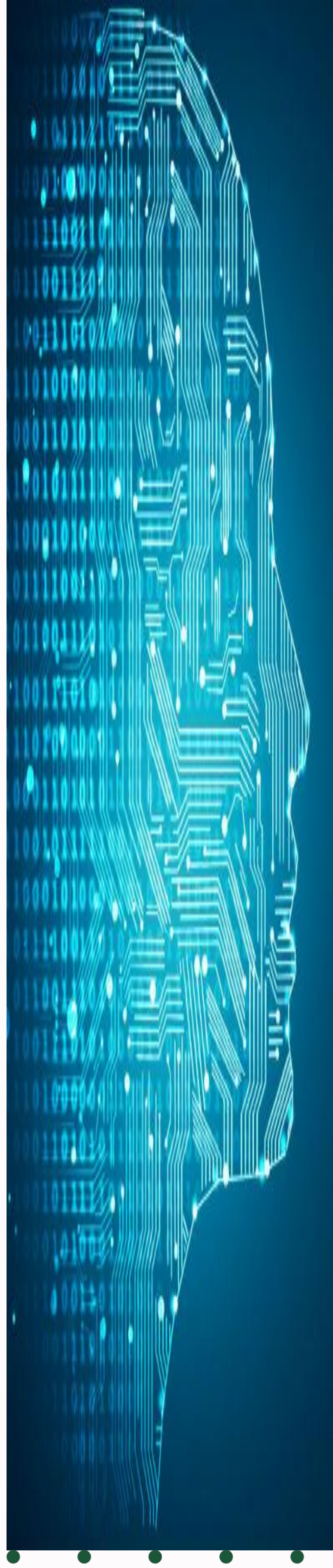
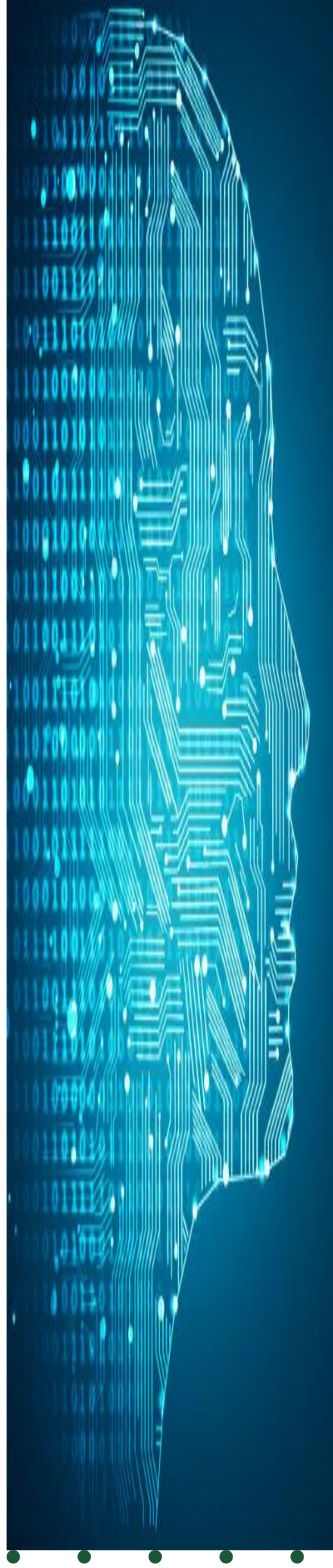
OBJECT-ORIENTED PROGRAMMING 1

