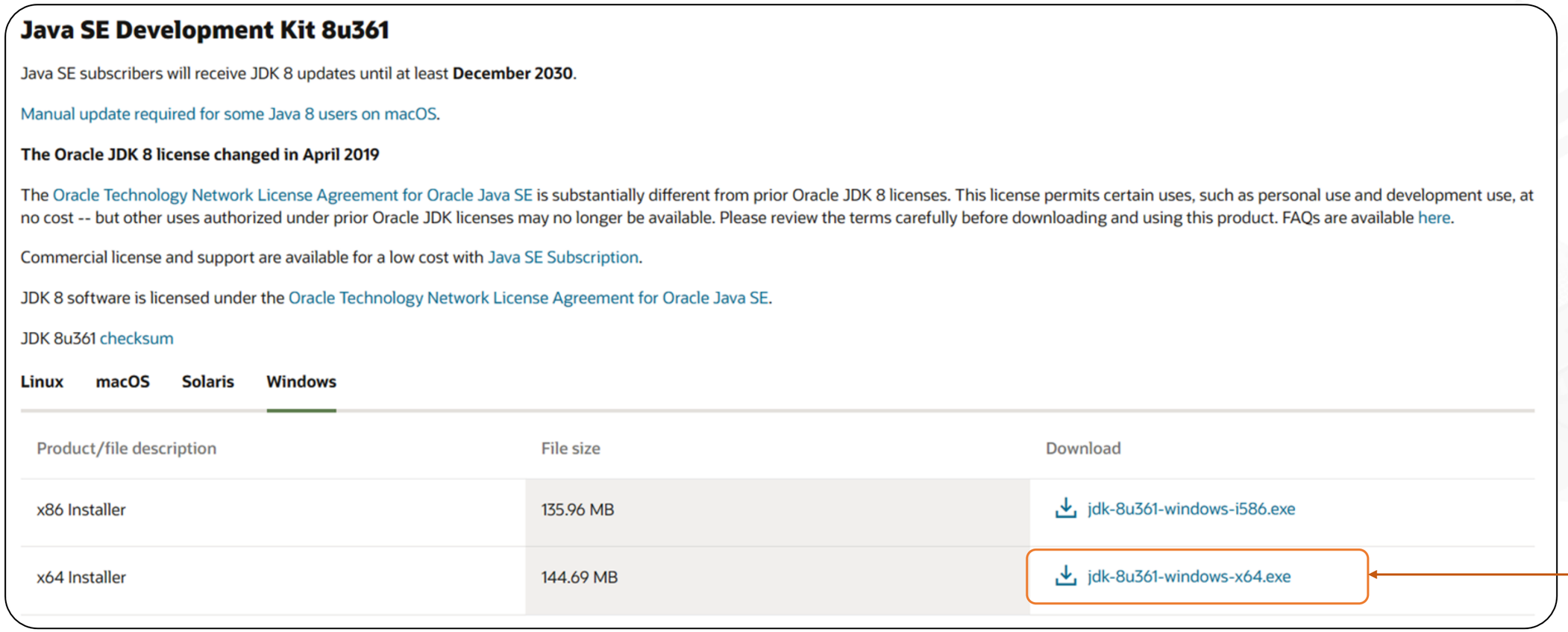
04 **Web applications**: It develops scalable e-commerce platforms for high traffic and secure transactions



Installing and Configuring Java

Installing Java

Navigate to https://www.oracle.com/Java/technologies/downloads/ and click on the respective **jdk-8u361** hyperlink based on the system architecture



Java SE Development Kit 8u361

Java SE subscribers will receive JDK 8 updates until at least **December 2030**.

Manual update required for some Java 8 users on macOS.

The Oracle JDK 8 license changed in April 2019

The Oracle Technology Network License Agreement for Oracle Java SE is substantially different from prior Oracle JDK 8 licenses. This license permits certain uses, such as personal use and development use, at no cost -- but other uses authorized under prior Oracle JDK licenses may no longer be available. Please review the terms carefully before downloading and using this product. FAQs are available here.

Commercial license and support are available for a low cost with Java SE Subscription.

JDK 8 software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE.

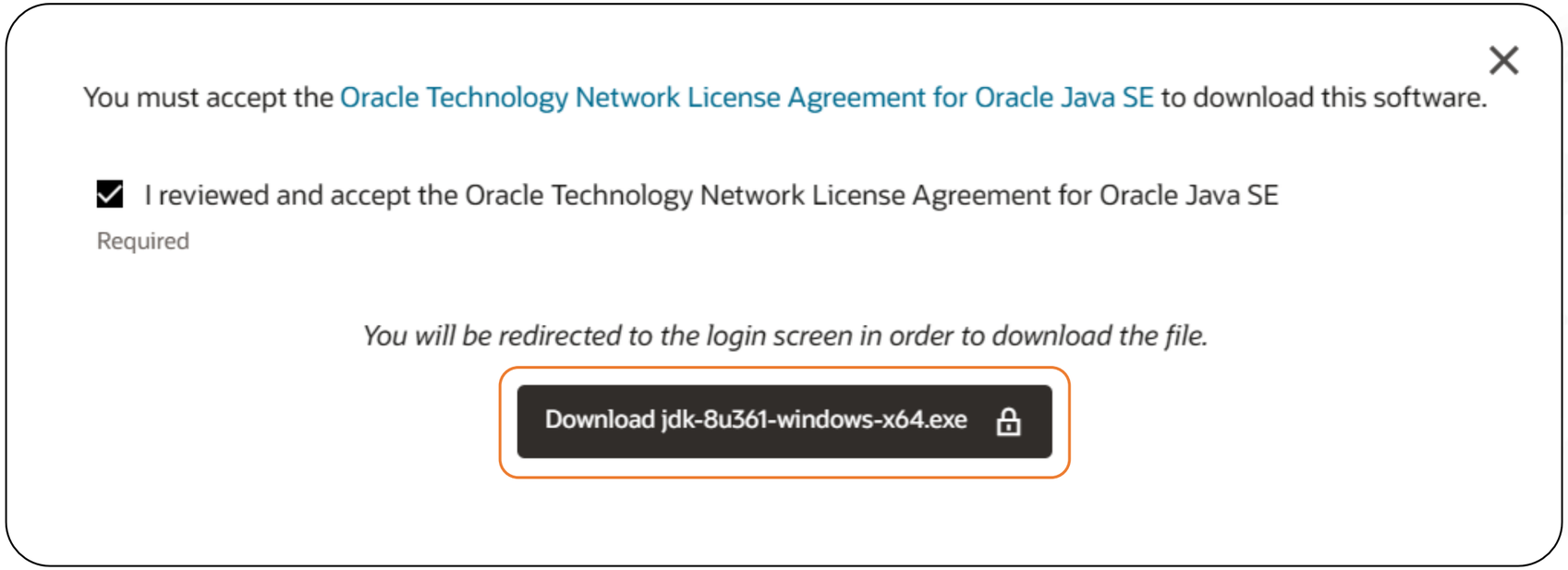
JDK 8u361 checksum

Linux **macOS** **Solaris** **Windows**

Product/file description	File size	Download
x86 Installer	135.96 MB	jdk-8u361-windows-i586.exe
x64 Installer	144.69 MB	jdk-8u361-windows-x64.exe

Installing Java

If required, select the checkbox to accept the Oracle License Agreement.
Then, click on the download link.

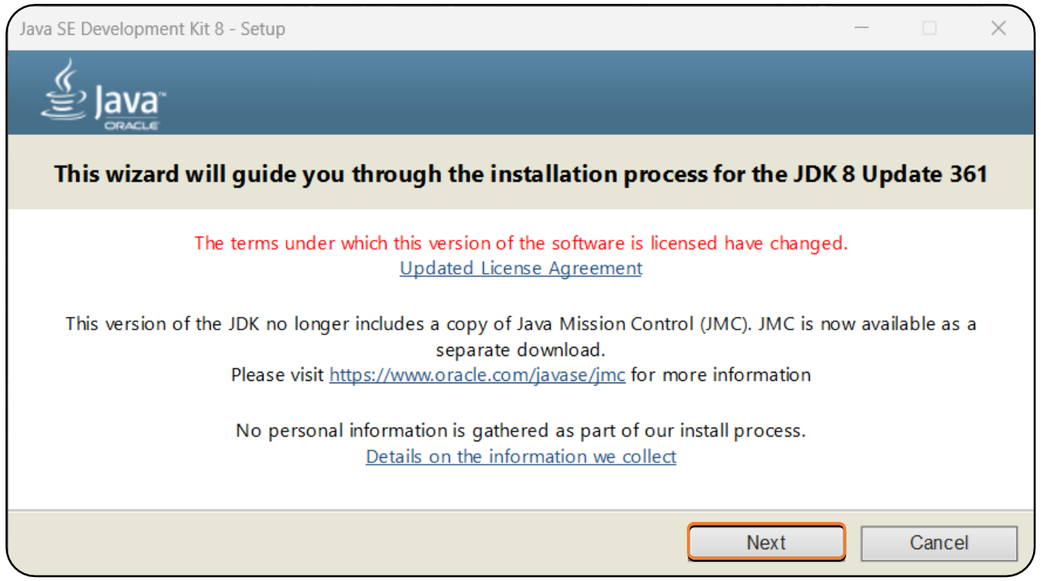


Note

An Oracle account is required for downloading Java JDK.

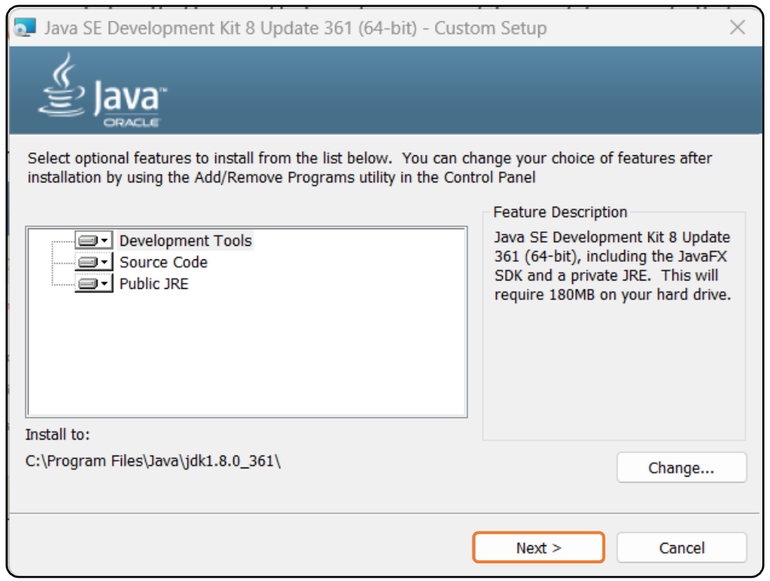
Installing Java

Run the Java executable file from the downloaded location in your system



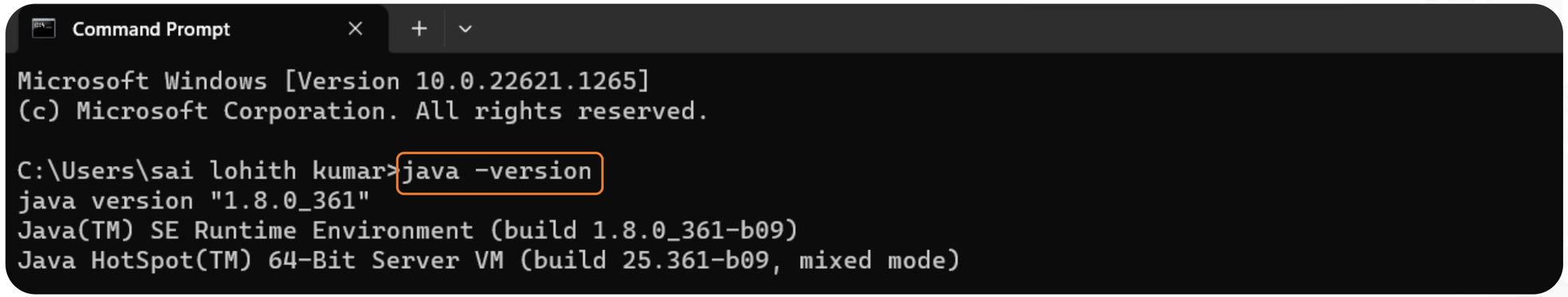
Installing Java

The path location must be set for installation in the setup wizard.



