Lecture 1

PROGRAMMING (2)

BY: DR. MOHAMMED ABDELFATTAH

Course Objectives

- Review Essentials
- OOP Concepts
- Exception Handling
- Java IO Streams
- Packaging
- GUI
- Advanced Concepts:
  - Threads,
  - Processes.
- … etc.

Lecture Objectives

- Review.
  - ✓ Brief information
  - ✓ How Java work?
  - ✓ Data types
  - ✓ Data casting
  - ✓ Operators.
  - ✓ User Output & Input
  - ✓ IF Statements
  - ✓ Nested IF
  - ✓ Ternary Operator
  - ✓ Nested Ternary Operator
  - ✓ Switch Statements
  - ✓ Loops Statements
  - ✓ Break and Continue
  - ✓ Nested Loops
  - ✓ For Each Loop
  - ✓ Method
  - ✓ Method: Parameters
  - ✓ Method: Recursion
  - ✓ Method: Types
  - ✓ Scope
  - ✓ Examples

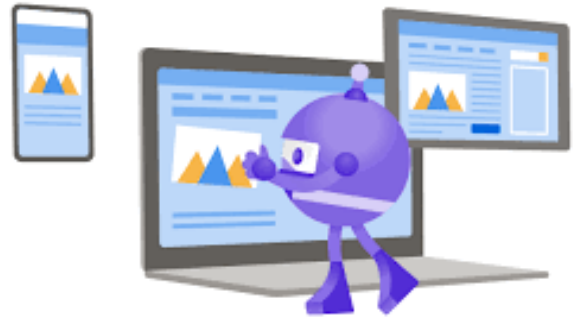
What is Java?

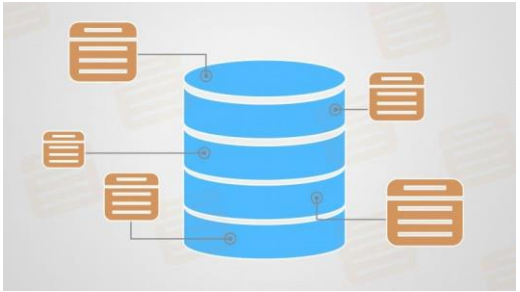
Java is a popular programming language, created in 1995.

It is owned by Oracle, and more than 3 billion devices run Java.

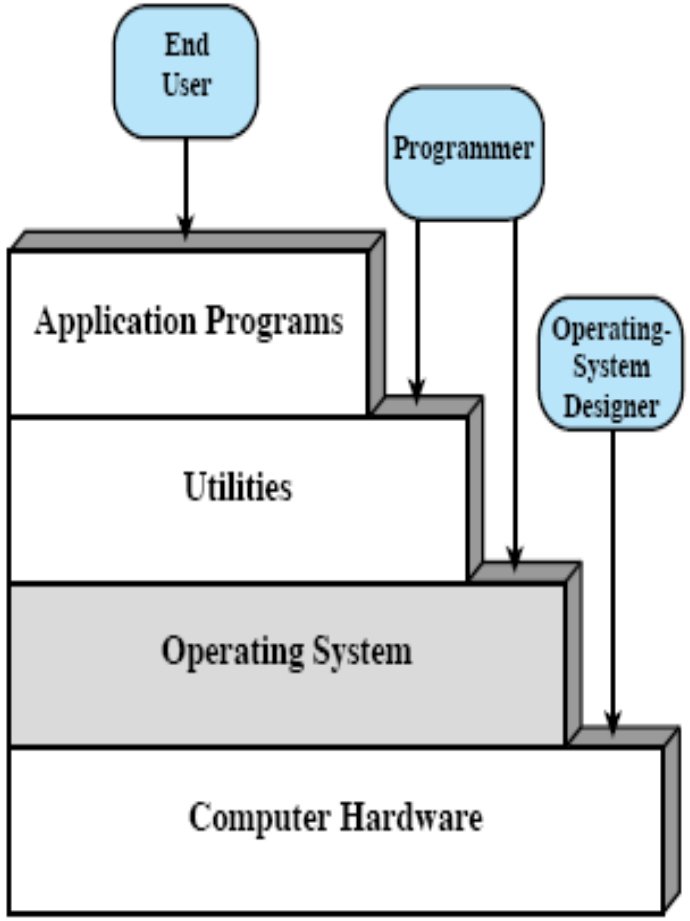
It is used for:

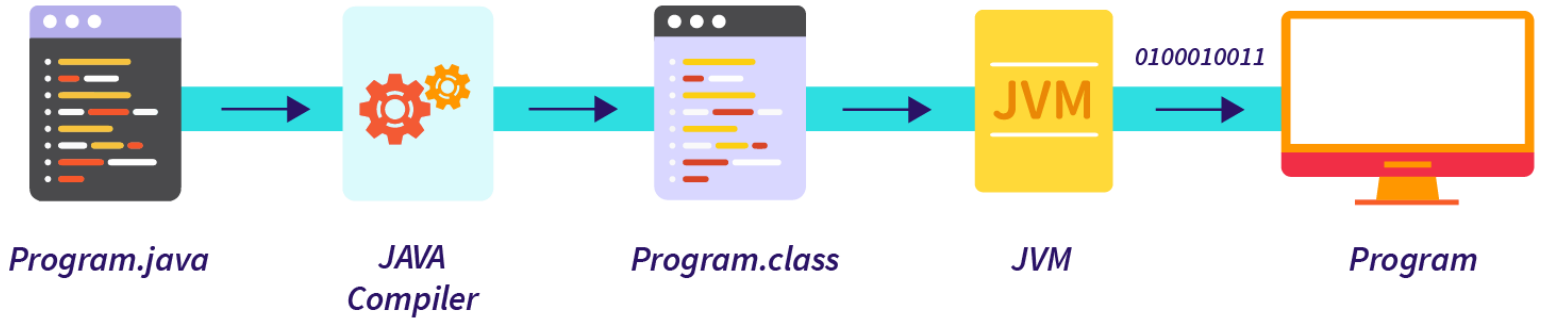
- Mobile applications (specially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection
- And much, much more!



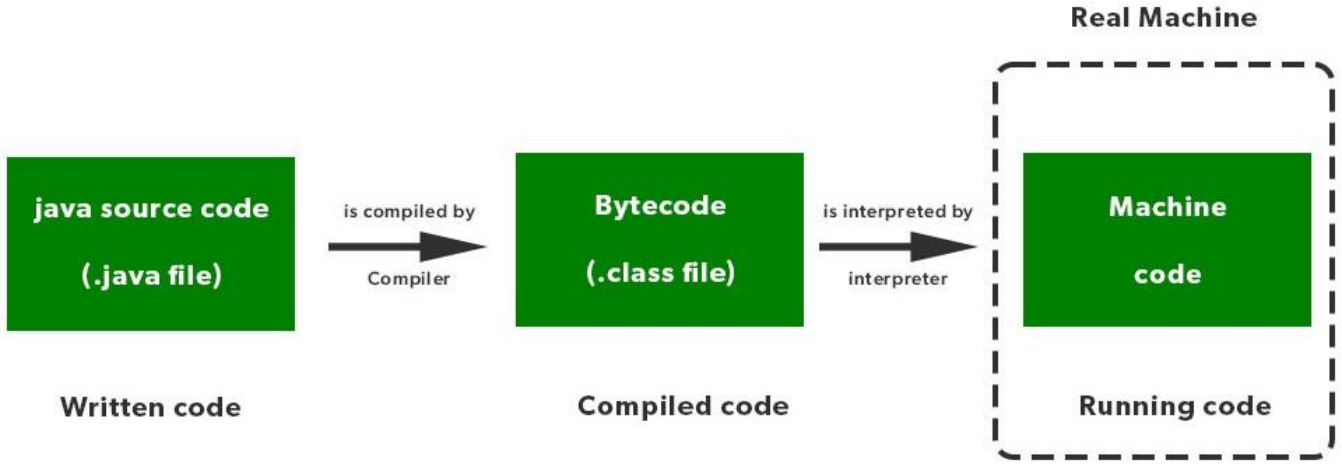


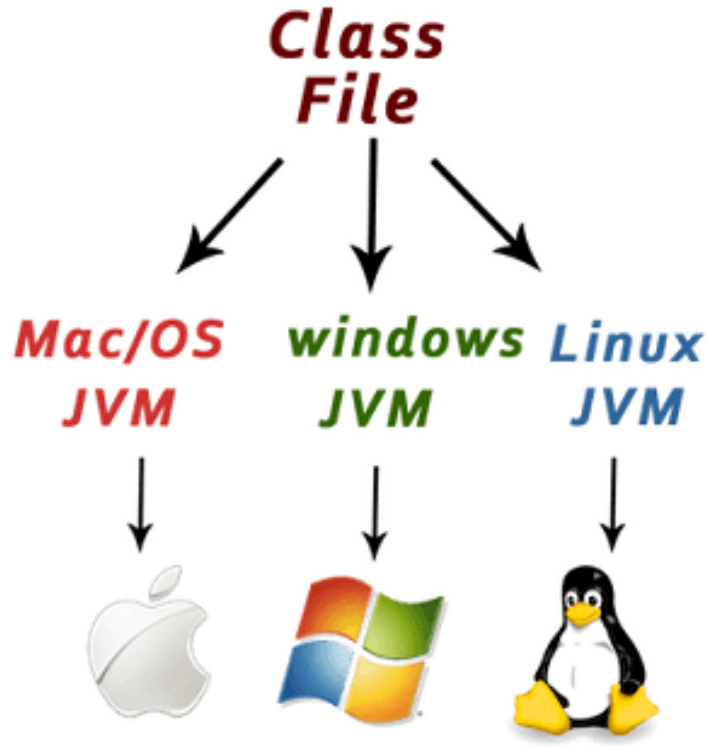
How Java Work?