- JAVA LANGUAGE



Topic Flow

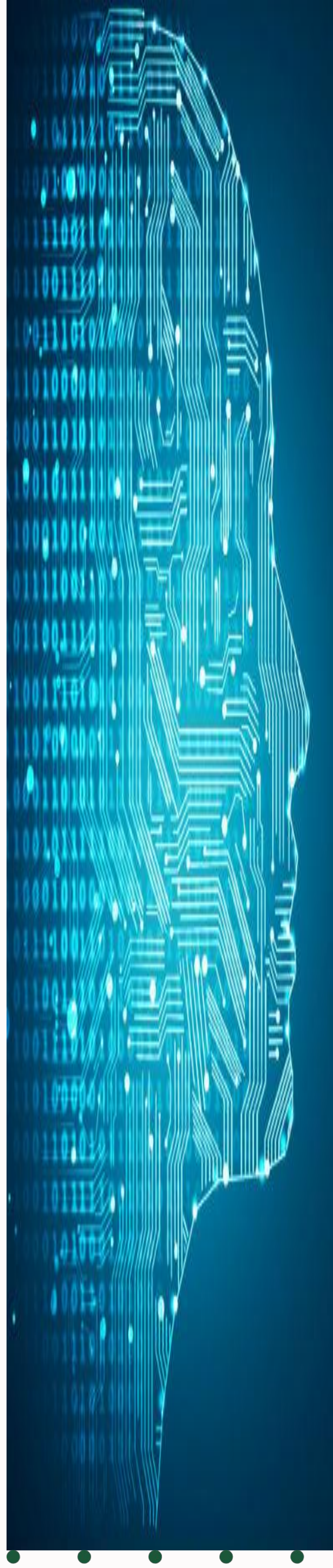
- Java Introduction
- Basic Syntax
- Data Types
- Operators



What is Java?

- Java is a popular high-level, class-based object-oriented programming language originally developed by Sun Microsystems and released in 1995.

- is a general-purpose programming language intended to let programmers **Write Once and run Anywhere (WORA)**. This means that compiled Java code can run on all platforms that support Java without the need to recompile.



Why to Learn Java?

Java is a MUST for students and working professionals to become a great Software Engineer.

The reasons that makes Java as the first choice of any programmer:

- ❑ Java is Open Source which means its available free of cost.
- ❑ Java is simple and so easy to learn
- ❑ Java is much in demand and ensures high salary
- ❑ Java has a large vibrant community
- ❑ Java has powerful development tools
- ❑ Java is platform independent

Java is guaranteed to be Write Once, Run Anywhere

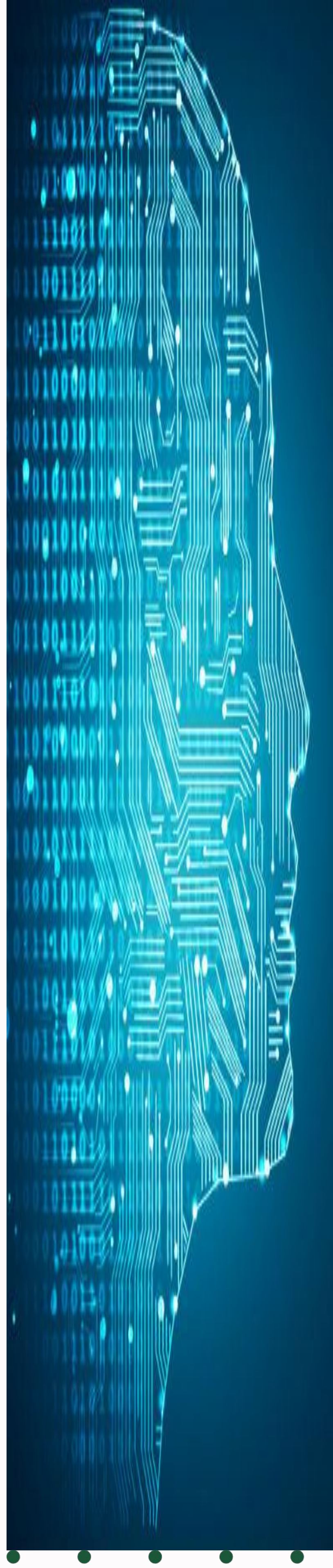
Java is –

- **Object Oriented** – In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Simple** – Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it will be easy to master.
- **Secure** – Java's secure feature it enables to develop of virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **High Performance** – With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** – Java is designed for the distributed environment of the internet.
- **Dynamic** – Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Java - Basic Syntax

- -**Object** – Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.