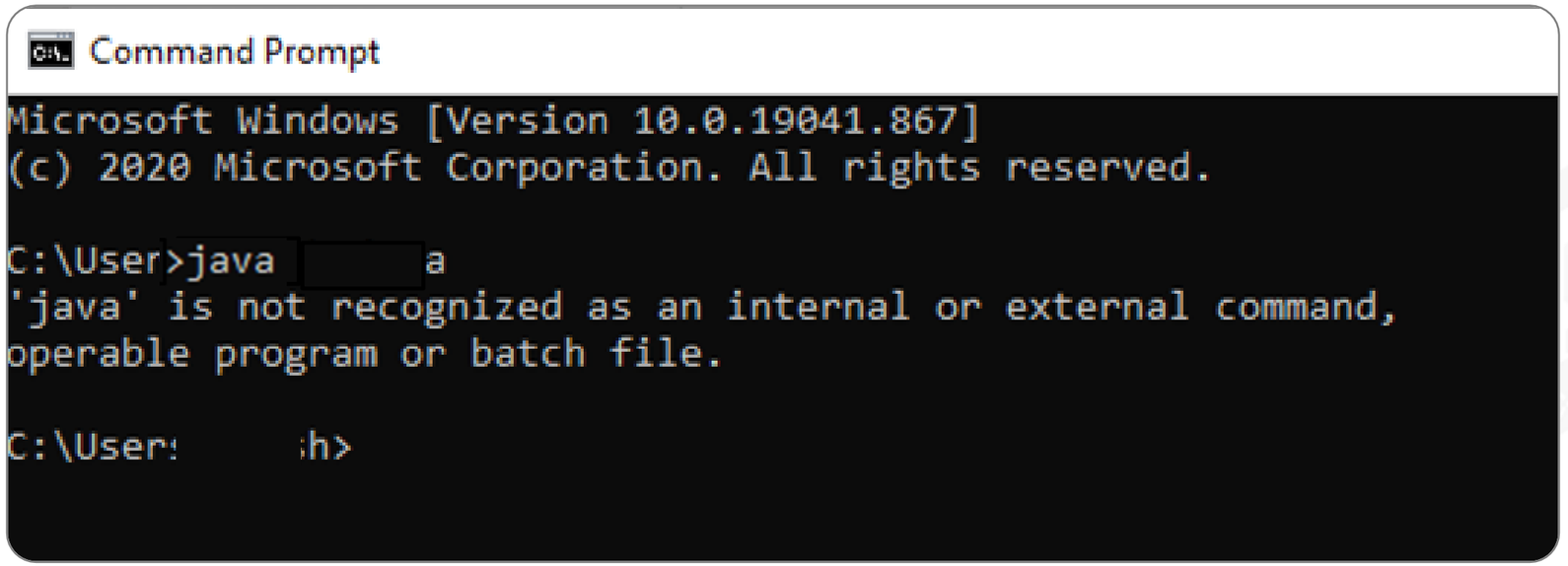
Configuring Java

Type in the Java command **java -version** in the Windows command prompt upon completion of installing the JDK



Configuring Java

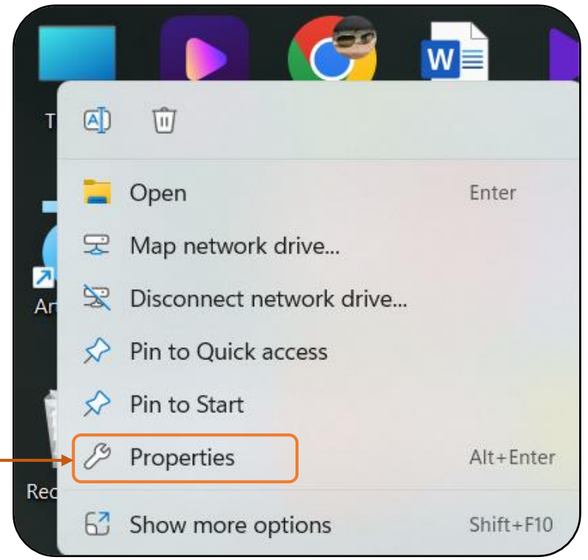
An error is displayed as shown below:



The error means that Java is not properly configured on your system.

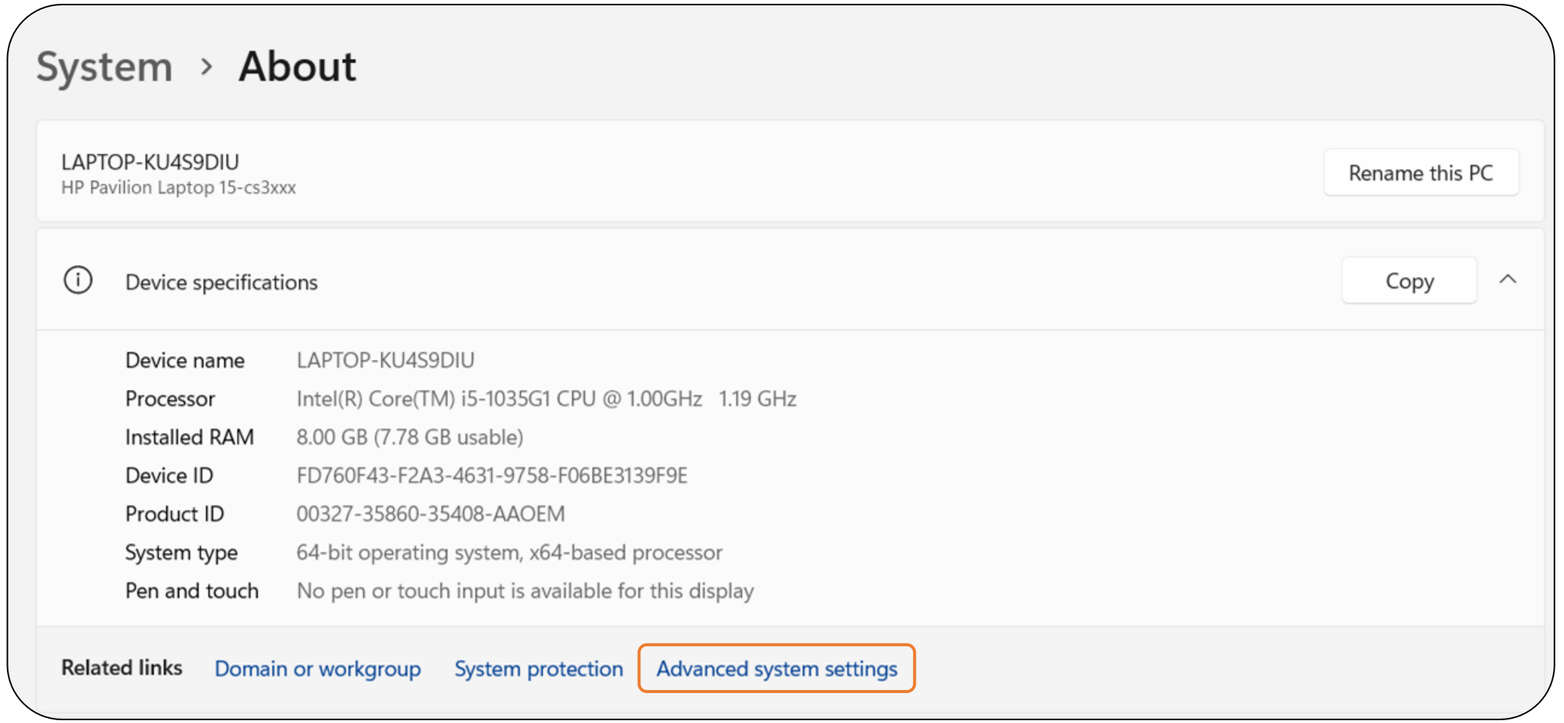
Configuring Java

Right-click on **This PC** icon from the Desktop and select **Properties** from the drop-down list



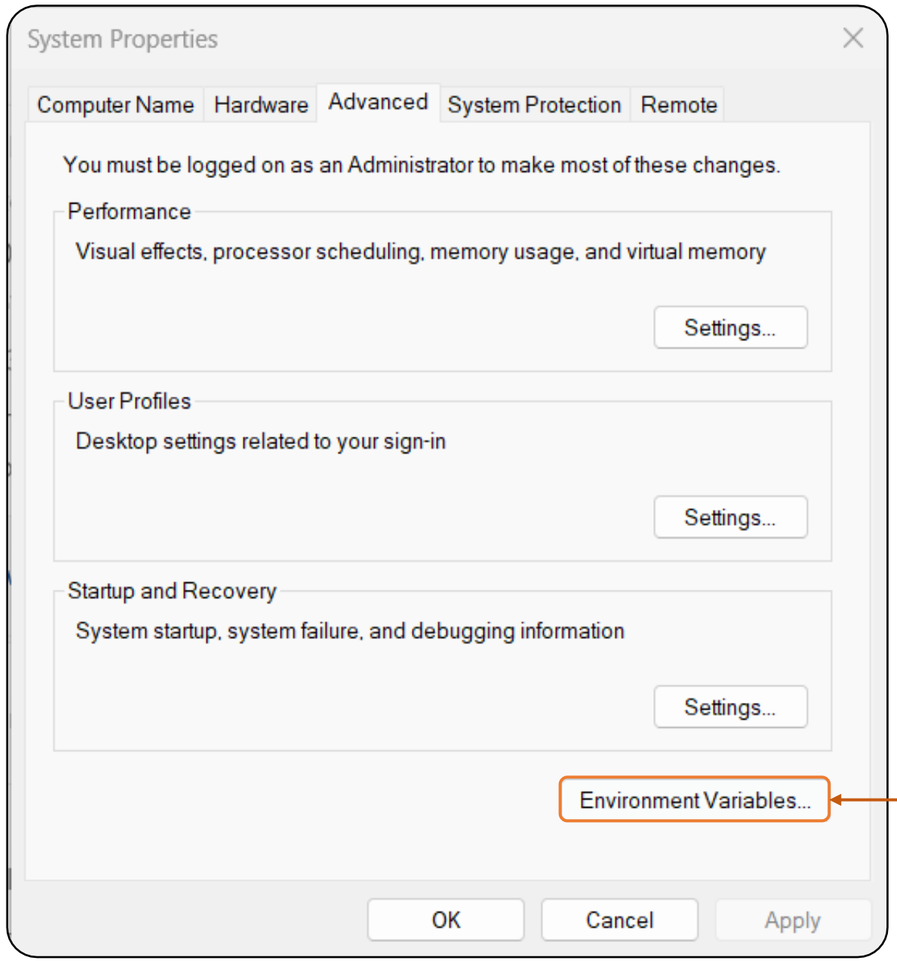
Configuring Java

Search for the **Advanced system settings** link and click on it



Configuring Java

Click on the **Environment Variables** button in the **System Properties** dialog box

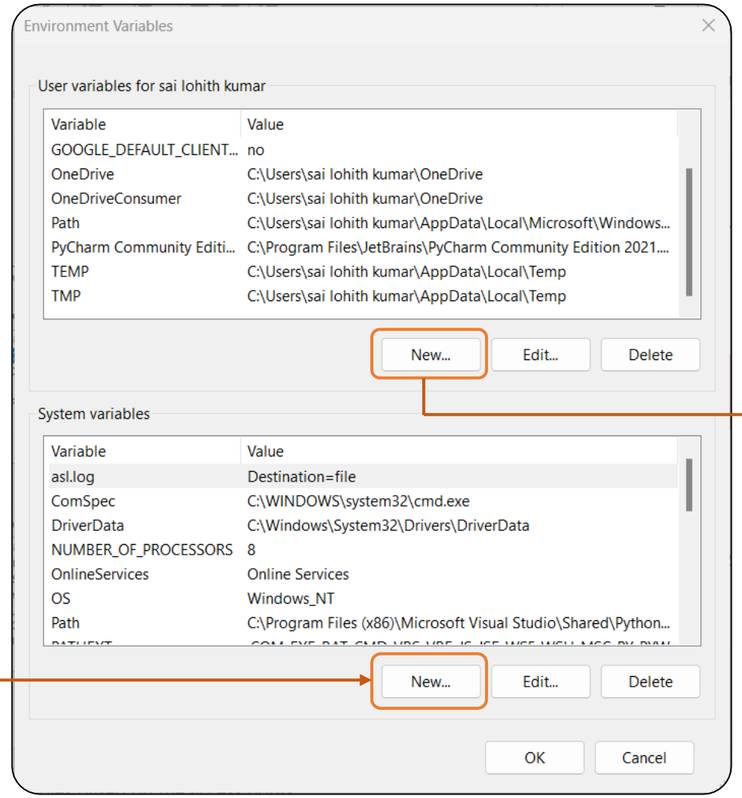


Configuring Java

Click on **New** (either for user variables or for system variables). You can choose by looking at the access privileges.