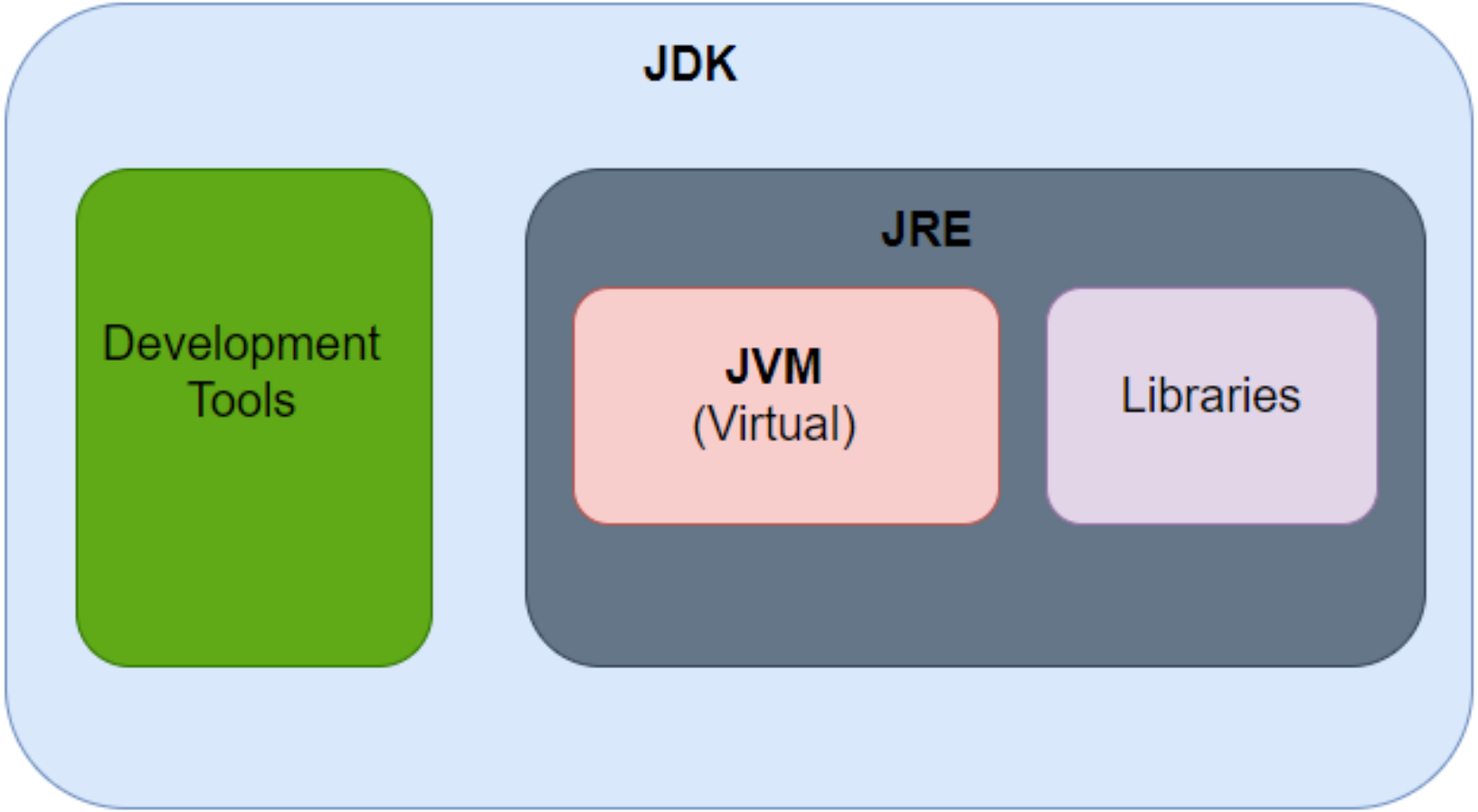


How Java Work?





How Java Work?



Java Install

- Some PCs might have Java already installed.

To check if you have Java installed on a Windows PC, search in the start bar for Java or type the following in Command Prompt (cmd.exe):

C:\Users*Your Name*>java -version

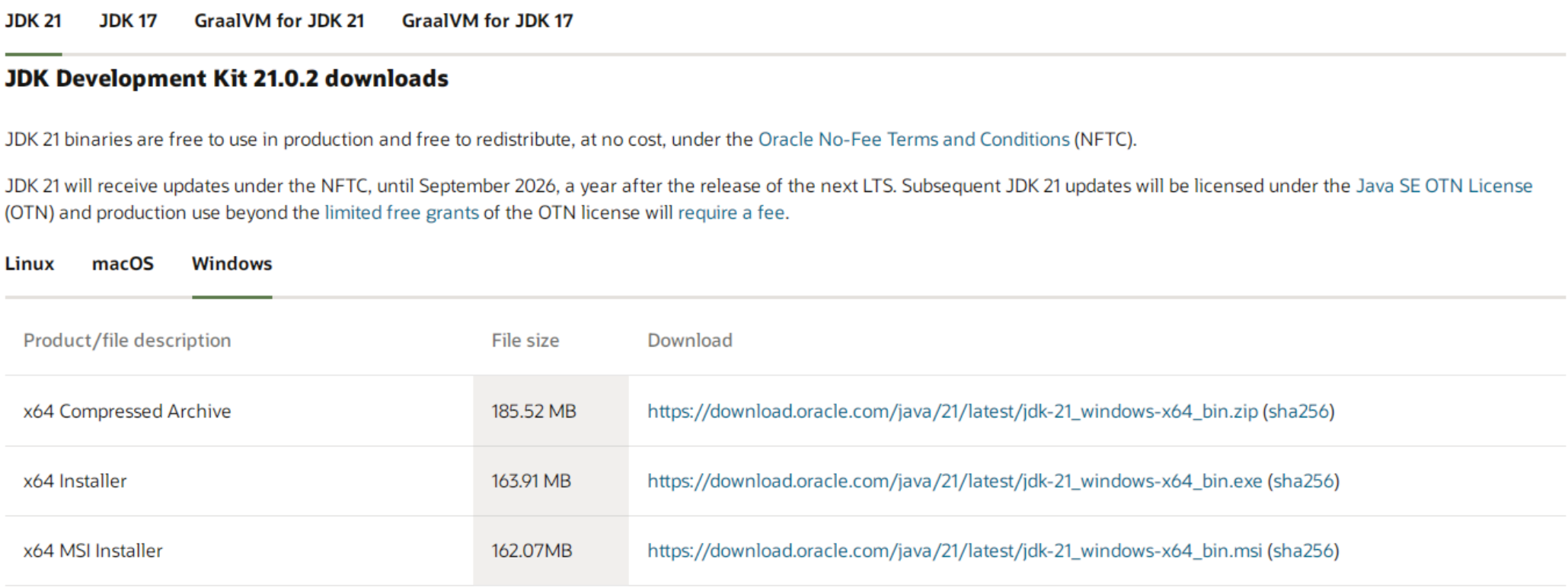
- If Java is installed, you will see something like this (depending on version):

java version "11.0.1" 2018-10-16 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)

- If you do not have Java installed on your computer, you can download it for free at oracle.com.

Java Install

Go to: https://www.oracle.com/java/technologies/downloads/#jdk21-windows



JDK 21 JDK 17 GraalVM for JDK 21 GraalVM for JDK 17

JDK Development Kit 21.0.2 downloads

JDK 21 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions (NFTC).

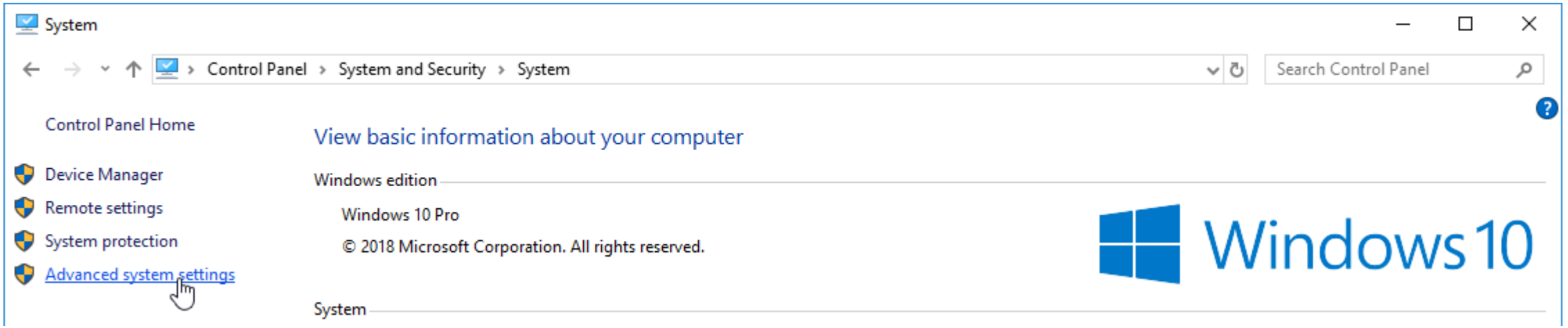
JDK 21 will receive updates under the NFTC, until September 2026, a year after the release of the next LTS. Subsequent JDK 21 updates will be licensed under the Java SE OTN License (OTN) and production use beyond the limited free grants of the OTN license will require a fee.