- **Class** – A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.



Java - Basic Syntax

- **-Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

- **Instance Variables** – Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

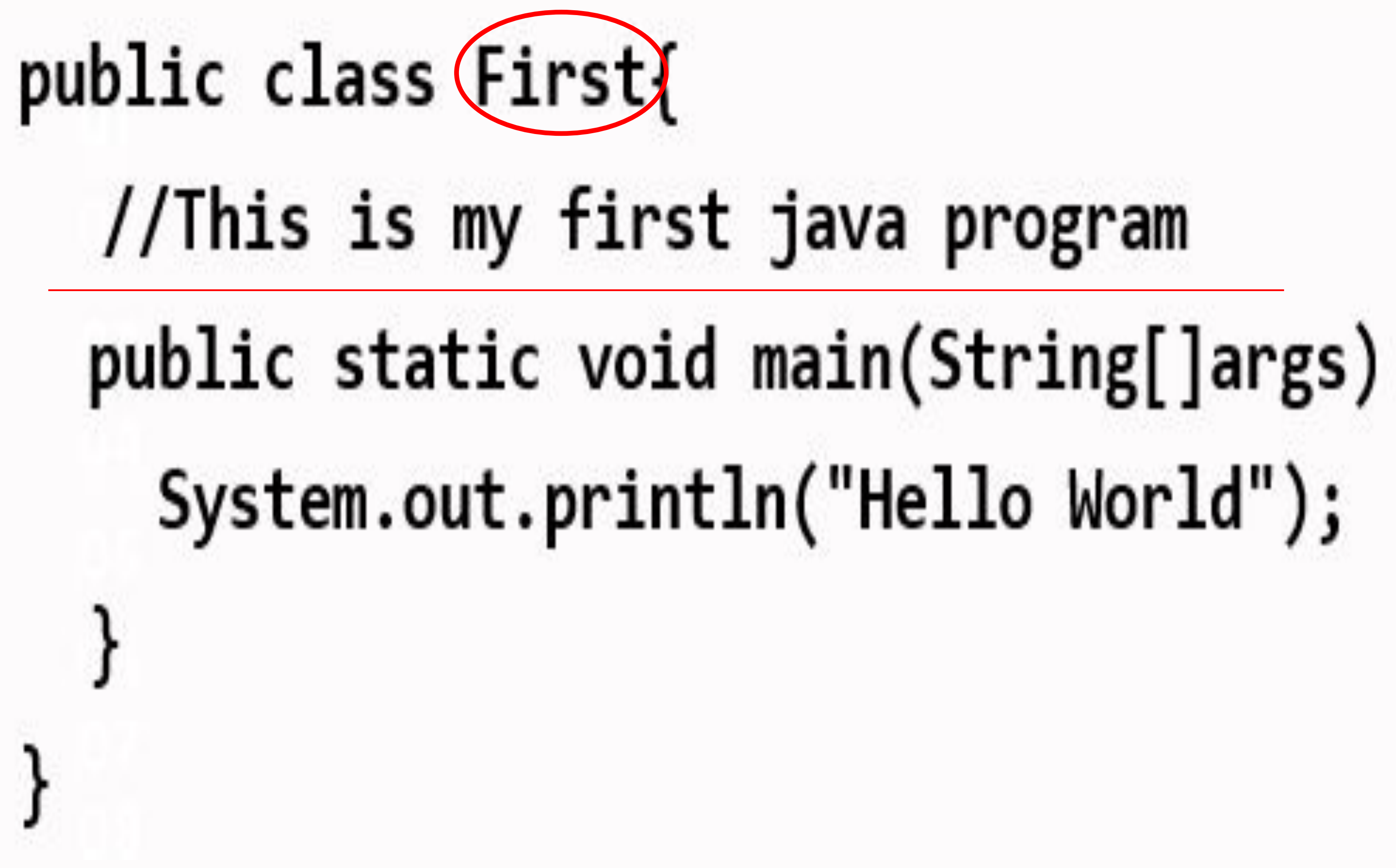
First Java Program

```java
public class First{
 //This is my first java program
  public static void main(String[]args){
    System.out.println("Hello World");
  }
}
```