System variables

Java environment will be configured for all the users who have access to that system.



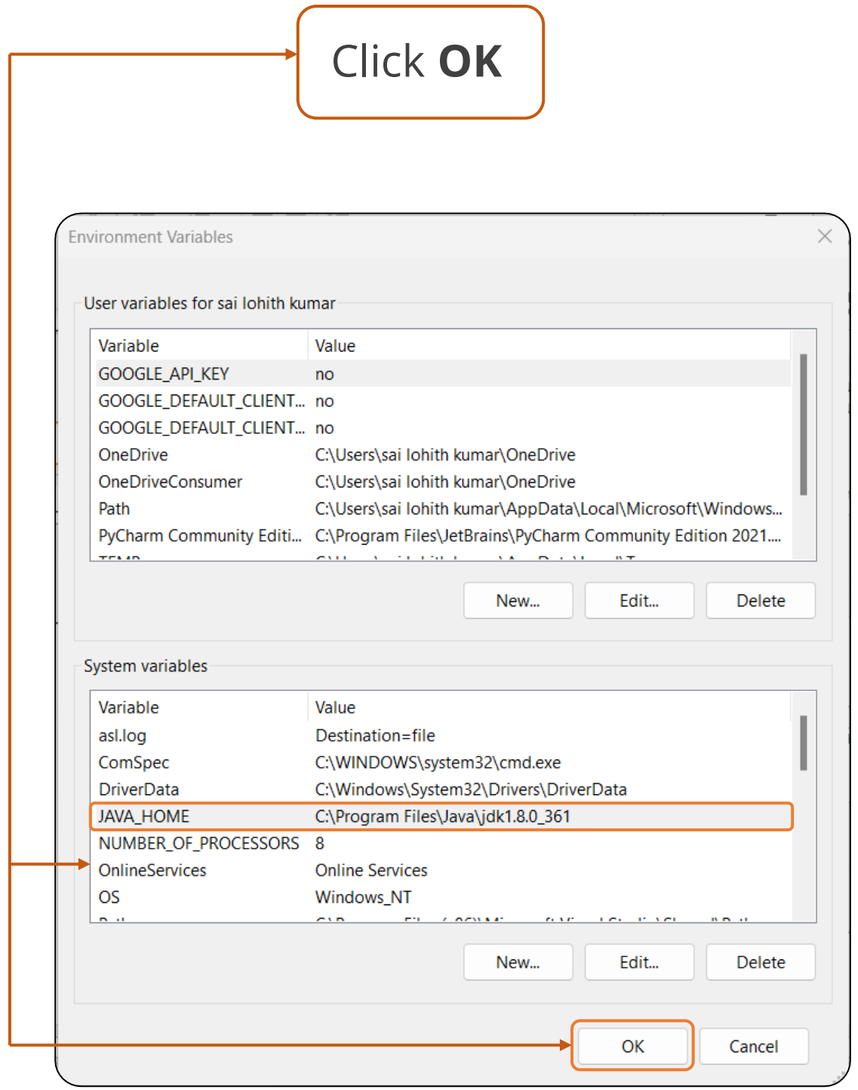
User variables

Java environment will be configured only for a particular user.

Configuring Java

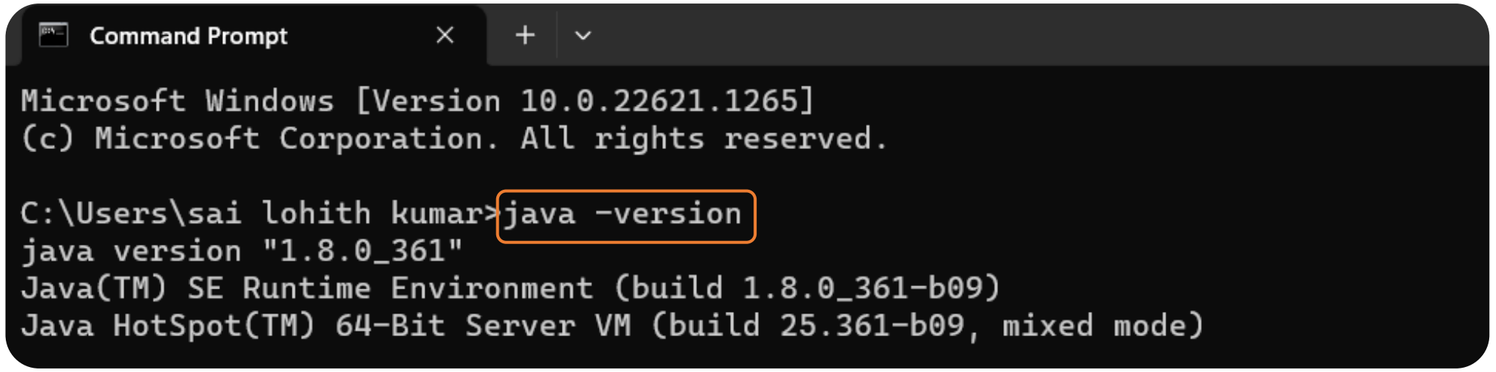
Enter **JAVA_HOME** as the variable name and set the variable value as the path where JDK was installed previously

Configuring Java



Configuring Java

Type in the command **Java -version** in the windows command prompt



Creating the First Java Program

Problem Statement:

You have been assigned the task of creating the first Java program.

Outcome:

By the end of this first Java program, you will understand Java syntax and basic programming constructs. You will also have hands-on experience in writing, compiling, and executing Java code, which will set a strong foundation for your future programming projects.

Note: Refer to the demo document for detailed steps: 01_Creating_the_First_Java_Program

ASSISTED PRACTICE

Assisted Practice: Guidelines

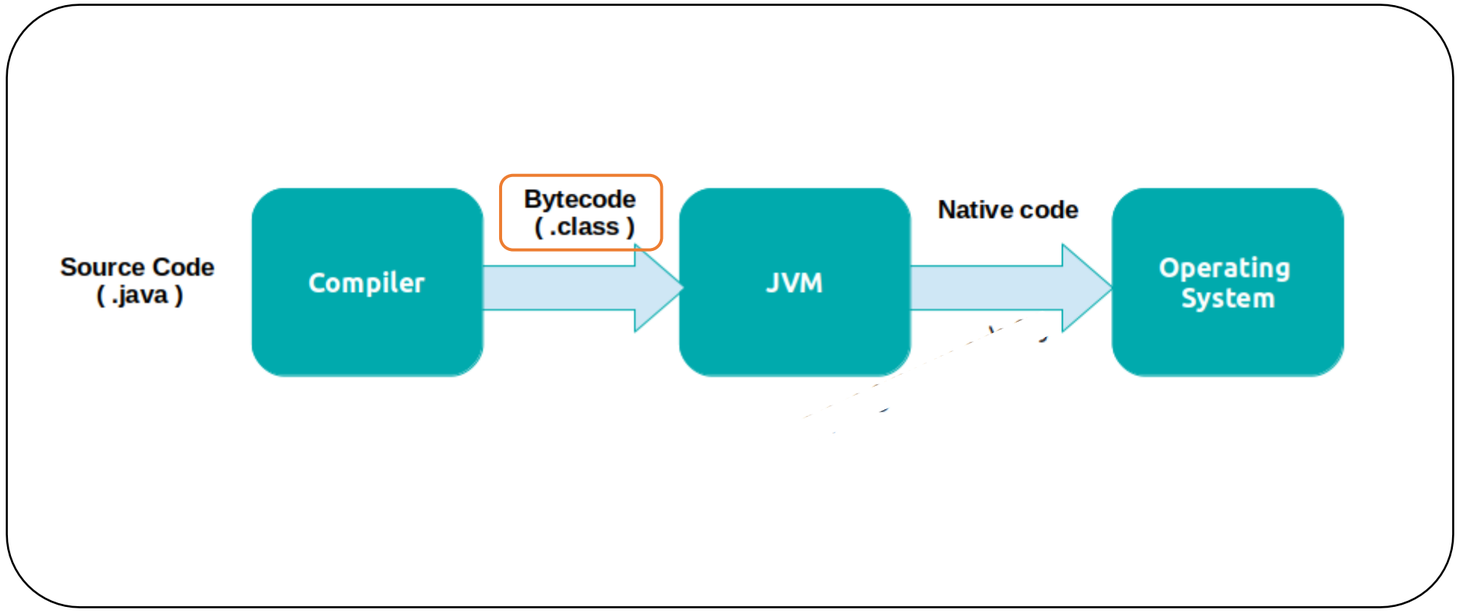
Steps to be followed are:

1. Create a folder on the Eclipse IDE
2. Create the first program in Java

Bytecode and Class Files

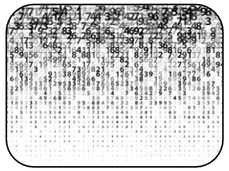
Bytecode

It is an intervening code generated between the actual execution and the compilation of the Java program.

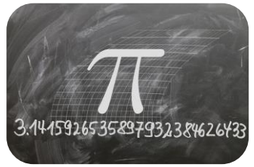


Bytecode

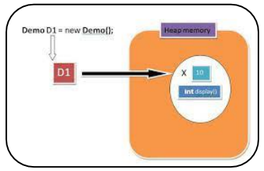
It is not understandable by humans because it consists of machine-readable instructions, such as:



Numerics



Constants



References

```
Compiled from "Customer.java"
class rentalStore.Customer {
  private java.lang.String _name;

  private java.util.Vector<rentalStore.Rental> _rentals;

  public rentalStore.Customer(java.lang.String);
    Code:
      0: aload_0
      1: invokespecial #14                 // Method java/lang/Object."<init>":()V
      4: aload_0
      5: new           #17                 // class java/util/Vector
      8: dup
      9: invokespecial #19                 // Method java/util/Vector."<init>":()V
     12: putfield      #20                 // Field _rentals:Ljava/util/Vector;
     15: aload_0
     16: aload_1
     17: putfield      #22                 // Field _name:Ljava/lang/String;
     20: return
```

Bytecode

It is responsible for analyzing:

```
// Bytecode stream: 03 3b 84 00 01 1a 05
68 3b a7 ff f9
// Disassembly:

iconst_0        // 03
istore_0        // 3b
iinc 0, 1       // 84 00 01
iload_0         // 1a
iconst_2        // 05
imul            // 68
istore_0        // 3b
goto -7         // a7 ff f9
```

- Types of program objects
- Scopes of program objects
- Nesting depth of program objects

Java Bytecode

It eliminates platform dependency.