Linux macOS Windows

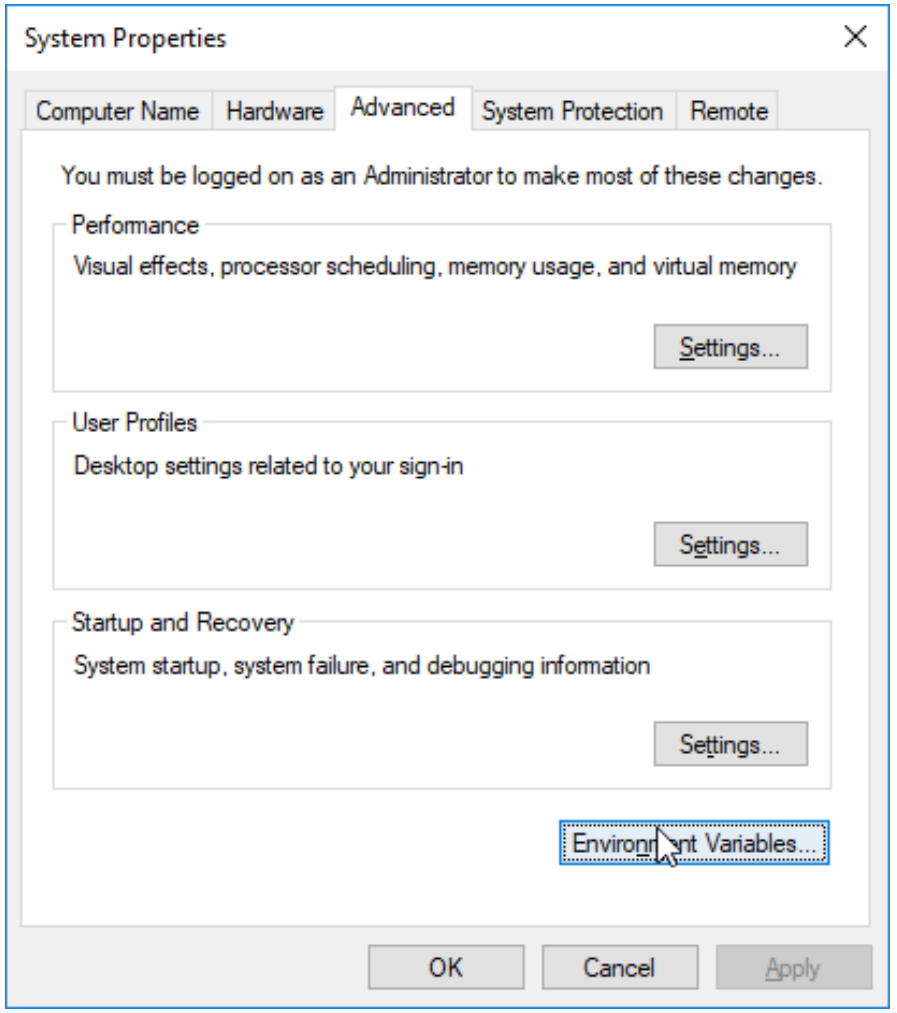
Product/file description	File size	Download
x64 Compressed Archive	185.52 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.zip (sha256)
x64 Installer	163.91 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.exe (sha256)
x64 MSI Installer	162.07MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.msi (sha256)

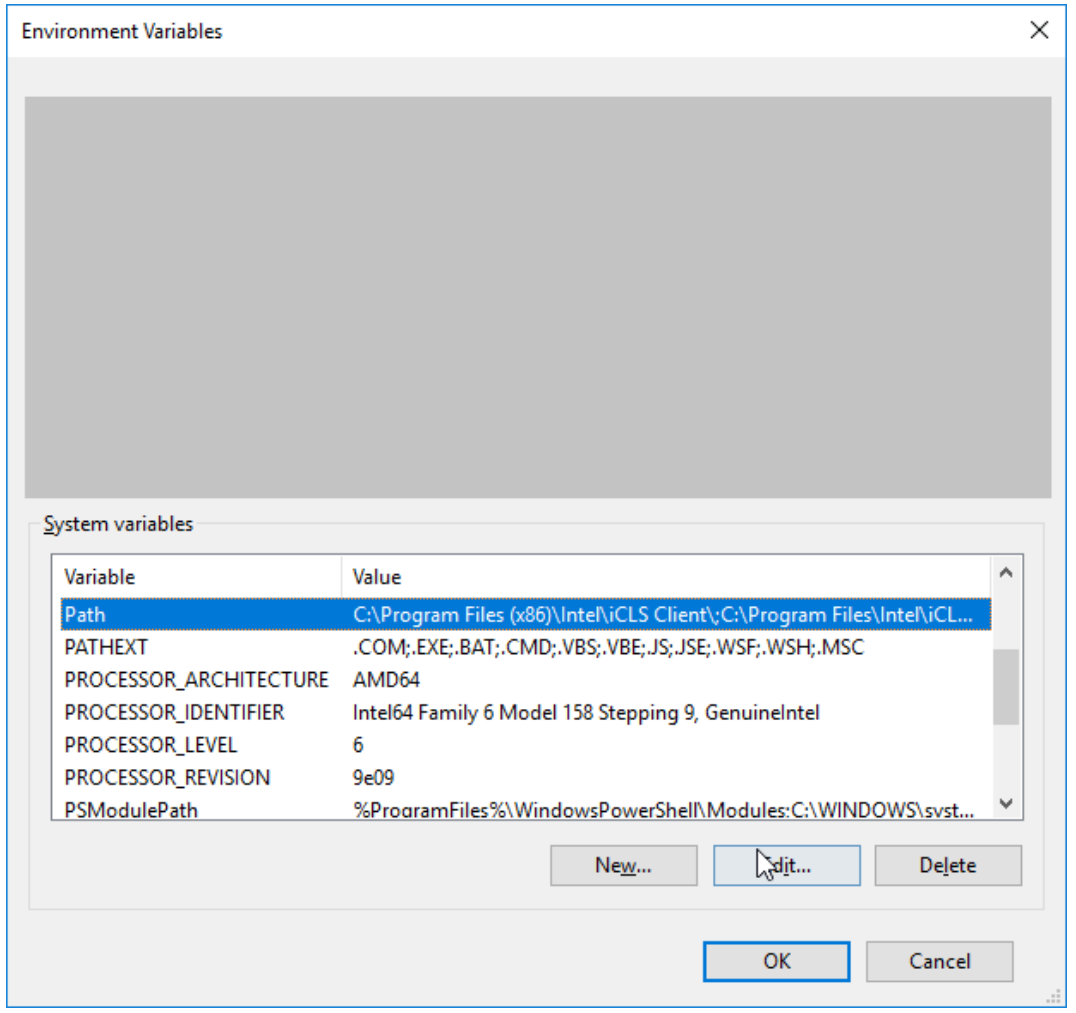
Setup for Windows

To install Java on Windows:

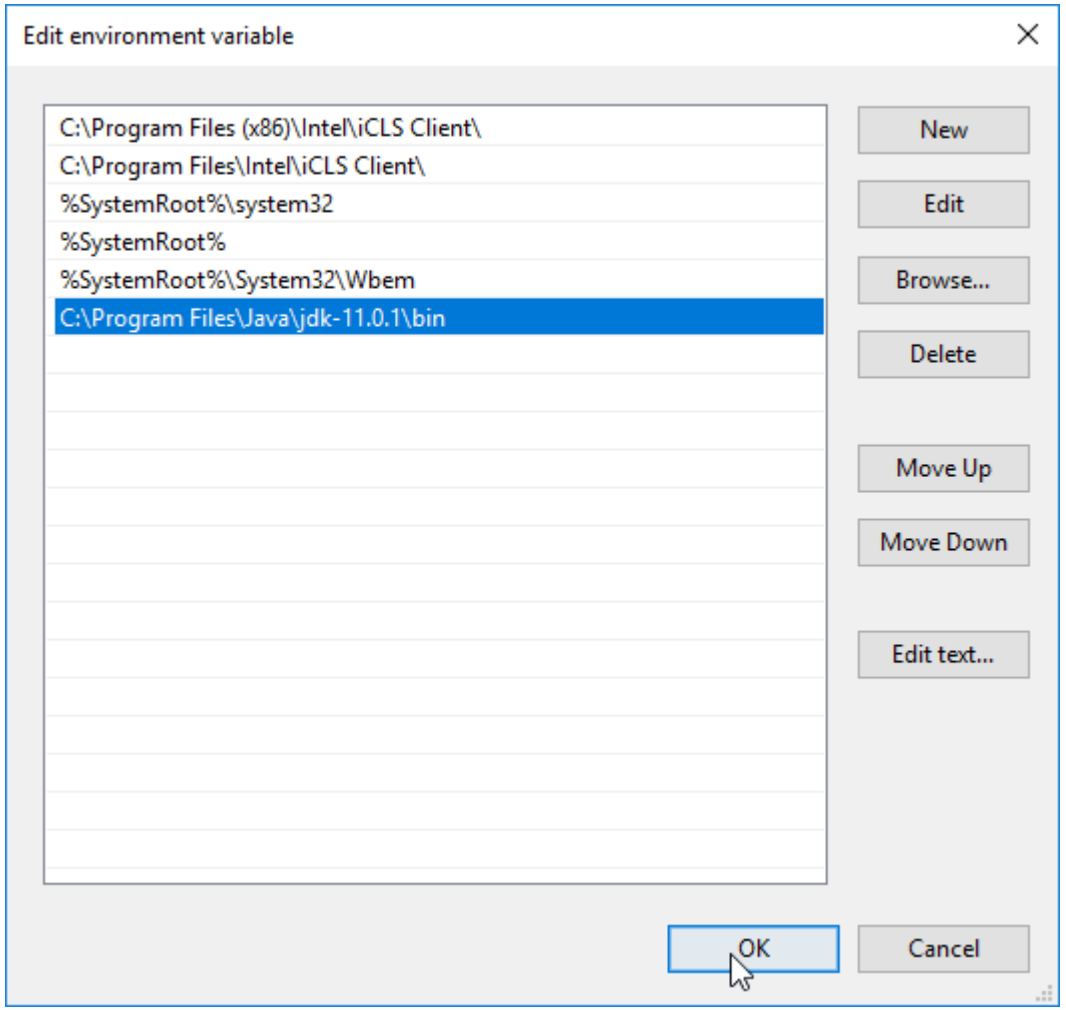


Setup for Windows





Setup for Windows



Write the following in the command line (cmd.exe):