How Java “Hello, World!” Program Works?

1. Every line of code that runs in Java must be inside a class. In our example, we named the class **First**. A class should always start with an **uppercase** first letter.

2. In Java, any line starting with // is a comment. Comments are intended for users reading the code to understand the intent and functionality of the program. It is completely ignored by the Java compiler (an application that translates Java program to Java bytecode that computer can execute).



```
public class First{
  //This is my first java program
  public static void main(String[]args)
    System.out.println("Hello World");
  }
}
```

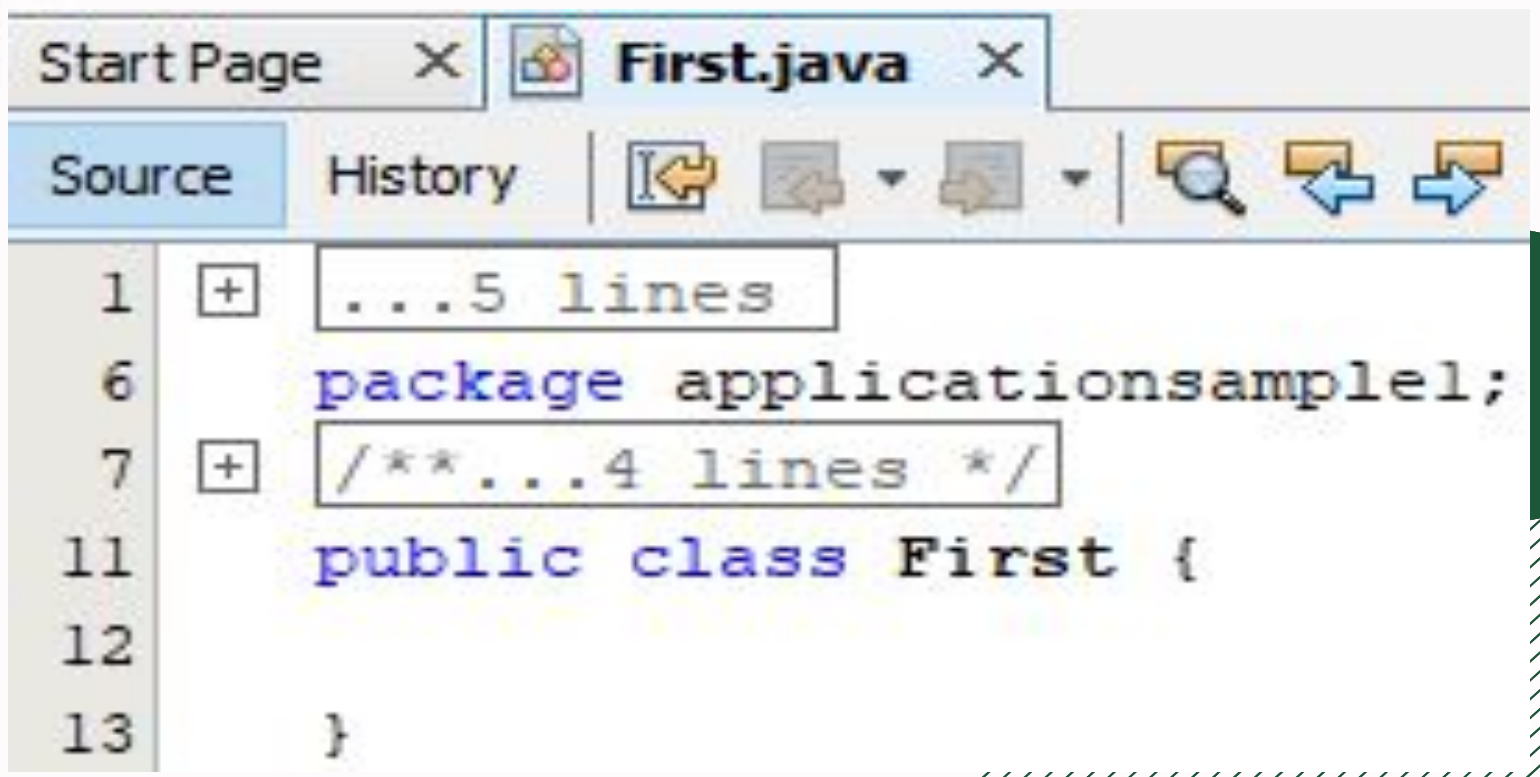
How Java “Hello, World!” Program Works?