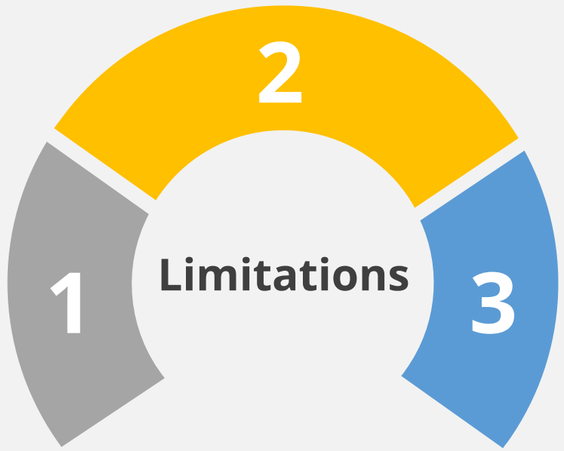
James Gosling

He is the creator of Java and designed the language to eliminate platform dependency. He achieved this by using bytecode, a binary format that can run on any platform with a Java Virtual Machine (JVM). Thanks to bytecode, Java programs may be compiled once and run on any platform that has a compatible JVM, making Java a fully cross-platform language.

Java Bytecode

Bytecode is a machine-executable code for the Java Virtual Machine.

Is proportional to the interpreter's availability



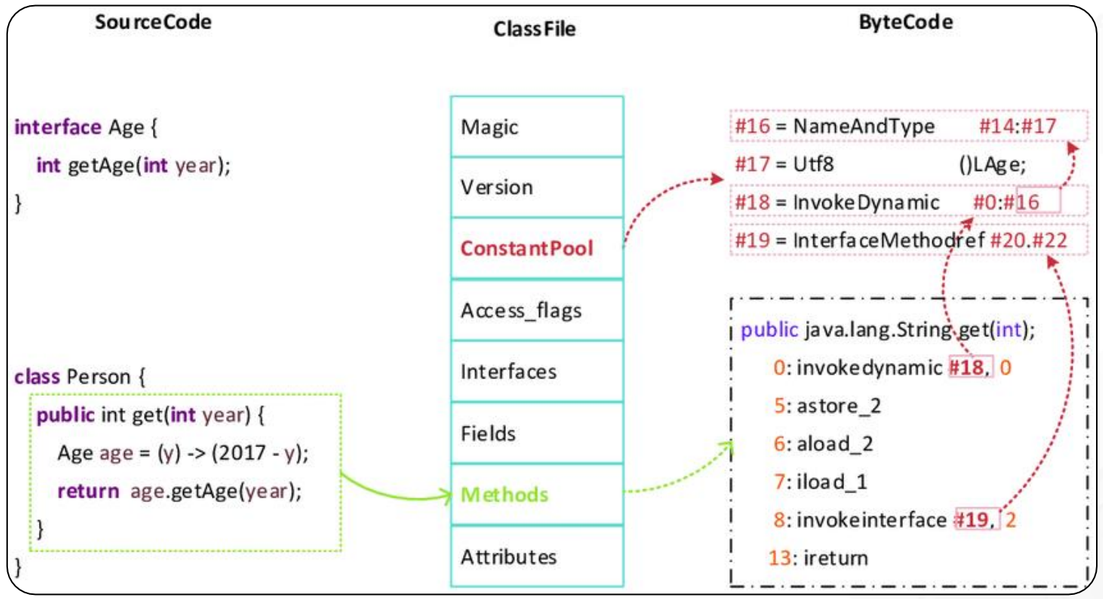
Cannot execute code

Leads to performance overhead

Java Bytecode

Every time a Java class is loaded, a stream of bytecodes for each class method is received.

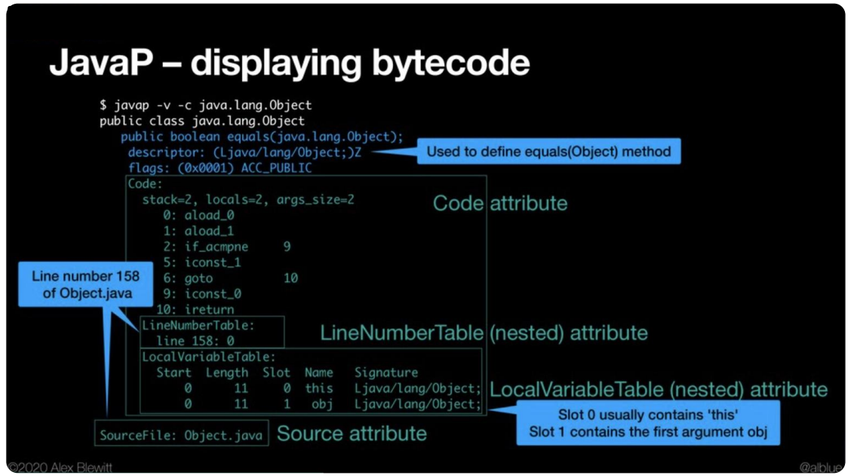
The bytecode is invoked when the class is called upon while running Java code.



Compilation Process

Compilation Process

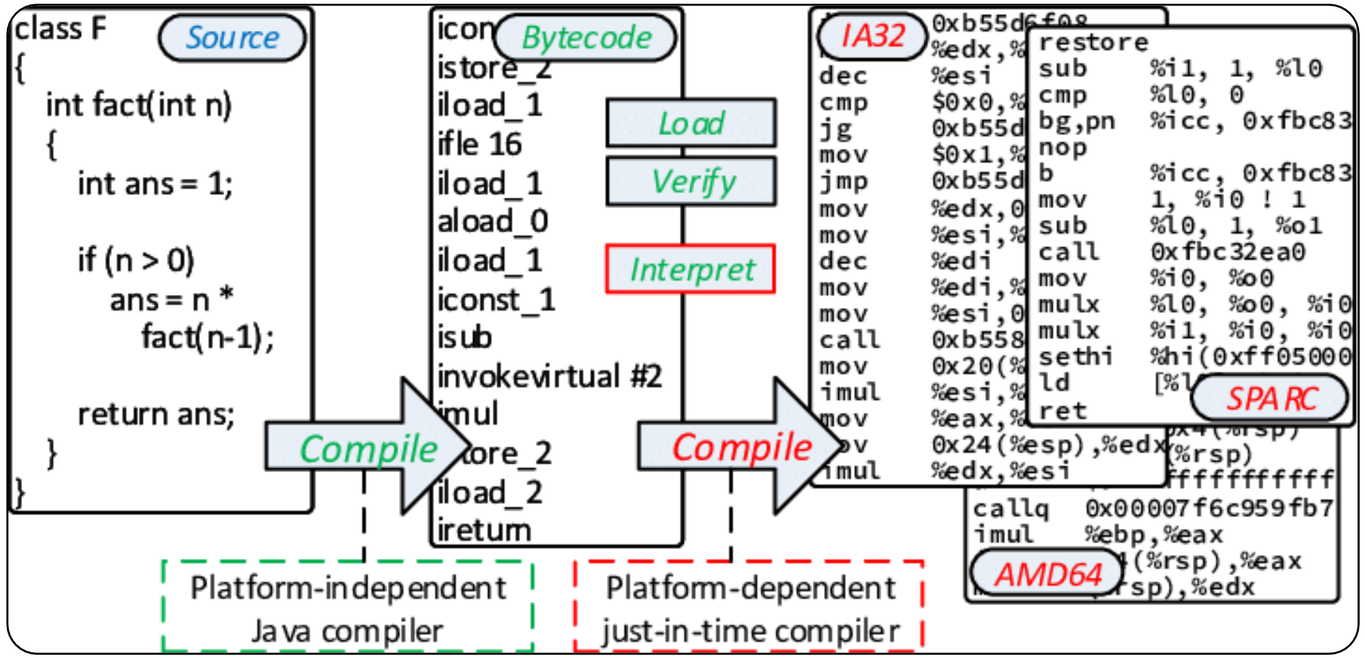
It transforms source code from a high-level programming language to a lower-level language, such as machine code or bytecode that a computer can execute.



The bytecode can be represented by appending **.class** in the file name.

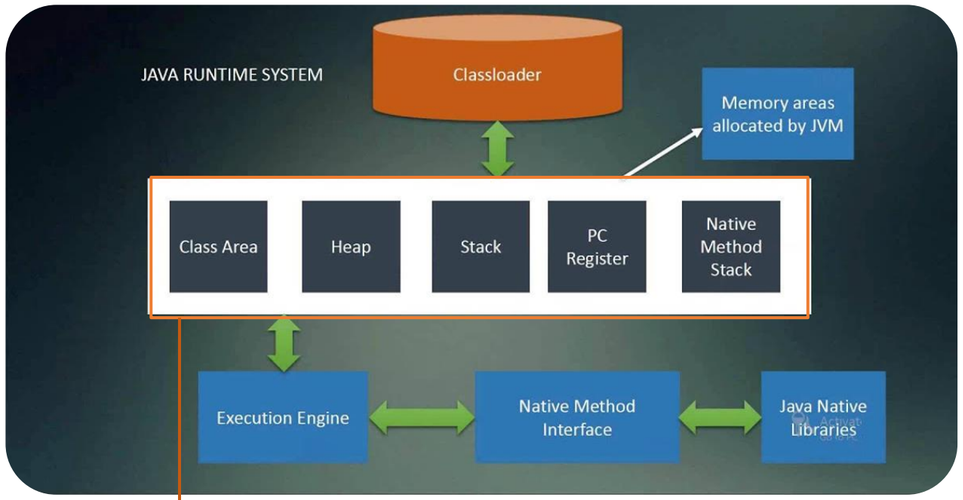
Java Compiler

The compiler supports cross-platform or device compatibility while executing the bytecode.



Java Compiler

The Java Virtual Machine (JVM) now takes over the job and executes the bytecode sequentially.



Only JVM has access to these memory locations.

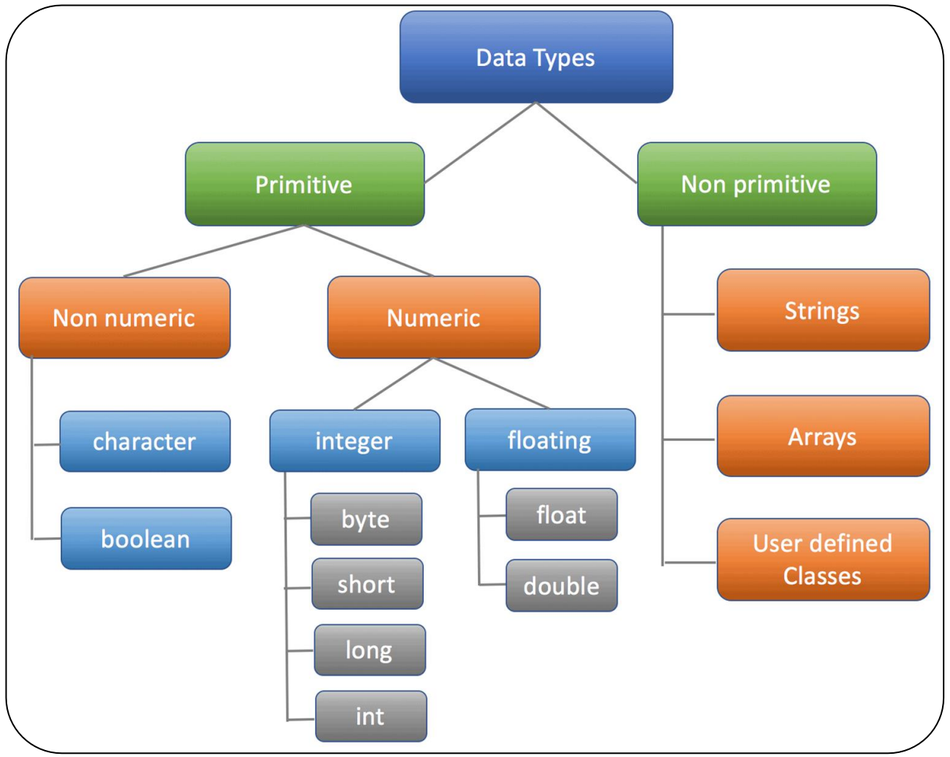
Conclusion:

The fundamental Java installation is sufficient if a user wishes to run a Java program on any application.

Data Types

Data Types

Data types define the values that need to be placed inside a variable.