C:\Users*Your Name*>java -version

If Java was successfully installed, you will see something like this (depending on version):

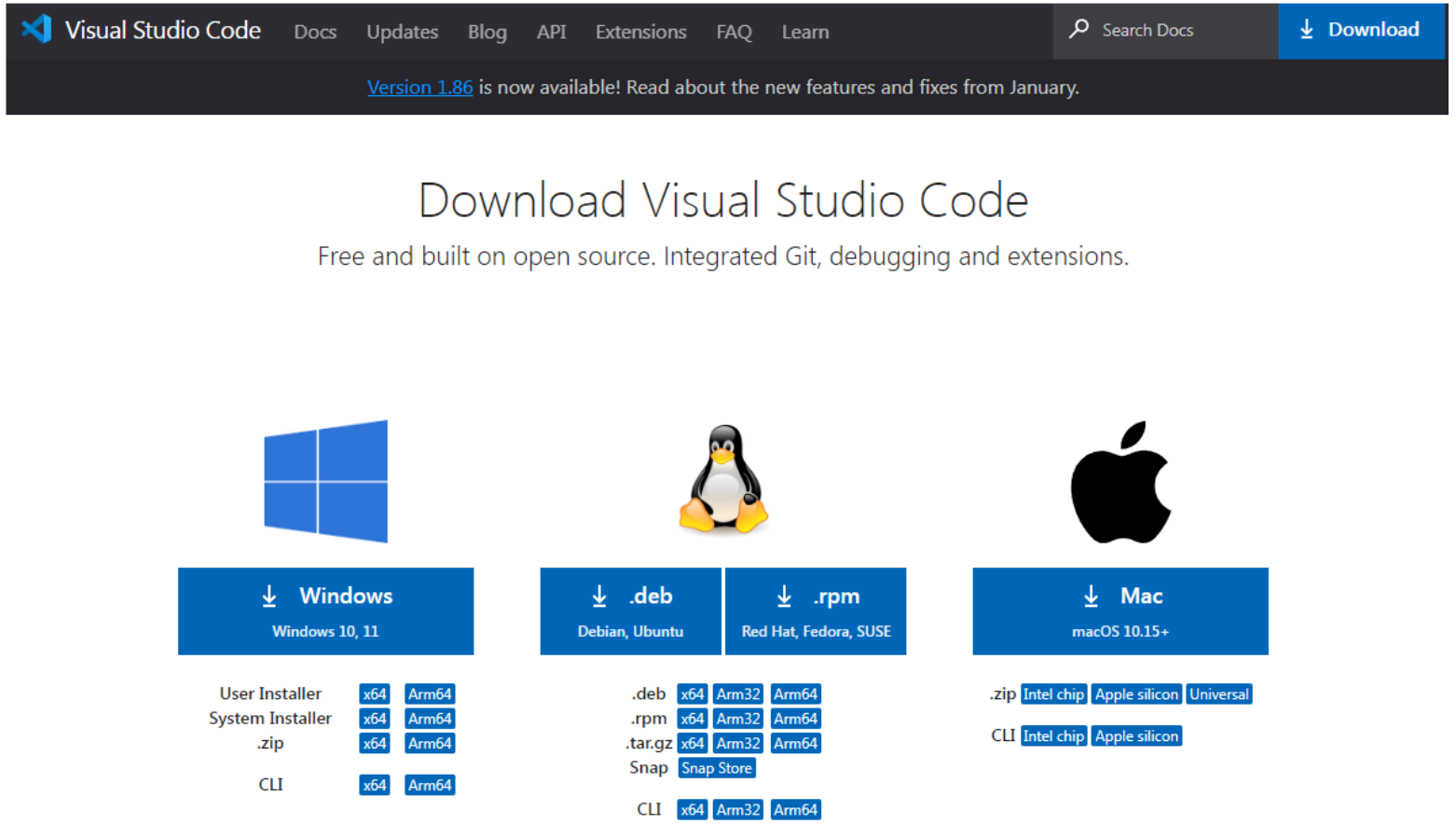
```
java version "11.0.1" 2018-10-16 LTS
Java(TM) SE Runtime Environment 18.9 (build
11.0.1+13-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build
11.0.1+13-LTS, mixed mode)
```

Integrated Development Environment (IDE)

It is possible to write Java in an Integrated Development Environment, such as IntelliJ IDEA, Netbeans or Eclipse, which are particularly useful when managing larger collections of Java files.