3. The name of the java file **must match** the class name. When saving the file, save it using the class name and add ".java" to the end of the filename.

4. public static void main(String[] args) {

Java program processing starts from the main() method which is a mandatory part of every Java program.

```
public class First{
  //This is my first java program
  public static void main(String[]args)
    System.out.println("Hello World");
  }
}
```



```
+ ...5 lines
  package applicationsample1;
+ /**...4 lines */
  public class First {

  }
```

public static void main(String []args) {

- **public**- It is an access specifier that means the main() method is accessible globally available. This is necessary because this method is being called by the Java Runtime Environment (JRE) which is not located in your current class.

- **static**- The main() method in Java must be static because they can then be invoked by the runtime engine without having to instantiate any objects then the code in the body of main() method will do the rest.

- **void**- The "***void***" is a return type i.e. it does not return any value. When the main() method terminates, the java program terminates too. If you try to return something from the main method, it will give compilation error as an unexpected return value.

public static void main(String []args) {

- ✔ **main**()- It’s the name of function name. This name is fixed and as it's called by the JVM as entry point for an application. It's not a keyword.

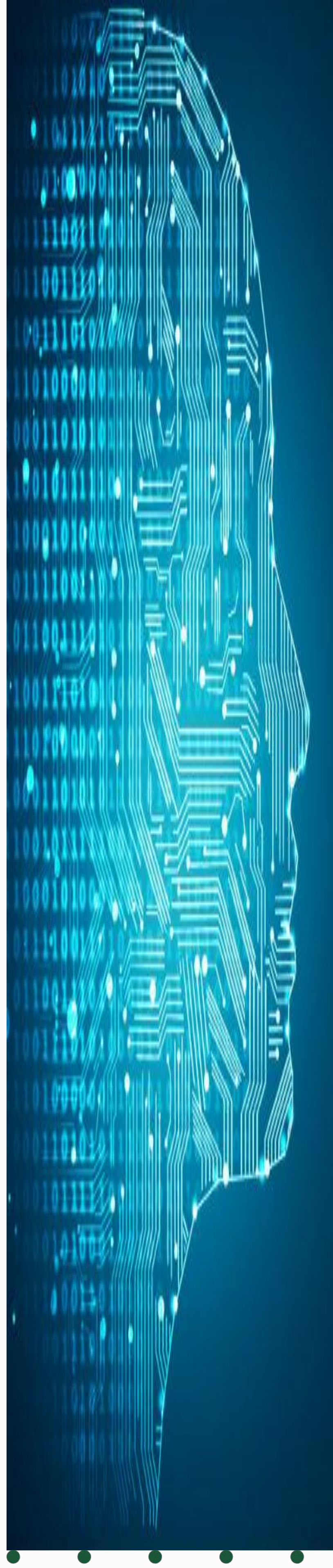
- ✔ **String [] args-** These are the arguments of type String that your Java application accepts when you run it. Java main() method accepts only string type of argument and stores it in a string array. It is a collection of Strings, separated by a space, which can be typed into the program on the terminal.