Primitive Data Types

Data Type	Default Value	Default Size
Boolean	False	1 bit
char	'\u0000'	2 byte
byte	0	1 bit
short	0	2 byte
int	0	4 byte
long	0	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Primitive Data Types

A Boolean expression returns a Boolean value, either true or false.

Example

```
Boolean car = false;
```

- Permissible for only two values
- Provides the function of toggling on various conditions
- The size cannot be measured since it is a bit of data.

Primitive Data Types

Byte refers to the values between -128 and 127.

Illustration

```
byte x = 5;
byte y = -10;
```

The byte data type is an 8-bit signed two’s complement integer. It can hold values from -128 to 127.

Primitive Data Types

Short is a shorter version of int and is for effective memory utilization.

Illustration

```
short x = 500;
short y = -500;
```

The code initializes x with the value 500 and y with the value -500, both within the range of values that can be stored in a short variable.

Primitive Data Types

Long is a longer version of int and comes into the picture when the values are greater than the count given by int.

Illustration

```
long a = 100000L;
long b = -200000L;
```

The L at the end of each literal (100000L and -200000L) indicates that these values are of the type long.

Primitive Data Types

Float is depicted by its floating point, defined by IEEE guidelines. The scope of float can be removed when precision is needed.

Illustration

```
float f1 = 254.5f;
```

Non-Primitive Data Types

Some of the common non-primitive data types in Java are:

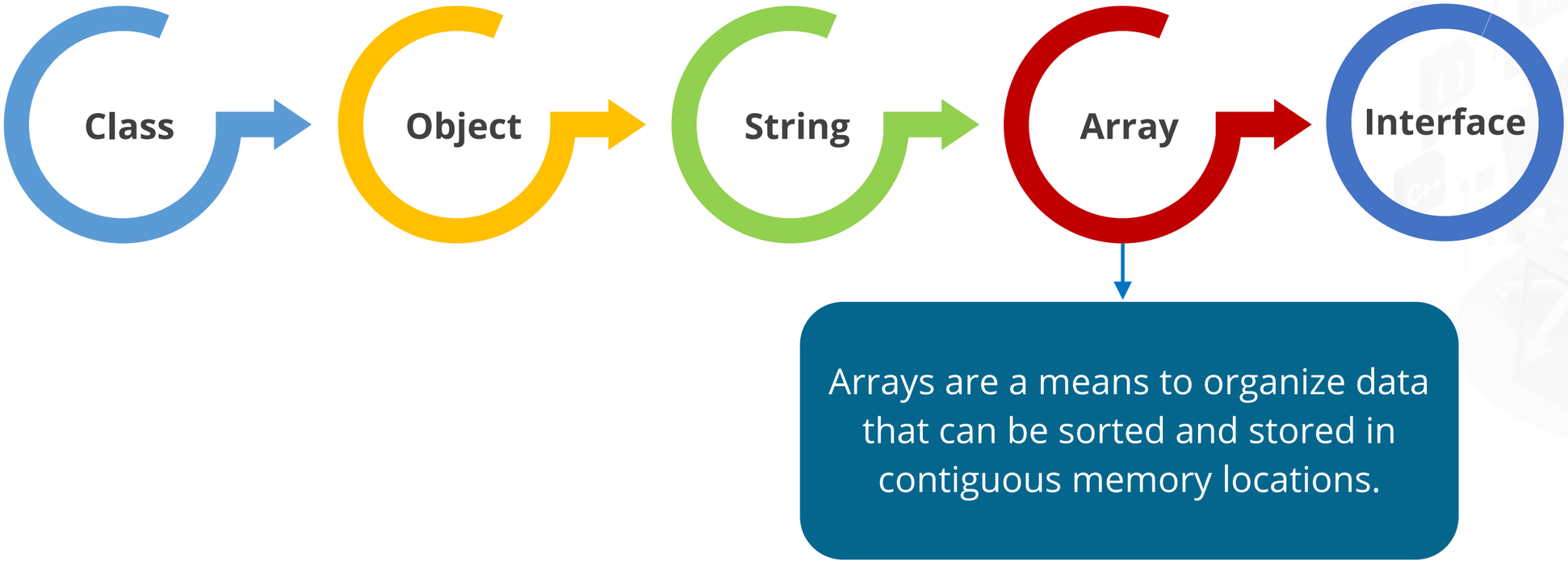
Non-Primitive Data Types

User will create classes and objects in Java.

Non-Primitive Data Types

A string data type is traditionally an order of characters in succession, either as a literal constant or as a kind of variable.

Non-Primitive Data Types



Non-Primitive Data Types

An interface is a reference type in Java. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Implementing Data Types in Java

Problem Statement:

You have been given a task to implement Data Types in Java pro.

Outcome:

By implementing data types in a Java program, you will learn how to effectively manage and use different types of data in your coding projects. You will also gain a deeper understanding of Java's type system, enhancing your ability to write efficient and error-free code.

Note: Refer to the demo document for detailed steps: 02_Implementing_Data_Types_in_Java

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a new class on the Eclipse IDE
2. Create the program to implement Data Types in Java

TECHNOLOGY

Operators

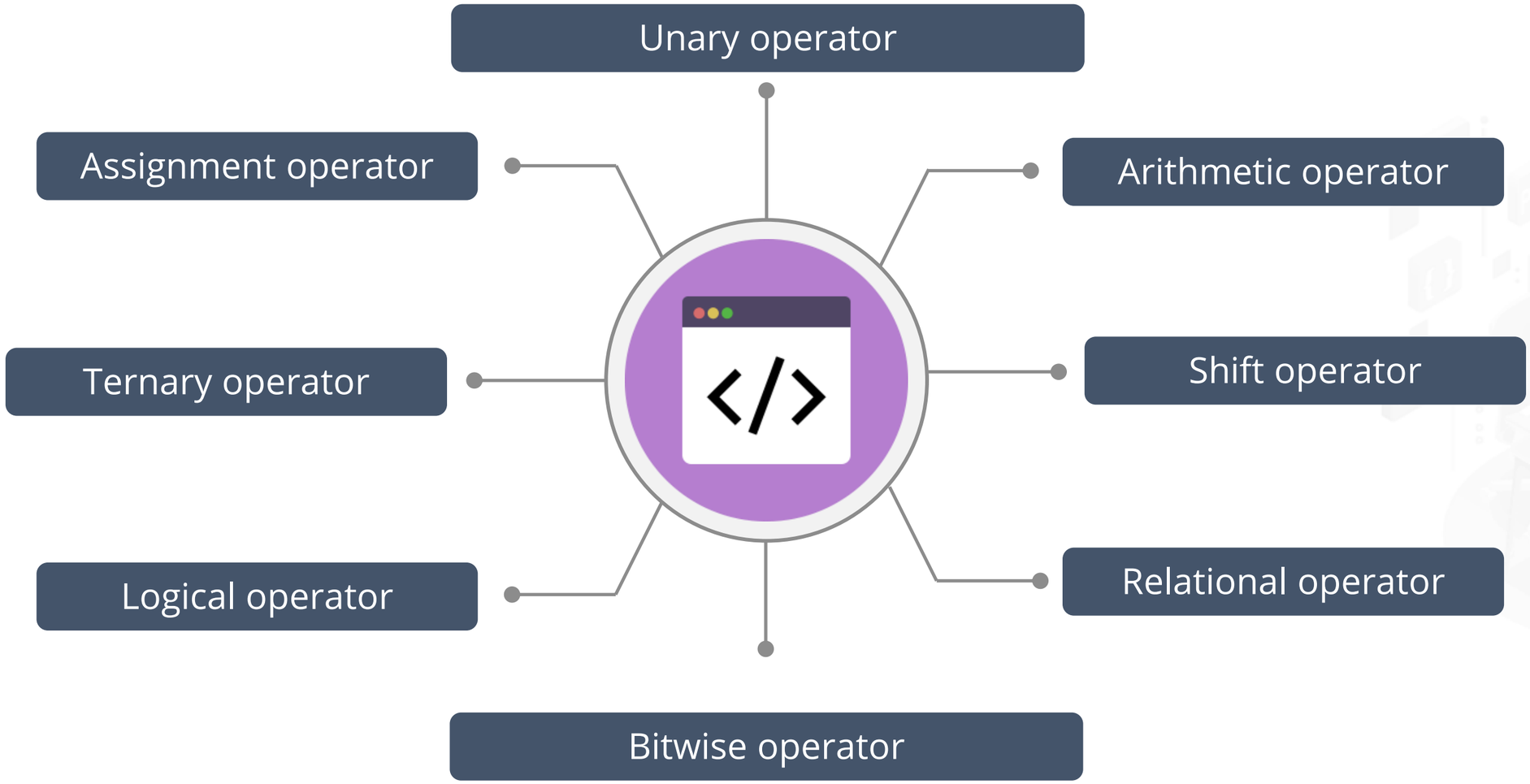
simplilearn

Operators

They hold significant importance when it comes to performing calculations and are composed of operands.

```
D:\Studyopedia>javac StudyDemo1.java

D:\Studyopedia>java StudyDemo1
Studyopedia Java Tutorial
Arithmetic Operators:

Addition Arithmetic Operator
Value of (a + b) is 150

Subtract Arithmetic Operator
Value of (b - a) is 50

Division Arithmetic operator
Value of (c / a) is 3

Multiplication Arithmetic operator
Value of (a * b) is 5000

Modulus Arithmetic operator
Value of (c % b) is 50
```

Types of Operators



Types of Operators

Operator Type	Category	Precedence
Unary	postfix	expr++ expr--
	prefix	++expr --expr +expr -expr ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Unary Operator

Example:

Unary operator: ++ and –

```
public class OperatorExample{
public static void main(String
args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Unary operator: ~ and !

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=false;
boolean d=true;
System.out.println(~a);//-11 (minus of
total positive value which starts from 0)
System.out.println(~b);//9 (positive of
total minus, positive starts from 0)
System.out.println(!c);//true (opposite of
boolean value)
System.out.println(!d);//false
}}
```

Arithmetic Operator

Example:

```
public class OperatorExample{
public static void main(String
args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

```
public class OperatorExample{
public static void main(String
args[]){
System.out.println(9*9/4+2-1*4/2);
}}
```

Left Shift Operator

Example:

```
public class OperatorExample{
public static void main(String args[]){
System.out.println(8<<2);//8*2^2=8*4=32
System.out.println(8<<3);//8*2^3=8*8=64
}}
```

Right Shift Operator

Example:

```
public OperatorExample{
public static void main(String args[]){
System.out.println(8>>2);//8/2^2=8/4=2
System.out.println(40>>2);//40/2^2=40/4=10
System.out.println(32>>3);//24/2^3=32/8=4
}}
```

Shift Operator

Example:

```
public class OperatorExample {
    public static void main(String args[]) {
        System.out.println(8 >> 2);
// Shifts 8 right by 2 bits (equivalent to dividing by
4), resulting in 2.
        System.out.println(40 >> 2);
// Shifts 40 right by 2 bits (equivalent to dividing by
4), resulting in 10.
        System.out.println(32 >> 3);
// Shifts 32 right by 3 bits (equivalent to dividing by
8), resulting in 4.
    }
}
```

AND Operator

Example:

```
public class OperatorExample{
public static void main(String args[]){
int a=9;
int b=4;
int c=19;
System.out.println(a>b&&a<c);// false
System.out.println(a<b&a<c);// false
}}
```

OR Operator

Example:

```
public class Operator{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 as second condition is not verified
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 as second condition is verified
}}
```

Implementing Operators in Java

Problem Statement:

You have been assigned a task to implement operators in a Java program.

Outcome:

By implementing operators in a Java program, you will master the use of arithmetic, logical, and relational operators to manipulate data effectively. This will enhance your problem-solving skills and enable you to build more dynamic and functional Java applications.

Note: Refer to the demo document for detailed steps: 03_Implementing_Operators_in_Java

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a new class on the Eclipse IDE
2. Create the program to implement operators in Java

Elements in Java

Java: Programming Language

Punctuation decisions characterize the Java programming language.

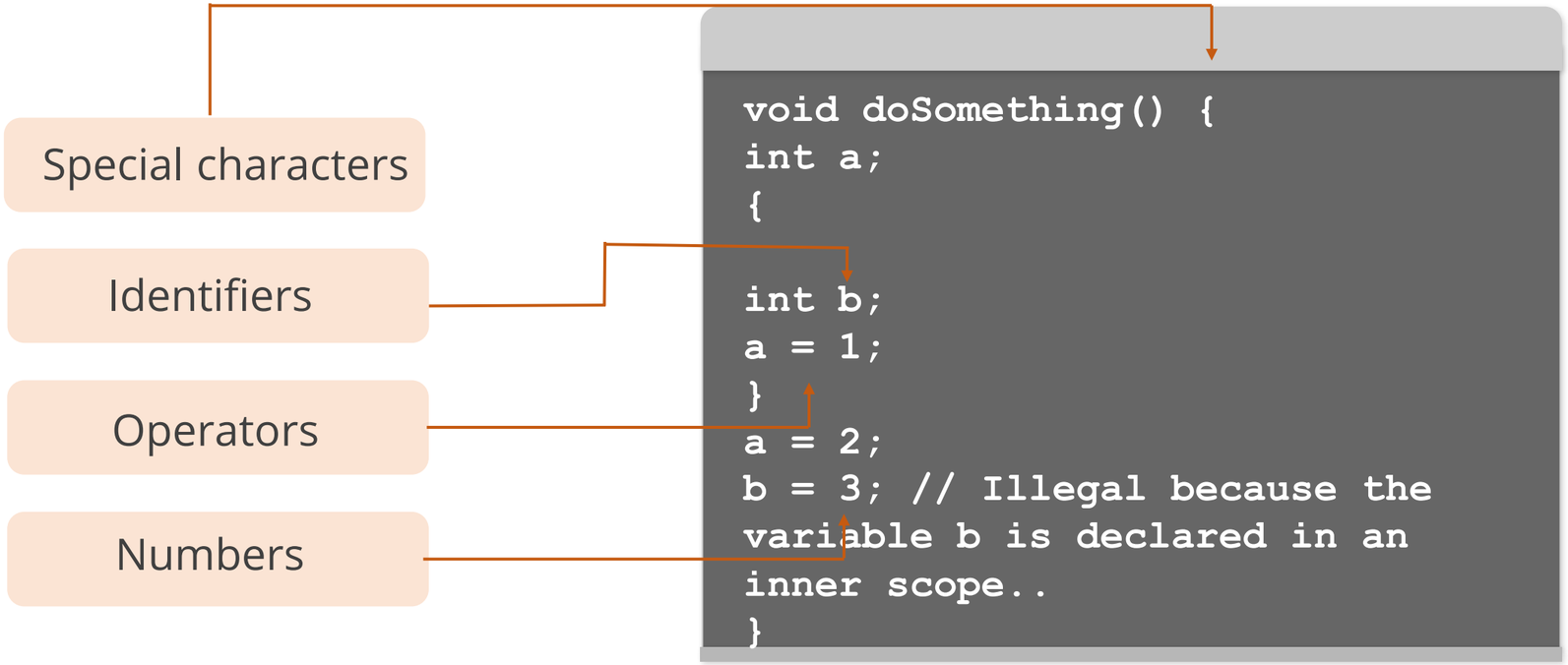
```
public void processbatal() {
  do {int data = getData();
  If (data < 0)
  performOperationi(data);
  Else
  performOperation2(data);
  } while (hasMoreData());
  }
```

These decisions:

- Indicate how the language components can be used to shape grammatically correct developments
- Determine the importance of grammatically lawful builds

Lexical Tokens

Java programs are made up of several low-level language components called lexical tokens. These are the building blocks for more complex developments.



These can assemble high-level builds like expressions, statements, methods, and classes.

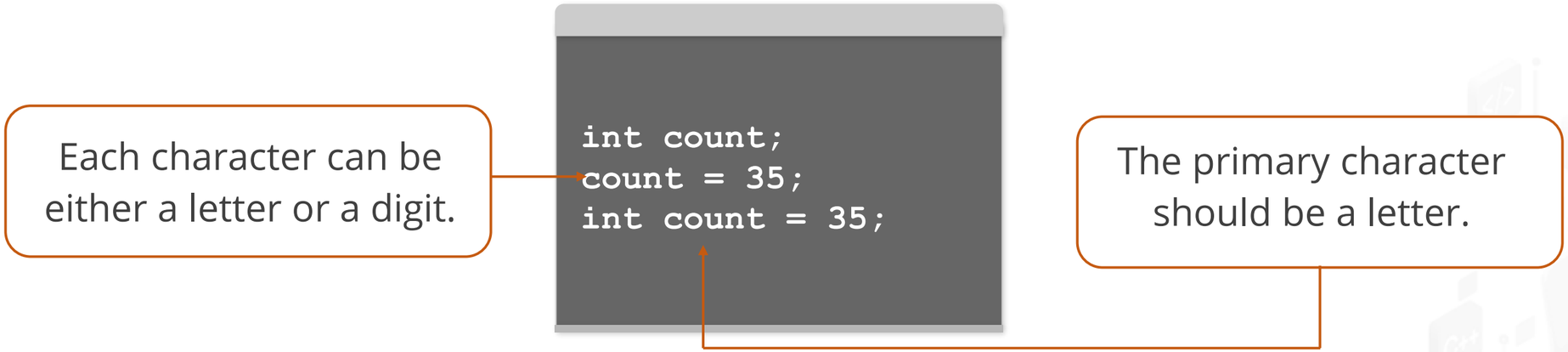
Identifier

It is a name in a Java program and can be utilized for:

- Labels
- Variables
- Classes
- Methods

Naming the Identifier

Identifiers follow a specific naming convention, which includes the following rules:

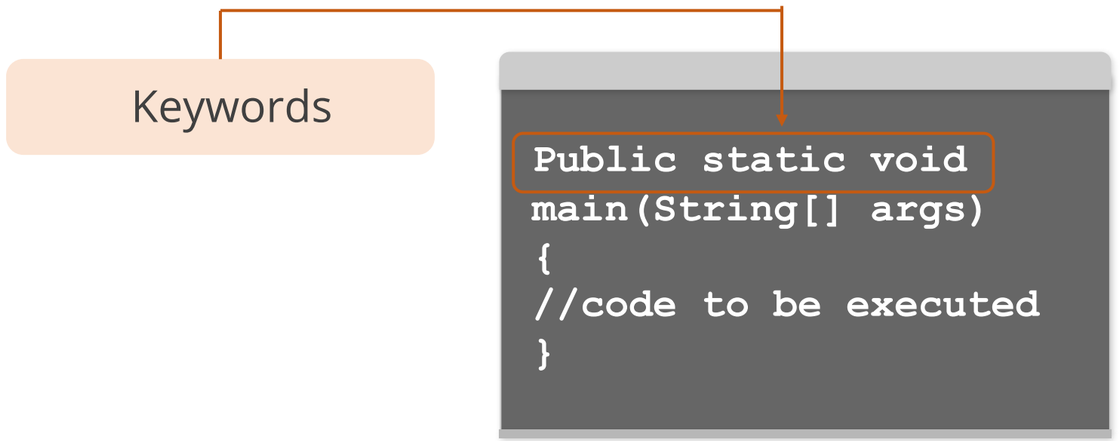


Note

Connecting punctuation (like underscore_) and any currency symbols, such as $, ¢, ¥, or £, should be kept away from identifier names.

Keywords

A keyword is a reserved word that has a specific meaning and purpose within the language.



Note

Keywords cannot be used as identifiers such as variable, method, or class names.

Comment

It provides descriptions and explanations for codes not meant to be executed by the compiler.

Comment

```
// This is a simple java program that prints a message to the console.
Public class HelloWorld
{
Public static void main(String[] args)
{
//Print a message to the console
System.out.println("Hello, world!");
}
}
```

Comment

There are three types of comments in Java:

Single-line comments:

```
// This is a single-line comment
int x = 10; // This is another
single-line comment
```

This type of comment starts with // and can be used to add a brief description to a single line of code.

Comment

Multi-line comments:

```
/*
This is a multi-line comment that
can be used
to provide detailed explanations for
a block of code
*/
int y = 20; /* This is another way
to add a
multi-line comment on a single line
of code */
```

This type of comment starts with /* and ends with */. It is used to add detailed explanations or to comment out multiple lines of code.

Comment

Javadoc comments:

```
/**
* This method adds two numbers and
returns the result.
* @param a the first number to be
added
* @param b the second number to be
added
* @return the sum of a and b
*/
public int add(int a, int b) {
   return a + b;
}
```

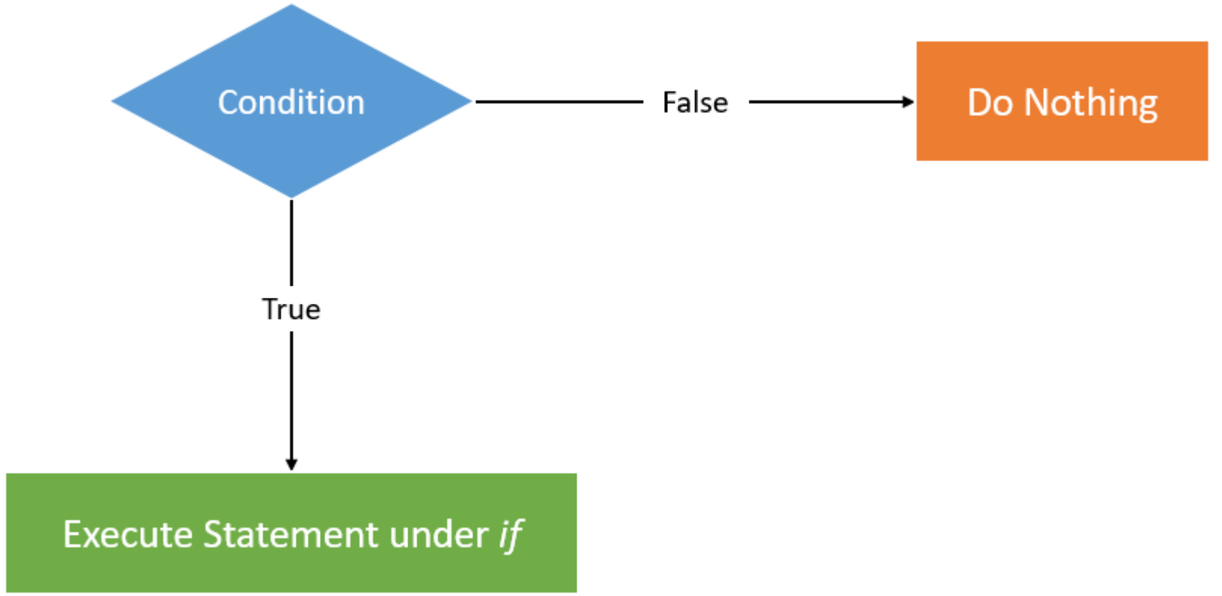
This type of comment is used to generate documentation for code using the javadoc tool. It starts with /** and ends with */.

TECHNOLOGY

Conditional Statements

Conditional Statements

They aid in making decisions by specifying which statements should be executed based on certain conditions.



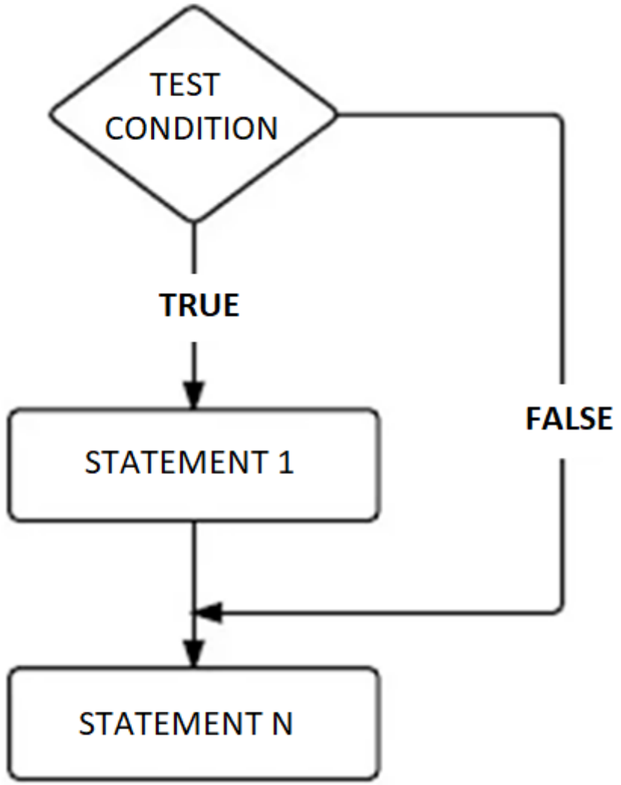
Conditional Statements

Some common types of conditional statements include:

- If statements
- If-else statements
- Nested If statements
- Switch statement

If Statements

An **if** statement decides whether a statement will be executed or not based on the given statement.



If Statements

Syntax for if statement is as follows:

```
if(condition) {
  // statements to be executed if
the condition is true
}
```

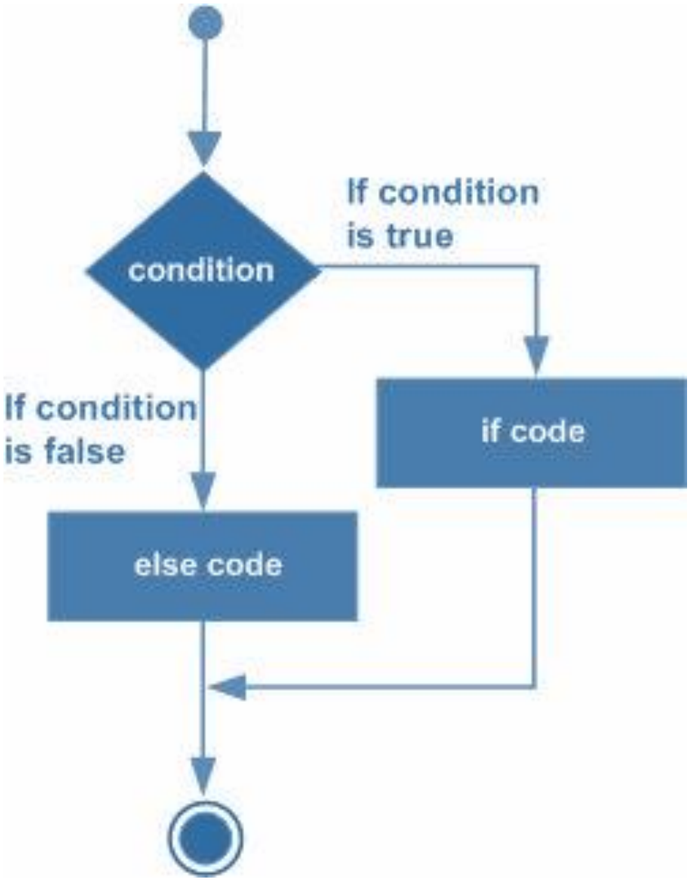
If the condition is true, the statements inside the curly braces are executed.

Example:

```
int x = 13;
int y = 12;
if(x < y) {
   System.out.println("x is less
than y");
}
```

If-Else Statements

The **if-else** statement is used to return something when the condition is false.



If-Else Statements

Syntax for if-else statement is as follows:

```
if(condition) {
  // code to be executed
} else {
  // code to be executed
}
```

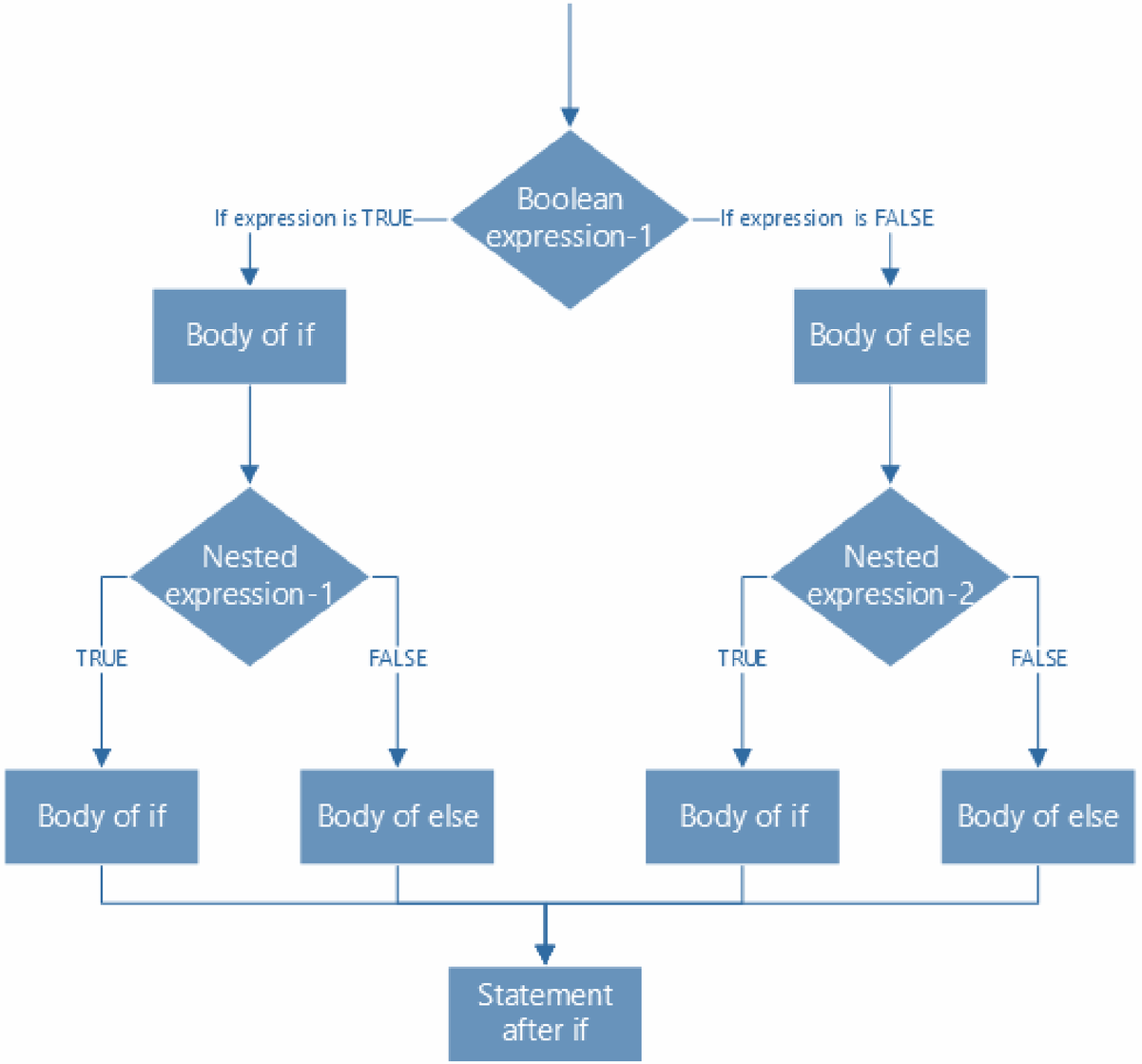
Example:

```
int x = 10;
int y = 12;
if(x < y) {
   System.out.println("x is less
than y");
} else {
    System.out.println("x is greater
than y");
}
```

If the condition is true, the if statement is executed.
If the condition is false, the else statement is executed.

Nested If Statements

The **if** statement focuses on another **if** statement.



Nested If Statements

Syntax for nested if statement is as follows:

```
if(condition1) {
    //Nested if-else inside the body of "if"
    if(condition2) {
       //Code inside the body of nested "if"
     }
     else {
       //Code inside the body of nested "else"
     }
}
else {
    //Code inside the body of "else."
}
```

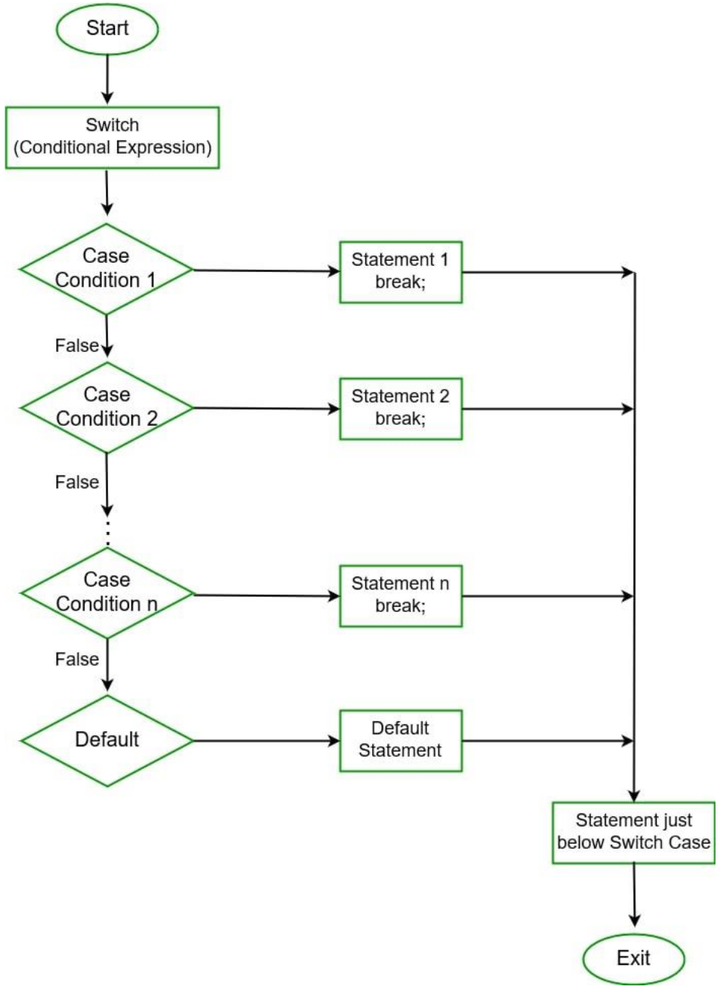
Example:

```
int age=25;
int weight=48;
if(age>=18){
  if(weight>50){
    System.out.println("You are
eligible for Boxing championship");
  } else{
    System.out.println("You are not
eligible for Boxing Championship");
  }
} else{
   System.out.println("Age must be
greater than 18");
}
```

If condition 1 is true, the code inside the first if statement is executed.
If condition 1 is false, the code inside the else block is executed.

Switch Statement

The **Switch** statement executes one statement from numerous conditions.



Switch Statement

Syntax for switch statement is as follows:

```
switch(expression){
 case value1:
   //code to be executed;
   break;  //optional
 case value2:
   //code to be executed;
   break;  //optional
 ...
 default:
   // code to be executed if all
cases are not matched;
}
```

Example:

```
int number=200;
switch(number){
   //Case statements
  case 100:
System.out.println("100");
    break;
  case 200:
System.out.println("200");
    break;
  case 300:
System.out.println("300");
    break;
   //Default case statement
  default:System.out.println("Not in
100, 200 or 300");
}
```

Implementing If-Else Statement

Problem Statement:

You have been given a task to implement the if-else statement in Java.

Outcome:

By implementing the if-else statement in Java, you will learn how to control the flow of your programs based on conditions. This will equip you with the ability to make dynamic decisions in your code, greatly enhancing the functionality and adaptability of your Java applications.

Note: Refer to the demo document for detailed steps: 04_Implementing_If-else_Statement

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a folder on the Eclipse IDE
2. Create a program to implement if-else statements in Java

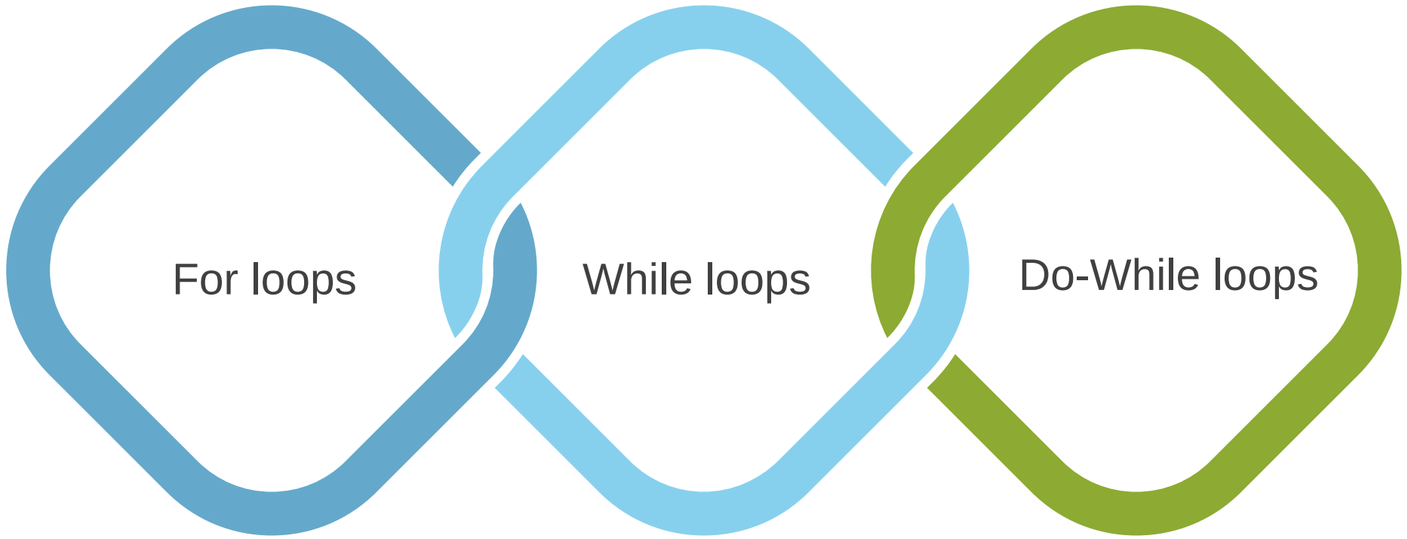
TECHNOLOGY

Looping Statements

Loops

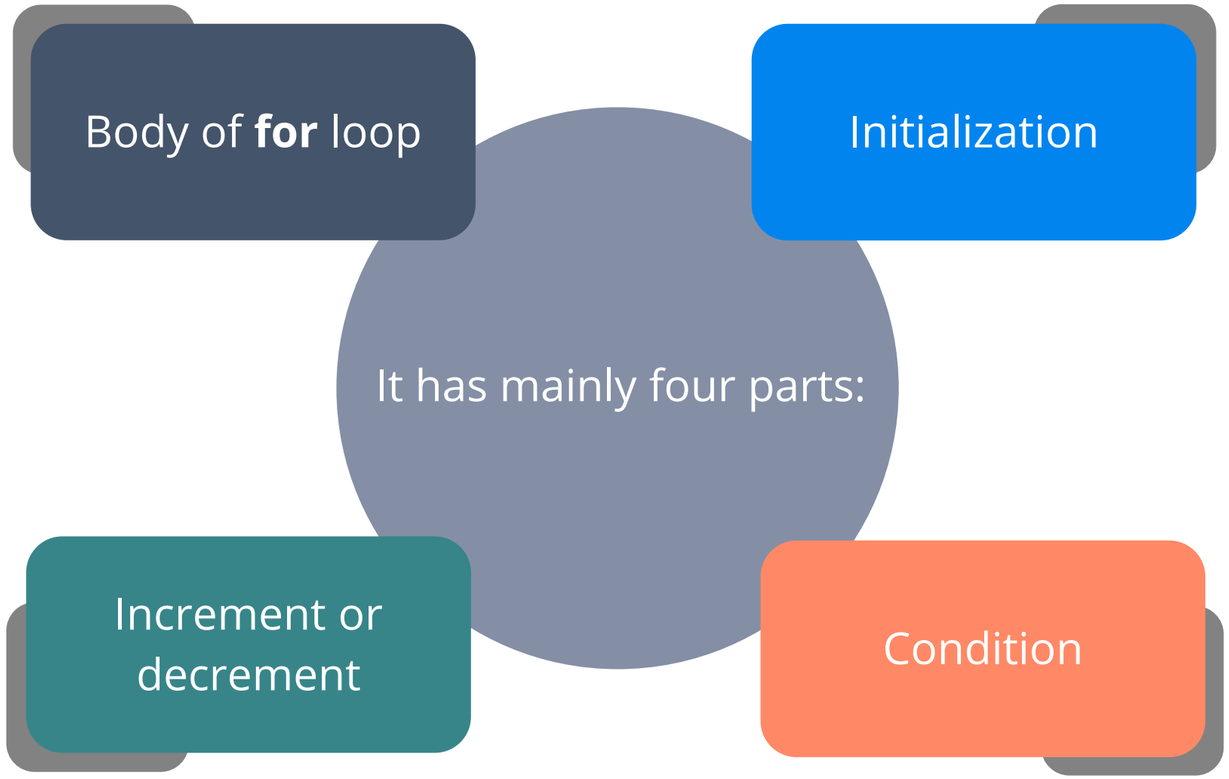
Loops are used to execute the same piece of code repeatedly until a certain condition is met.

There are several types of loops commonly used for repetitive execution:



For Loops

The **for** loop is employed to iterate a piece of a program repeatedly.



For Loops

Syntax:

```
for(initializer; condition;
increment/decrement) {
  // statements
}
```

Here, initialization is the initial value of the loop variable, condition is the condition for continuing the loop, and increment or decrement is the expression that updates the value of the loop variable in each iteration.

For Loops

Example:

```
for(int i=1;i<=10;i++){
        System.out.println(i*i);
}
```

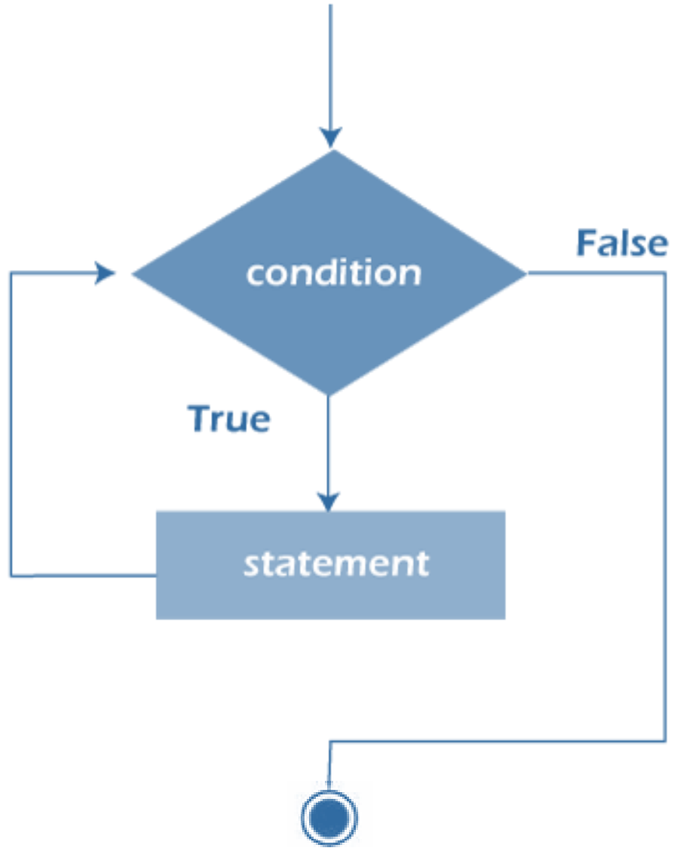
Output:

```
Output:

1
4
9
16
25
36
49
64
81
100
```

While Loops

The **while** loop is used to emphasize a piece of the program repeatedly.

The while loop is considered as an if statement.



While Loops

Syntax:

```
Initializer;
while(condition) {
 increment/decrement;
}
```

Here, condition is the expression evaluated before each iteration of the loop. If the condition is true, the code block inside the loop is executed. This continues until the condition becomes false.

While Loops

Example:

```
int i = 1;
while(i<=10){
    System.out.println(i*i);
    i++;
 }
```

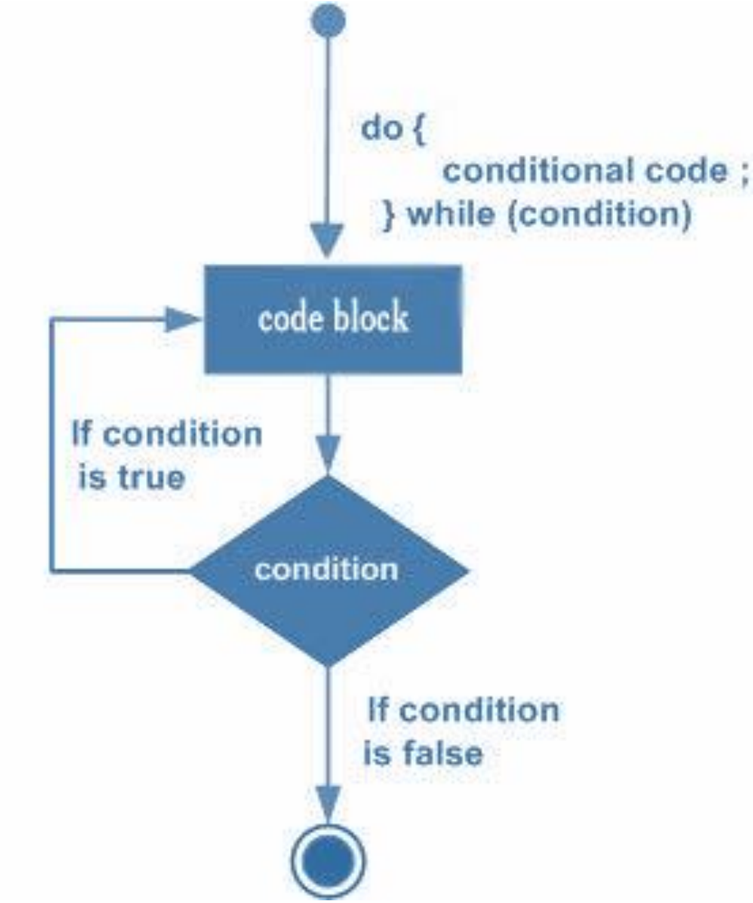
Output:

```
Output:

1
4
9
16
25
36
49
64
81
100
```

Do-While Loops

The **do-while** loop is used to iterate a piece of the program repeatedly until the predefined condition is valid.

Use the do-while loop if the number of iterations is not fixed.



Do-While Loops

Syntax:

```
initializer;
do {
 // statements to be executed
 increment/decrement;
} while(condition);
```

In a loop, if the condition is true, the code block executes repeatedly until the condition is false. Once false, the loop exits, and control moves to the next statement.

Do-While Loops

Example:

```
int i = 1;
do {
  System.out.println(i*i);
  i++;
} while(i<=10);
```

Output:

```
Output:

1
4
9
16
25
36
49
64
81
100
```

Implementing Loops in Java

Problem Statement:

You have been assigned a task to implement loops in Java.

Outcome:

By implementing loops in Java, you will learn how to efficiently manage repetitive tasks and automate processes within your programs. This skill will enable you to write cleaner, more efficient code and solve complex problems with ease.

> **Note:** Refer to the demo document for detailed steps: 05_Implementing_Loops_in_Java

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a folder on the Eclipse IDE
2. Create a program for the implementation of the loops in Java

Key Takeaways

- Java offers various types of operators, such as unary, arithmetic, relational, and logical.
- Lexical tokens make up a Java program.
- Identifiers can be used to name variables, labels, classes, and methods.
- Java provides single-line, multiple-line, and documentation comments.
- Java's conditional statements, including if, if-else, and switch, allow for better decision-making.
- Looping statements, such as for, while, and do-while, enable the repeated execution of the same code.