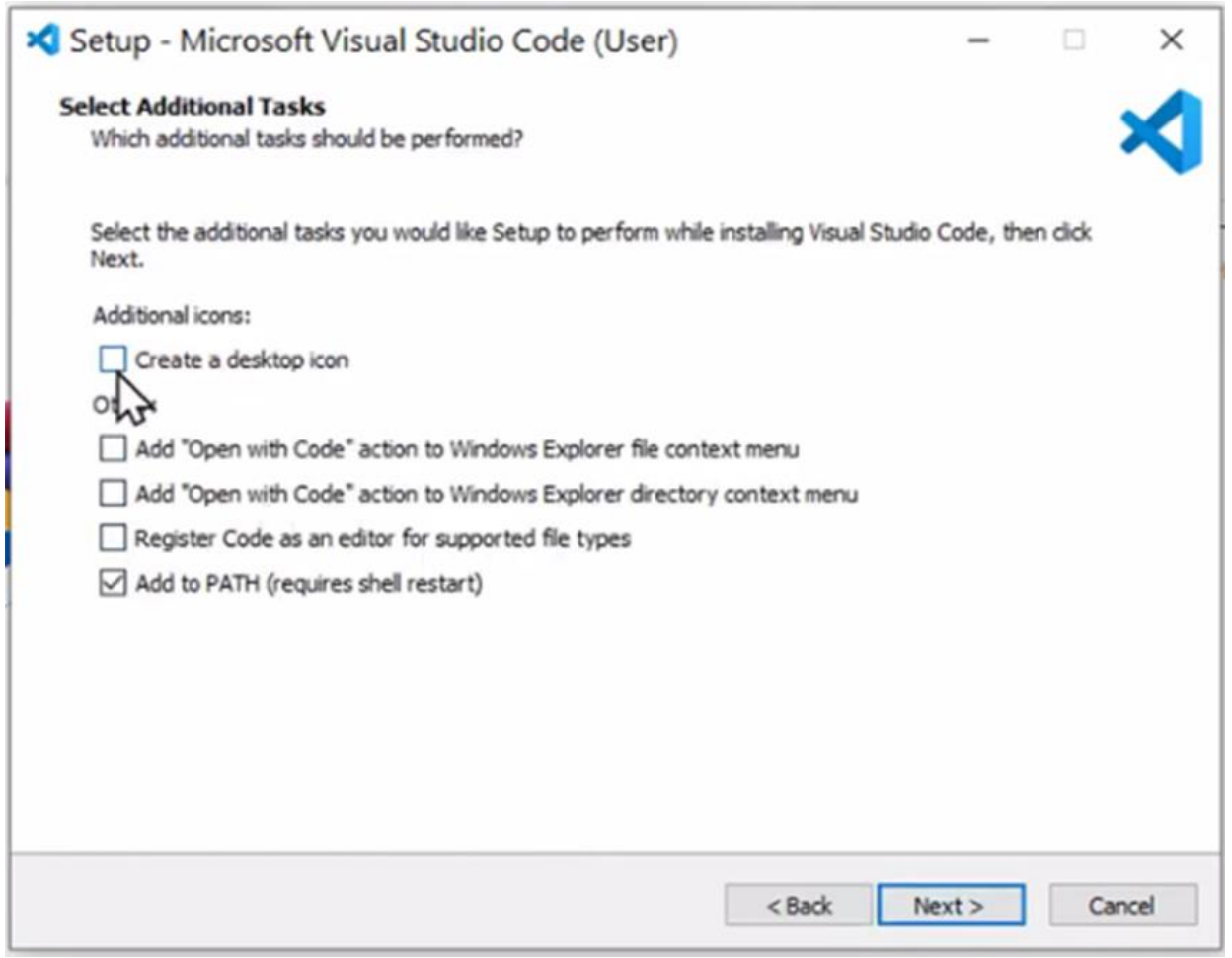
Setup Visual Studio Code

Go to: https://code.visualstudio.com/download

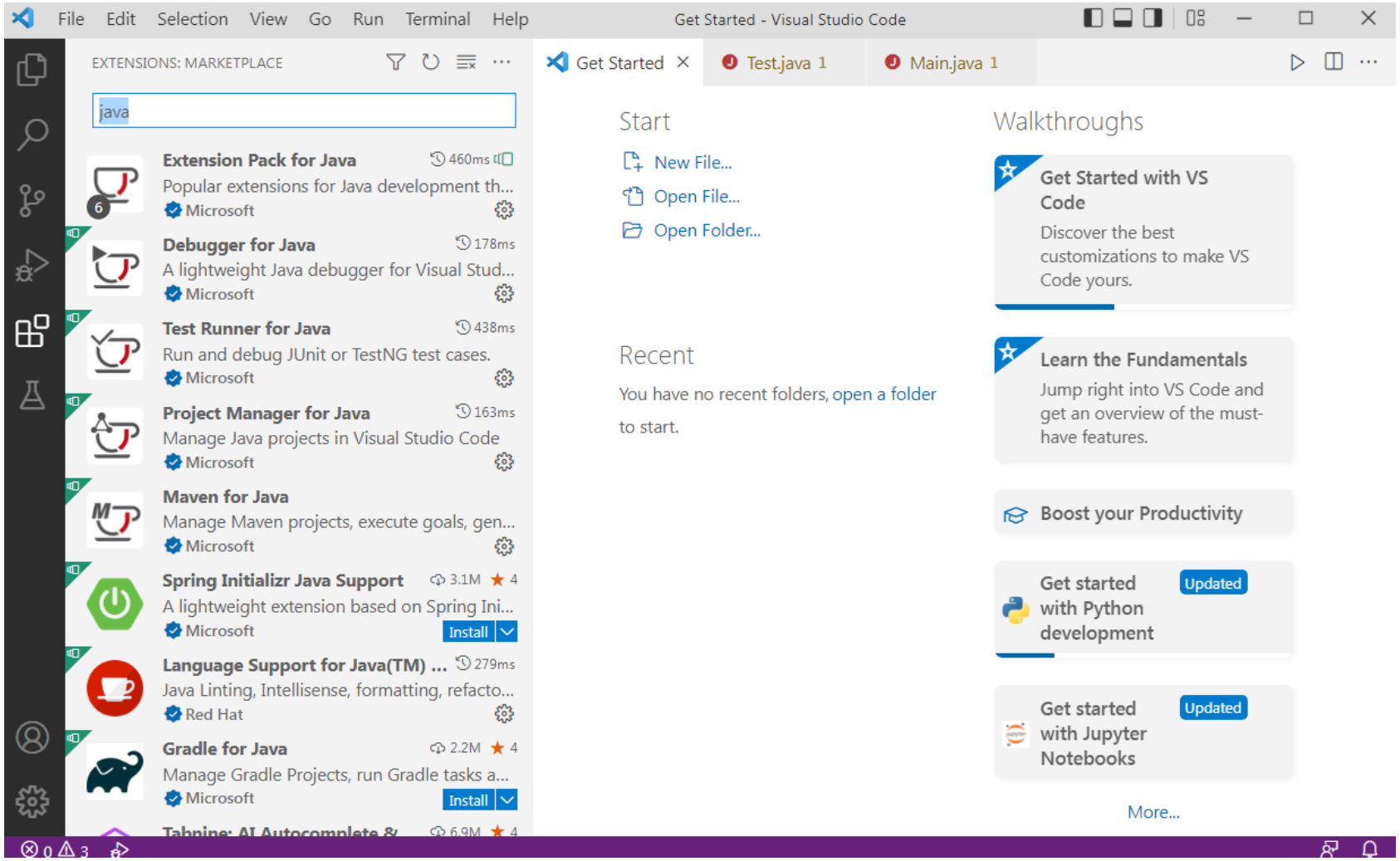


Setup Visual Studio Code

Note: Add to path.



Setup Visual Studio Code



Java Applications

In Java, every application begins with a class name, and that class must match the filename.

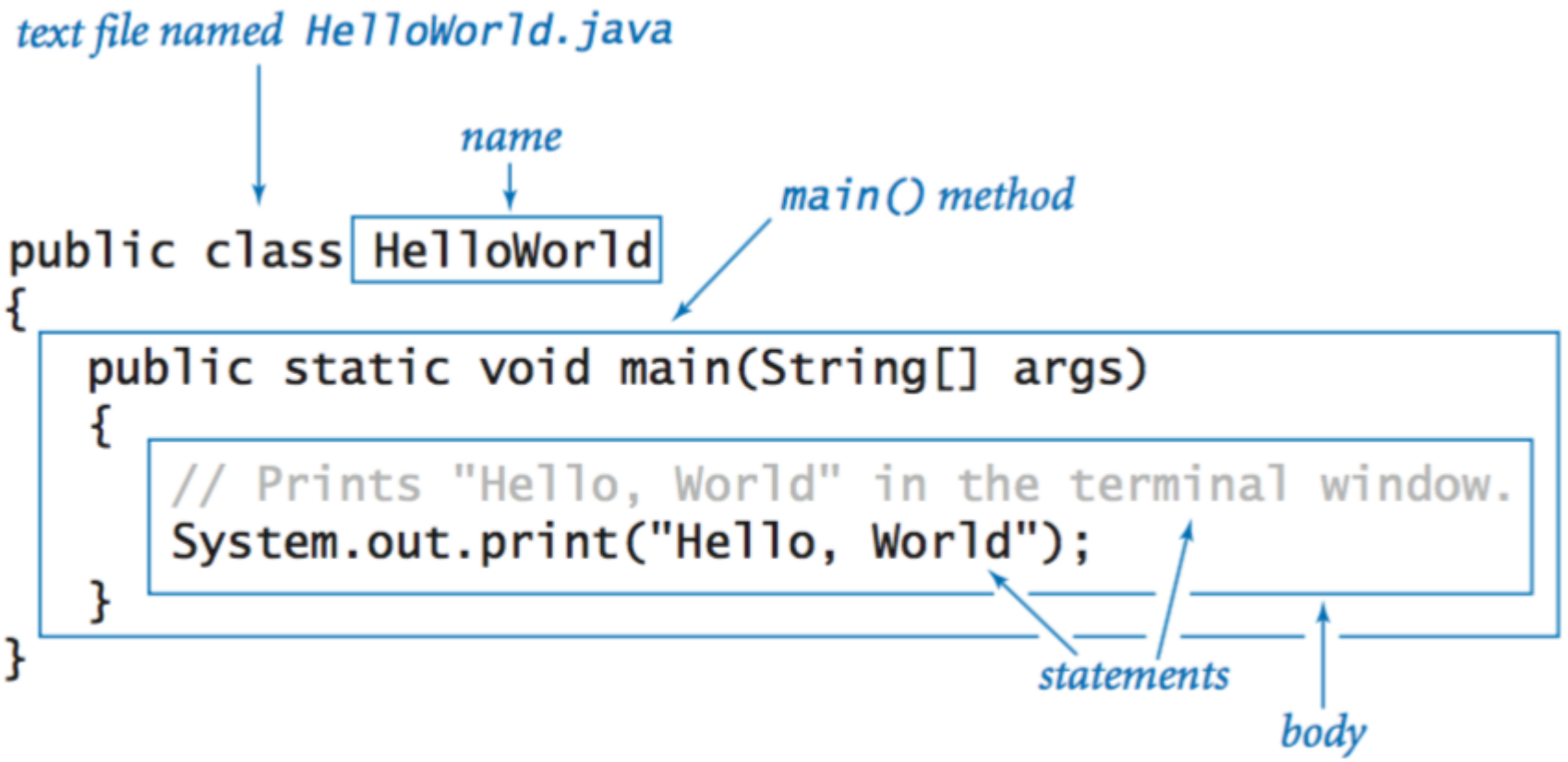
Let's create our first Java file, called Test.java, which can be done in any text editor (like Notepad).