System.out.println("Hello World");

method to print a line of text to the screen

=> **System.**out.println("Only this text will print"); - we will use if you only want to print a text.

=> **System.**out.println("Only this text will print:" + variable declared); - we will use if you want to print output with the use of data stored in the assigned variables.



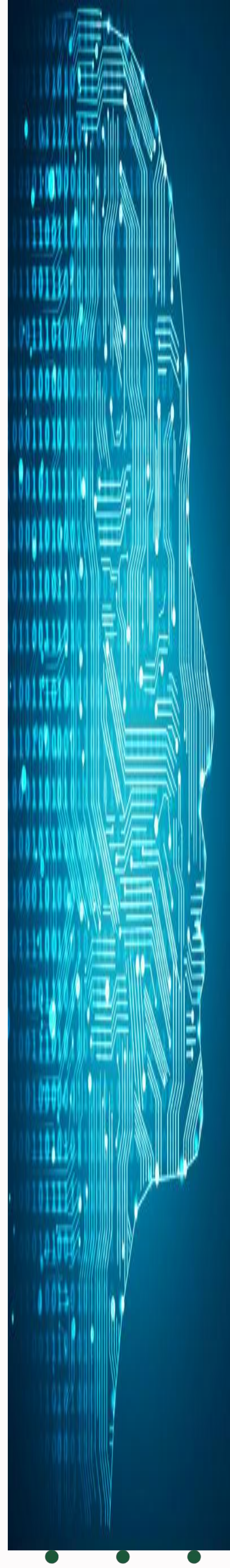
Basic Syntax

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity** – Java is case sensitive, which means identifiers **Hello** and **hello** would have different meanings in Java.

- **Class Names** – For all class names the first letter should be in **Upper Case**. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

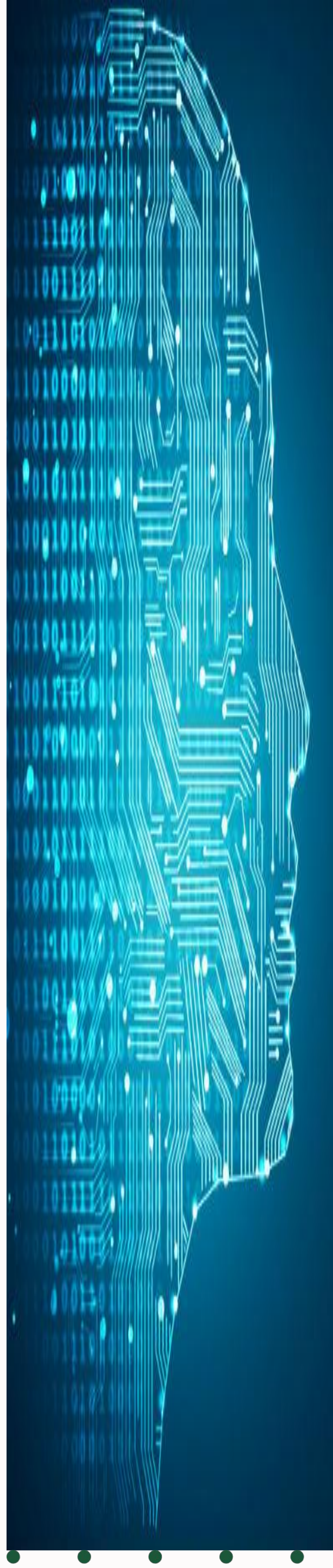
  Example: *class MyFirstJavaClass{*



- **Method Names** – All method names should start with a **Lowercase** letter.
  Example: *public void myMethodName()*

- **Program File Name** – The name of the program file should exactly match the class name.