The file should contain a "Hello World" message, which is written with the following code:

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

```
C:\Users\Your Name>javac Test.java

C:\Users\Your Name>java Test
```

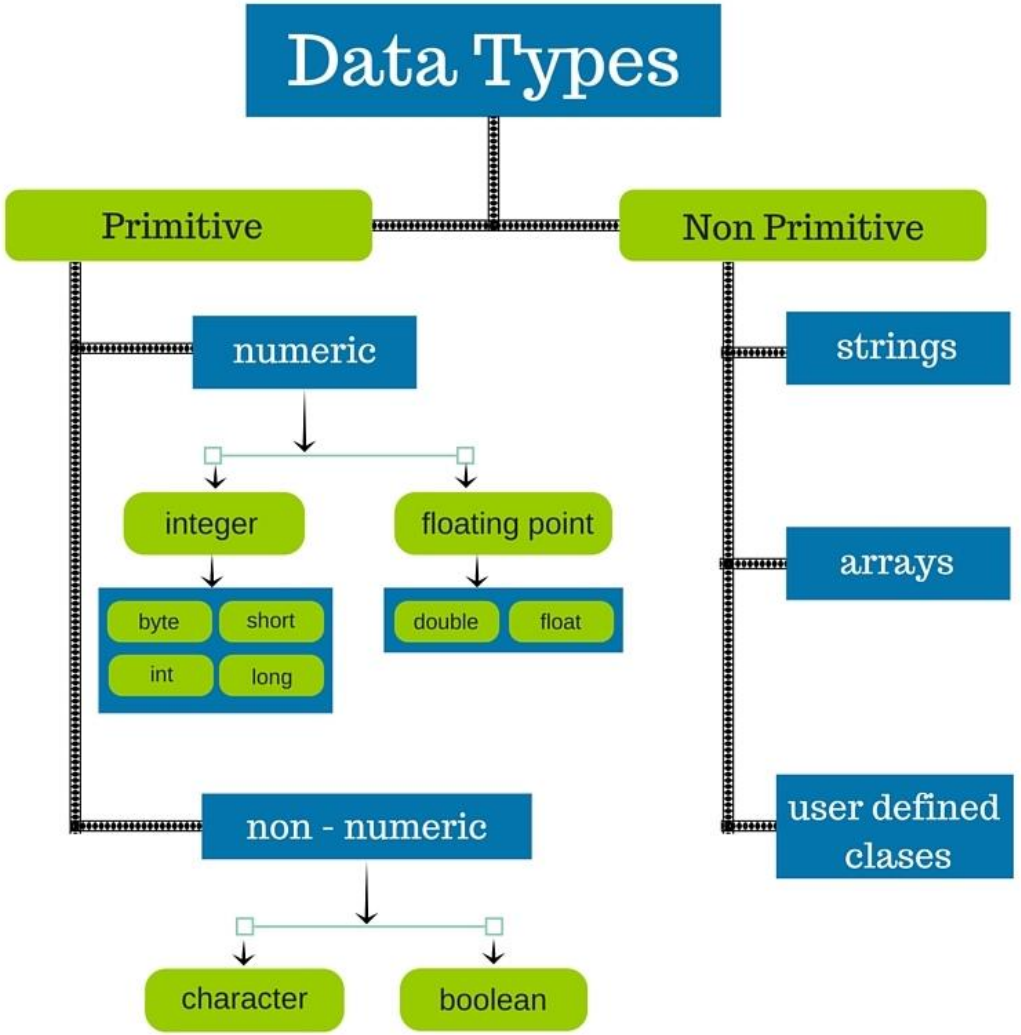
Java Syntax


text file named HelloWorld.java
name
main() method
public class HelloWorld
{
public static void main(String[] args)
{
// Prints "Hello, World" in the terminal window.
System.out.print("Hello, World");
}
}
statements
body

Java Comments

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

        // This is a comment
        System.out.println("Ali");

        System.out.println("Ahmed"); // This is a comment

        /* The code below will print the words Hello World
        to the screen, and it is amazing */
        System.out.println("University");



    }
}
```

Data Types


Data Types
Primitive
Non Primitive
numeric
integer
floating point
byte
short
int
long
double
float
non - numeric
character
boolean
strings
arrays
user defined clases

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods. There are eight primitive data types in Java:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Java Variables

Variables are containers for storing data values. In Java, there are different types of variables, for example:

- String - stores text, such as "Hello". String values are surrounded by double quotes
- int - stores integers (whole numbers), without decimals, such as 123 or -123
- float - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- boolean - stores values with two states: true or false

Java Variables

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

        String name = "John";
        System.out.println(name);

        int myNum1 = 15;
        System.out.println(myNum1);

        final int myNum2 = 15;
        myNum2 = 20;  // will generate an error: cannot assign a value to a final variable
        System.out.println(myNum2);

    }
}
```

Final Variables: If you don't want others (or yourself) to overwrite existing values, use the final keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only).

Other Types

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

        int myNum = 5;
        float myFloatNum = 5.99f;
        char myLetter = 'D';
        boolean myBool = true;
        String myText = "Hello";

        System.out.println(myNum);
        System.out.println(myFloatNum);
        System.out.println(myLetter);
        System.out.println(myBool);
        System.out.println(myText);

    }
}
```

Print Variables

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

        String name = "John";
        System.out.println("Hello " + name);

        String firstName = "John ";
        String lastName = "Doe";
        String fullName = firstName + lastName;
        System.out.println(fullName);

        int x = 5;
        int y = 6;
        System.out.println(x + y); // Print the value of x + y

    }
}
```

Declare Multiple Variables

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

        int x = 5, y = 6, z = 50;
        System.out.println(x + y + z);

        int x1, y1, z1;
        x1 = y1 = z1 = 50;
        System.out.println(x1 + y1 + z1);

    }
}
```

Identifiers

All Java variables must be identified with unique names. These unique names are called identifiers.

The general rules for naming variables are:

- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like Java keywords, such as int or boolean) cannot be used as names

```
public class Test {
    public static void main(String[] args) {
        // Good
        int minutesPerHour = 60;
        System.out.println(minutesPerHour);

    }
}
```

Characters Data Type

The char data type is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c‘.

Alternatively, if you are familiar with ASCII values, you can use those to display certain characters:

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

        char myGrade = 'B';
        System.out.println(myGrade);
        System.out.println("--------------");


        char myVar1 = 65, myVar2 = 66, myVar3 = 67;
        System.out.println(myVar1);
        System.out.println(myVar2);
        System.out.println(myVar3);

    }
}
```

```
B

--------------

A

B

C
```

Non-Primitive Data Types

Non-primitive data types are called reference types because they refer to objects.

The main difference between primitive and non-primitive data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.

Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc. You will learn more about these in a later lectures.

Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

- Widening Casting (automatically) - converting a smaller type to a larger type size

  byte -> short -> char -> int -> long -> float -> double

- Narrowing Casting (manually) - converting a larger type to a smaller size type

  double -> float -> long -> int -> char -> short -> byte

Widening Casting

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

        int myInt = 9;
        double myDouble = myInt; // Automatic casting: int to double

        System.out.println(myInt);      // Outputs 9
        System.out.println(myDouble);   // Outputs 9.0

    }
}
```

Narrowing Casting

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

        double myDouble = 9.78d;
        int myInt = (int) myDouble; // Manual casting: double to int

        System.out.println(myDouble);   // Outputs 9.78
        System.out.println(myInt);      // Outputs 9

    }
}
```

Operators

Operators are used to perform operations on variables and values.

Java divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Assignment Operators

Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
^=	x ^= 3	x = x ^ 3

Comparison Operators

Comparison operators are used to compare two values (or variables).

The return value of a comparison is either true or false.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

You can also test for true or false values with logical operators. Logical operators are used to determine the logic between variables or values.

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Example 1

Let's think of a "real life example" where we need to find out if a person is old enough to vote.

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {

        int votingAge = 20;

        //int personAge = 25;

        System.out.print("Please enter your age: ");

        Scanner scReadAge = new Scanner(System.in);          // Create a Scanner object
        int personAge= scReadAge.nextInt();                  // Read user input


        System.out.println(personAge >= votingAge);

      }
}
```

User Input (Scanner)

The Scanner class is used to get user input, and it is found in the java.util package. To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation.

- Input Types

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

User Input (Scanner)

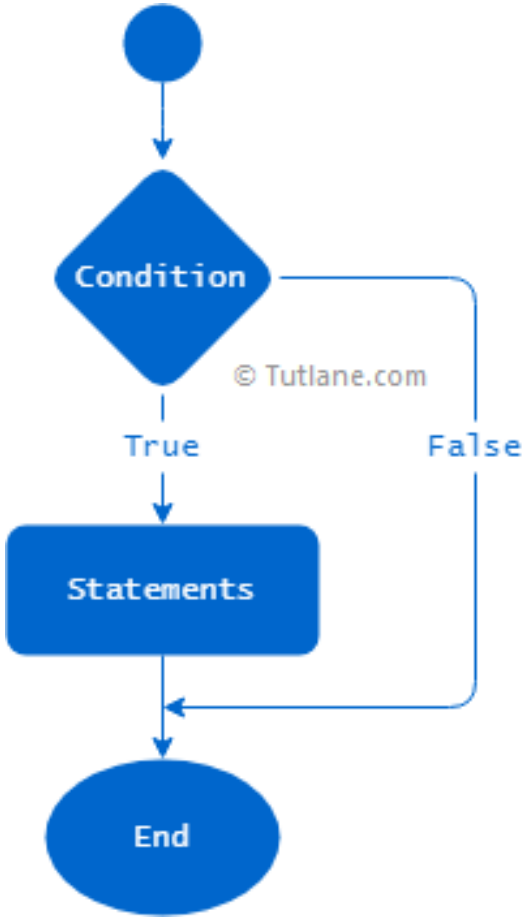
```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {

        Scanner myObj = new Scanner(System.in);

        System.out.println("Enter name, age and salary:");

        // String input
        String name = myObj.nextLine();

        // Numerical input
        int age = myObj.nextInt();
        double salary = myObj.nextDouble();

        // Output input by user
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Salary: " + salary);

    }
}
```

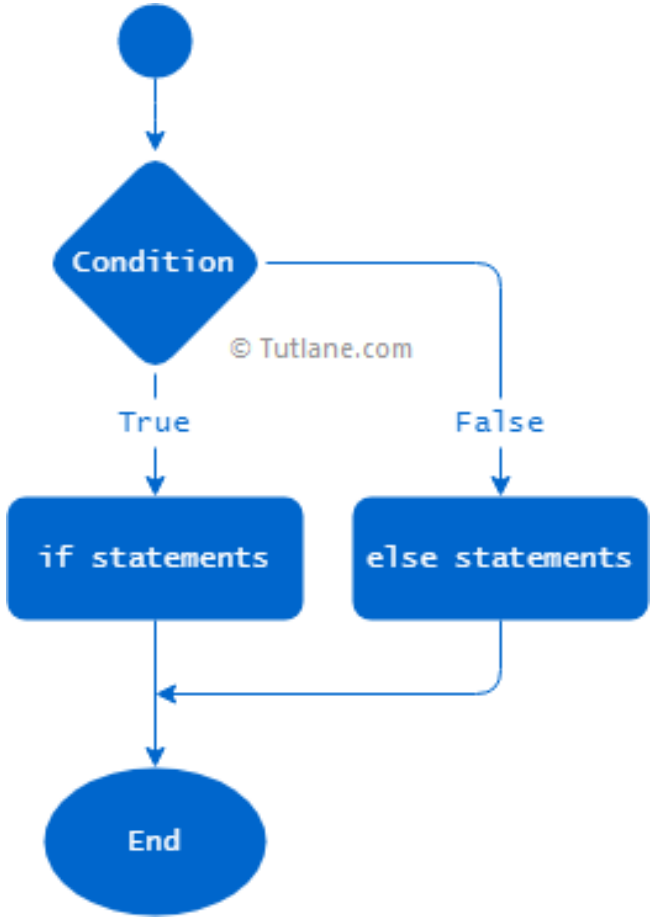
Example 2

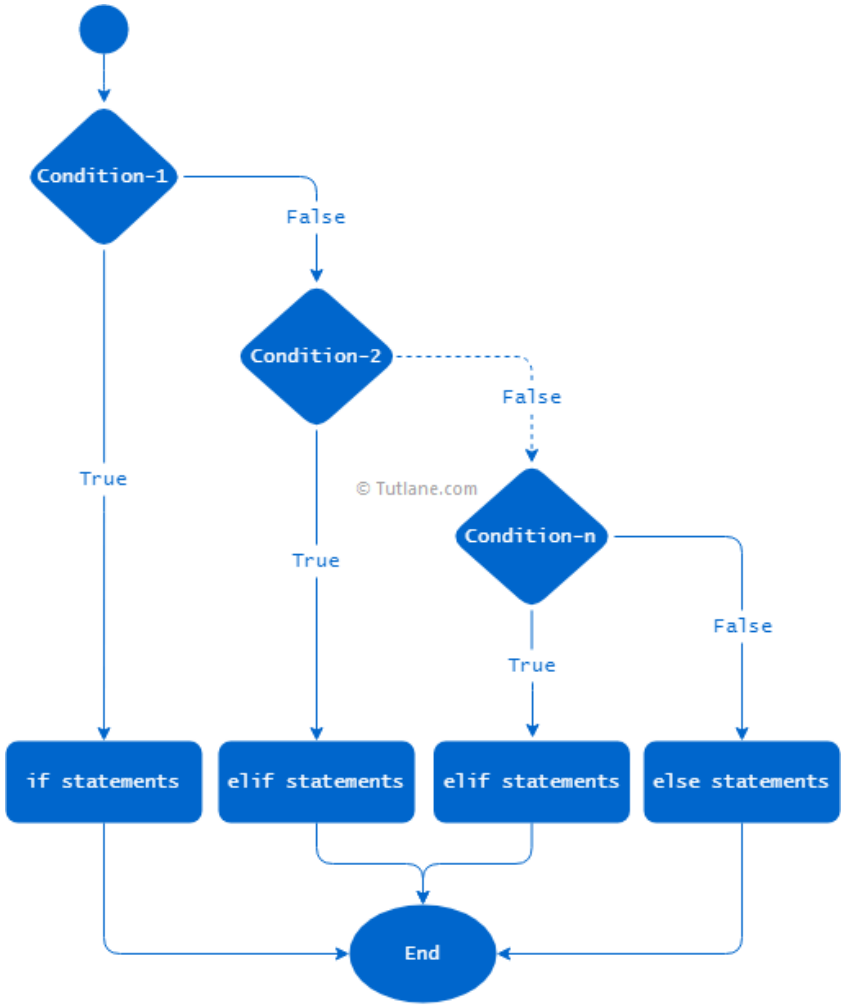
Using If condition:

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {

        int myAge = 25;
        int votingAge = 18;

        if (myAge >= votingAge) {
          System.out.println("Old enough to vote!");
        } else {
          System.out.println("Not old enough to vote.");
        }

    }
}
```

IF Statements







Example

```
public class Test{

    public static void main (String[] args){

        int time = 22;   // Using 24 format.


        if (time < 10) {
        System.out.println("Good morning.");
        }
        else if (time < 18) {
        System.out.println("Good day.");
        }
        else {
        System.out.println("Good evening.");
        }
        // Outputs "Good evening.
    }
}
```

Nested IF

```
public class Test{
    public static void main (String[] args){
        int x = 1010, y = 170, z = 169;
        if(x > y)
        {
            if(x > z)
            //prints x, if the above two conditions are true
            System.out.println("The largest number is: "+x);
            else
            //means x>y and x<z
            System.out.println("The largest number is: "+z);
        }
        else
        {
            if(y > z)
            //means z>x and y>z
            System.out.println("The largest number is: "+y);
            else
            //z>x and z>y
            System.out.println("The largest number is: "+z);
        }
    }
}
```

Ternary Operator

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

        if (time < 18)
        {
          System.out.println("Good day.");
        }
        else
        {
          System.out.println("Good evening.");
        }
    }
}
```

variable = (condition) ? expressionTrue : expressionFalse ;

```
public class Test
{
    public static void main (String[] args)
    {
        int time = 20;
        String result = (time < 18) ? "Good day." : "Good evening.";
        System.out.println(result);
    }
}
```

Nested Ternary Operator

```
Temp = a > b ? a : b ;

Largest = c > temp ? C : temp;
```

```
Result = c > (a>b ? a:b)  ?  C : ((a>b) ? a:b);
```

Switch Statements

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

        int day = 4;
        switch (day) {
          case 6:
            System.out.println("Today is Saturday");
            break;
          case 7:
            System.out.println("Today is Sunday");
            break;
          default:
            System.out.println("Looking forward to the Weekend");
        }
        // Outputs "Looking forward to the Weekend"
    }
}
```

The default keyword specifies some code to run if there is no case match:

Loops Statements: While / Do-While / For

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

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

    }
}
```

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

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

```java
public class Test {
    public static void main (String[] args) {
      int i = 0;
      do {
        System.out.println(i);
        i++;
      }
      while (i < 5);
    }
}
```

Break and Continue

- The break statement can also be used to jump out of a loop.

- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

- You can also use break and continue with loops.

Example

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

      for (int i = 0; i < 10; i++) {
        if (i == 4) {
          break;
        }
        System.out.println(i);
      }

    }
}
```

```
// Output
0
1
2
3
```

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

      for (int i = 0; i < 10; i++) {
        if (i == 4) {
          continue;
        }
        System.out.println(i);
      }

    }
}
```

```
// Output
0
1
2
3
5
6
7
8
9
```

Nested Loops

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still have to point to the two indexes).

```
public class Test
{
    public static void main (String[] args)
    {
        int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
        for (int i = 0; i < myNumbers.length; ++i)
        {
            for(int j = 0; j < myNumbers[i].length; ++j)
            {
              System.out.println(myNumbers[i][j]);
            }
        }
    }
}
```

	Column 0	Column 1	Column 2
Row 0	a[0][0]	a[0][1]	a[0][2]
Row 1	a[1][0]	a[1][1]	a[1][2]
Row 2	a[2][0]	a[2][1]	a[2][2]
Row 3	a[3][0]	a[3][1]	a[3][2]
Row 4	a[4][0]	a[4][1]	a[4][2]