- **public static void main(String args[])** – Java program processing starts from the main() method which is a mandatory part of every Java program.



JAVA COMMENTS

Comments can be used to explain Java code and to make it more readable. It can also be used to prevent execution when testing alternative code.

❑ Single-line Comments

Single-line comments start with two forward slashes (//).
Any text between // and the end of the line is ignored by Java (will not be executed).

❑ Java Multi-line Comments

Multi-line comments start with /* and end with */.
Any text between /* and */ will be ignored by Java.

JAVA IDENTIFIERS

All Java components require names. Names used for classes, variables, and methods are called **identifiers**.

In Java, there are several points to remember about identifiers. They are as follows –

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- Most importantly, identifiers are case-sensitive.

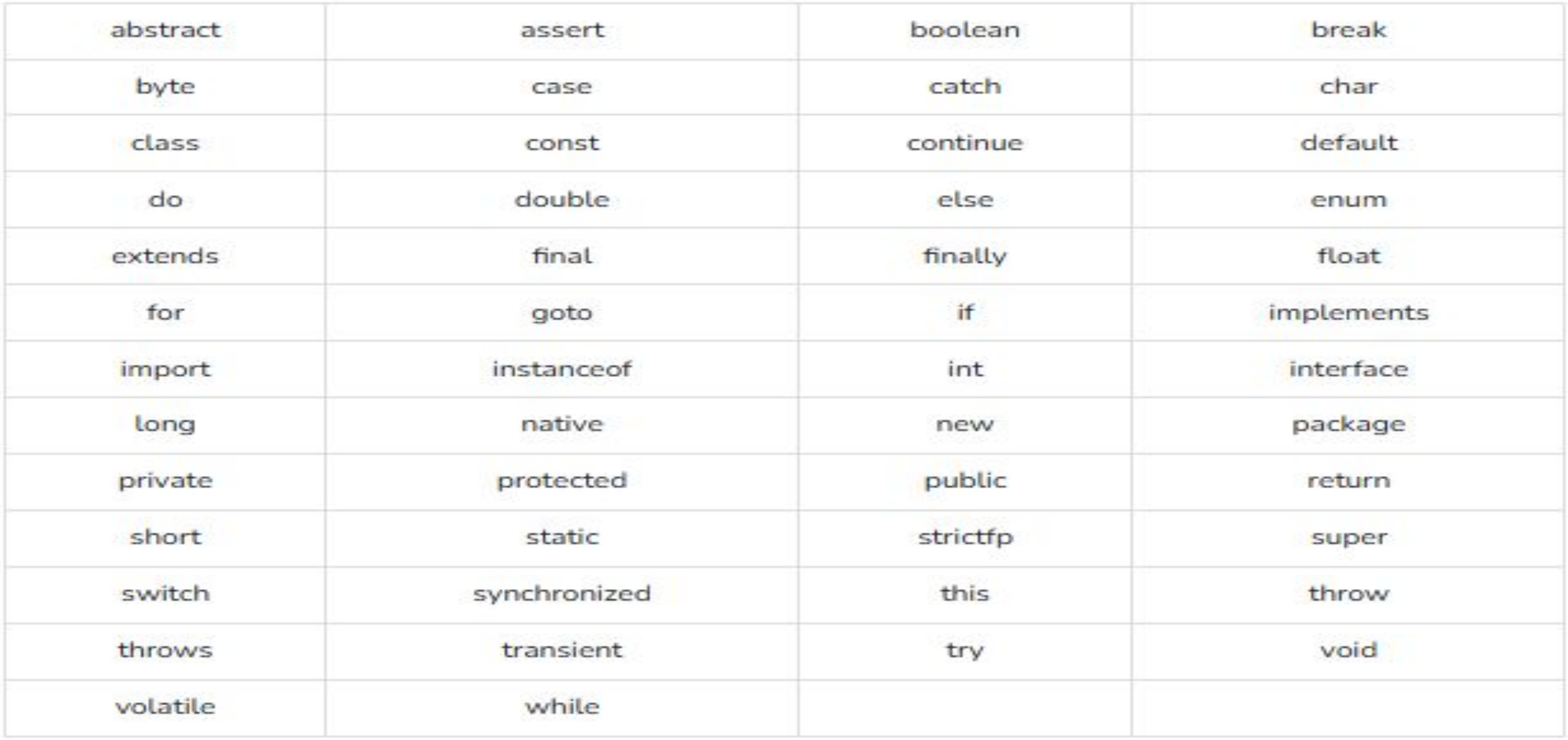
JAVA MODIFIERS

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers –

- **Access Modifiers** – default, public, protected, private
- **Non-access Modifiers** – final, abstract, strictfp

JAVA KEYWORDS

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.



abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

JAVA BASIC DATA TYPES

The purpose of data types in Java programming is to define the kind of data that can be stored and manipulated within a program. It determine the type of operations that can be performed on the data, the amount of memory allocated for the data, and how the data is stored in memory.

Data Type Categories:

- **Primitive Data Types** – most basic data types available within the Java Language. It has eight predefined data types.

- **Reference/Object Data Types-** used to refer to objects. Used and defined by classes.

JAVA BASIC DATA TYPES

Primitive Data Types examples:

- ❖ **byte**: 8-bit signed integer. Stores whole numbers.
  Example: byte a = 100;
- ❖ **short**: 16-bit signed integer. Stores whole numbers.
  Example: short b = 10000;
- ❖ **int**: 32-bit signed integer. Stores whole numbers.
  Example: int c = 100000;
- ❖ **long**: 64-bit signed integer. Stores whole numbers.
  Example: long d = 100000L;

JAVA BASIC DATA TYPES

Primitive Data Types examples:

- ❖ **float**: Stores fractional numbers. Sufficient for storing 6-7 decimal digits.