```
// Output
1
2
3
4
5
6
7
```

For Each Loop

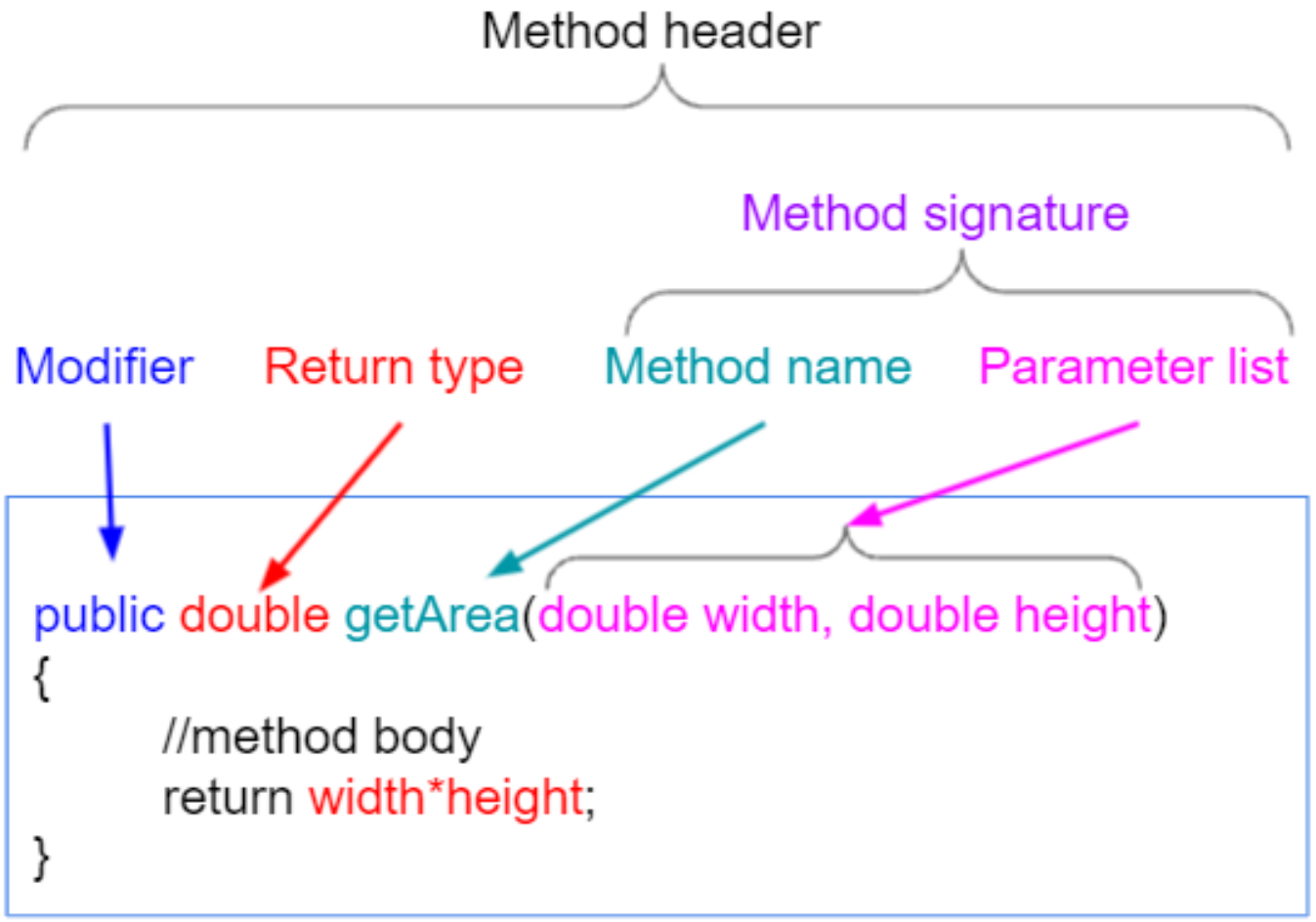
```
public class Test {
    public static void main (String[] args) {

      String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

      for (String i : cars)
      {
        System.out.println(i);
      }
    }
}
```

```
// Output
Volvo
BMW
Ford
Mazda
```

Methods

- A method is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as functions.
- Why use methods? To reuse code: define the code once, and use it many times.
- A method must be declared within a class.

Method


Method header
Method signature
Modifier
Return type
Method name
Parameter list
public double getArea(double width, double height)
{
//method body
return width*height;
}

Example

```
public class Test
{
    public static void main (String[] args)
    {
        myMethod();
    }

    static void myMethod()
    {
        System.out.println("I have been executed!");
    }
}
```

// Output
I have been executed!

Method Parameters

```
public class Test
{
    public static void main (String[] args)
    {
        myMethod("Ali", 5);
        myMethod("Sara", 8);
        myMethod("Mohammed", 31);
    }

    static void myMethod(String fname, int age) {
        System.out.println(fname + " is " + age);
      }
}
```

```
// Output

Ali is 5

Sara is 8

Mohammed is 31
```

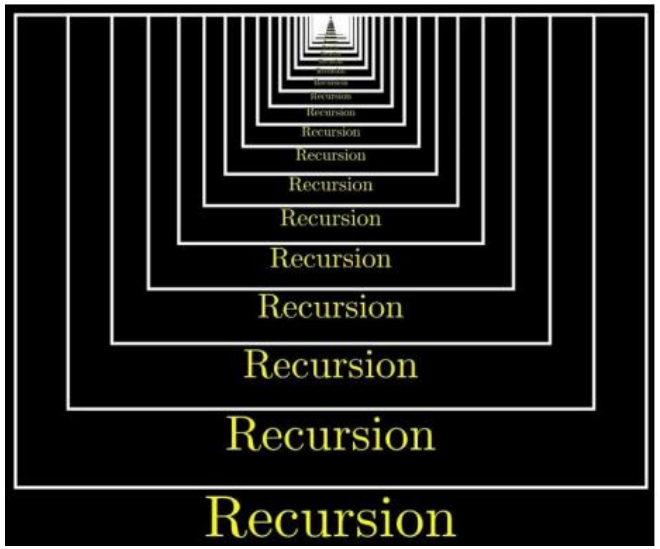
Scope

```
public class Test
{
    public static void main (String[] args)
    { // This is a block

        // Code here CANNOT use x

        int x = 100;

        // Code here can use x
        System.out.println(x);

    }  // The block ends here

        // Code here CANNOT use x
}
```

Recursion

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

```
10 + sum(9)
10 + ( 9 + sum(8) )
10 + ( 9 + ( 8 + sum(7) ) )
...
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
```



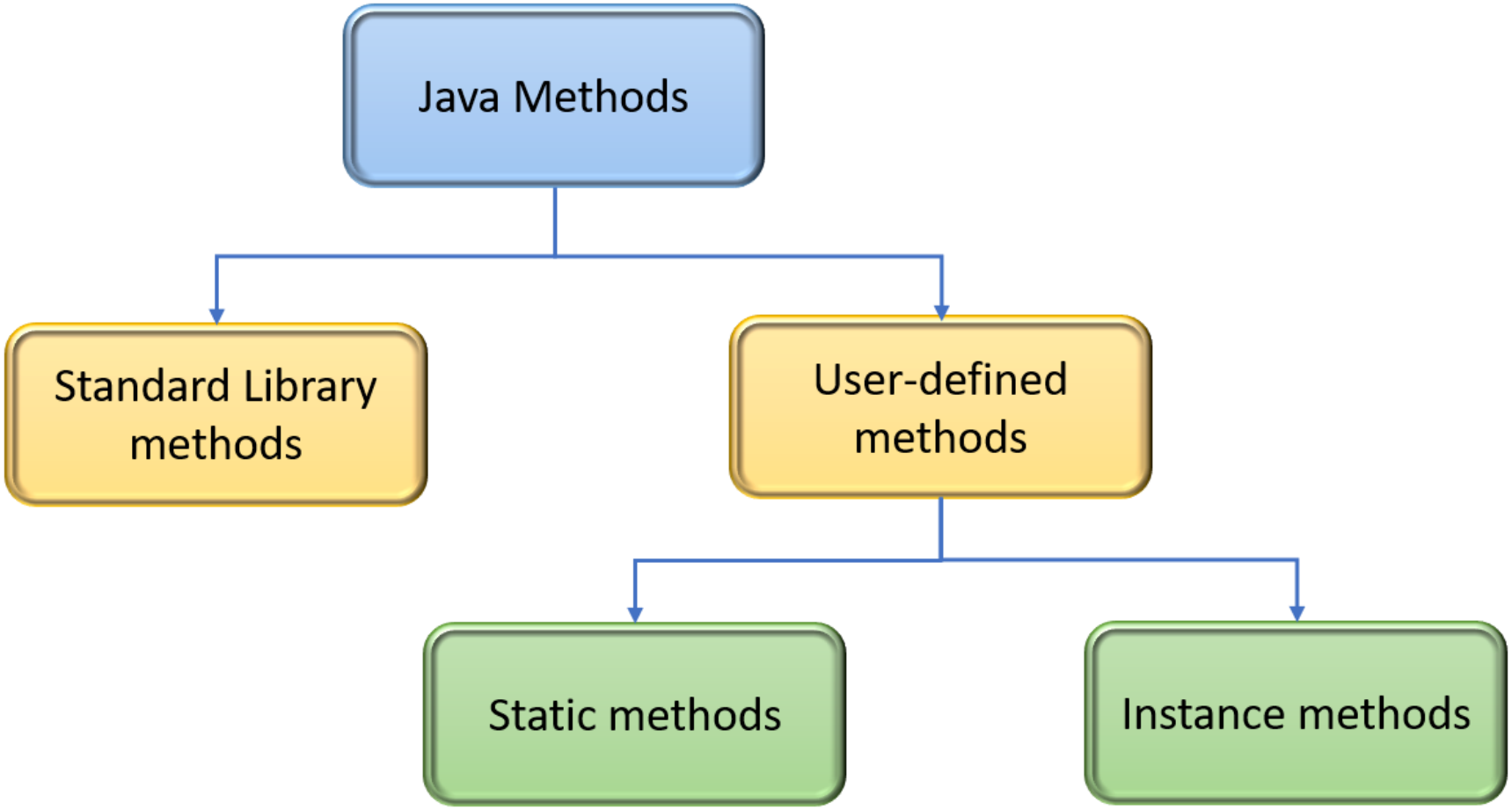
- When the sum() function is called, it adds parameter k to the sum of all numbers smaller than k and returns the result. When k becomes 0, the function just returns 0.
- Since the function does not call itself when **k** is **0**, the program **stops** there and returns the result.

Example

```java
public class Test
{
    public static void main (String[] args)
    {
        int result = sum(10);
        System.out.println(result);
    }

    public static int sum(int k)
    {
        if (k > 0) {
          return k + sum(k - 1);
        } else {
          return 0;
        }
    }
}
```

```
// Output
55
```

Methods Types



Thanks

References: https://www.w3schools.com