  Example: float e = 234.5;
- ❖ **double**: Stores fractional numbers. Sufficient for storing 15 decimal digits.
  Example: double f = 123.4;
- ❖ **boolean**: Represents one bit of information; either true or false. Example: boolean g = true;
- ❖ **char**: A single 16-bit Unicode character.
  Example: char h = ‘A’;
- ❖ **String**: Stores a series of characters

JAVA OPERATORS

Java Operators are special symbols used to perform operations on variables and values. They are categorize as follows:

- **Arithmetic Operators-** used to perform basic arithmetic operations. Assume integer variable A holds 10 and variable B holds 20, then=

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
*** (Multiplication)**	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

JAVA OPERATORS

- **Relational Operators-** used to compare two values. Assume integer variable A holds 10 and variable B holds 20, then=

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

JAVA OPERATORS

- **Logical Operators-** used to perform logical operations. Assume Boolean variables A holds true and variable B holds false, then=

Operator	**Description**	**Example**
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

JAVA OPERATORS

- **Assignment Operators-** used to assign values to variables.

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
***=**	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A

VARIABLE TYPES

A class can contain any of the following variable types.

- **Local variables** – Variables defined inside methods, constructors or blocks. The variable will be declared and initialized within the method and the variable will be destroyed when the method has been completed.
- **Instance variables** – Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables** – Class variables are variables declared within a class, outside any method, with the static keyword.

```
public class Dog {
   String breed;
   int age;
        public   String
color;

   void barking() {
     }

   void hungry() {
     }
   void sleeping() {
   }
}
```

SAMPLE PROGRAM

Write a Java program that will print the sum of the following numbers: 12, 76,34,52,89.

```
public class Sum {
    public static void main(String[]args){
int a=12, b=76, c=34, d=52, e=89;
int sum;
sum=a+b+c+d+e;
System.out.println("The sum is:"+sum);
    }
}
```

THANK YOU