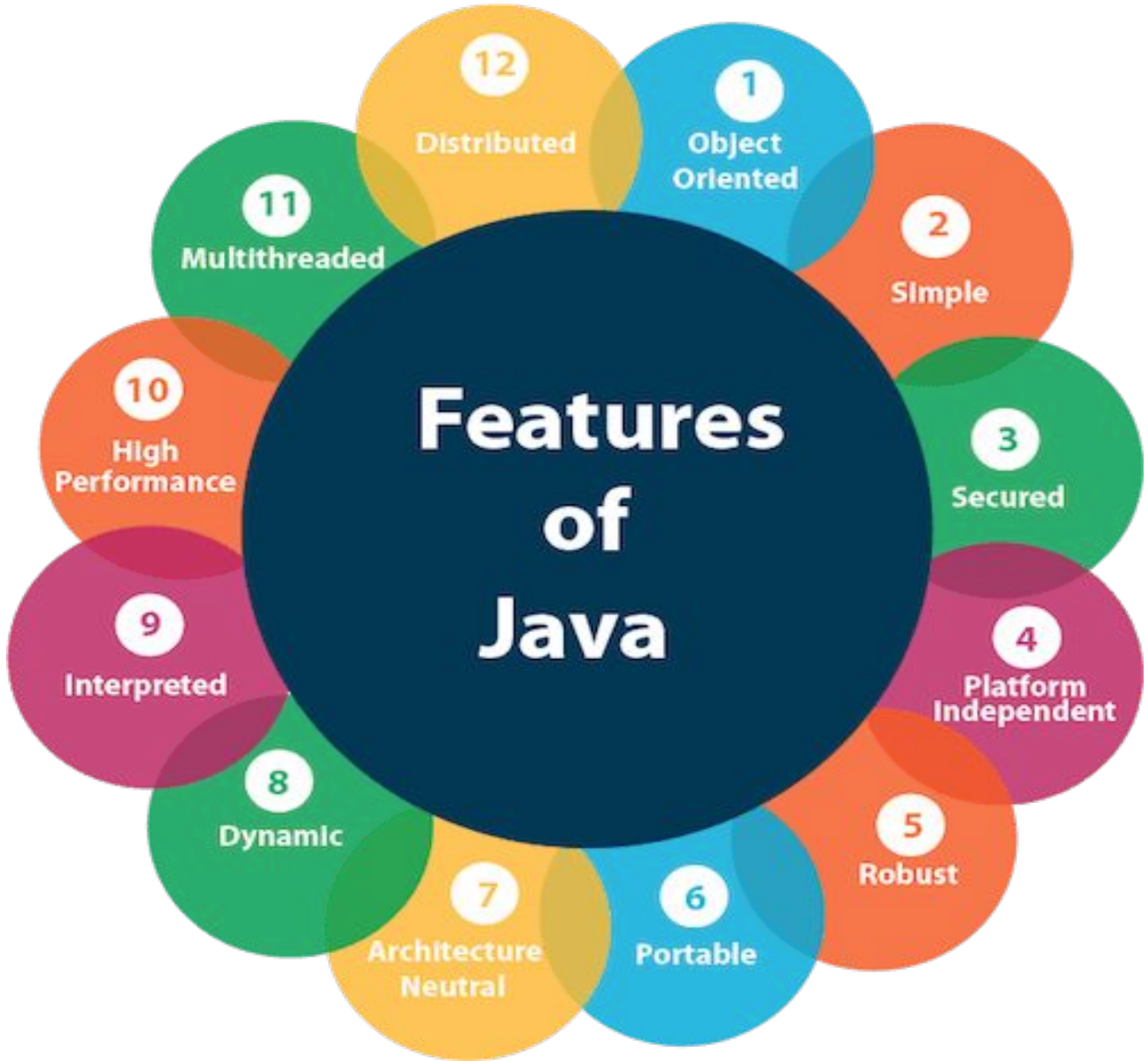
Java Fundamentals

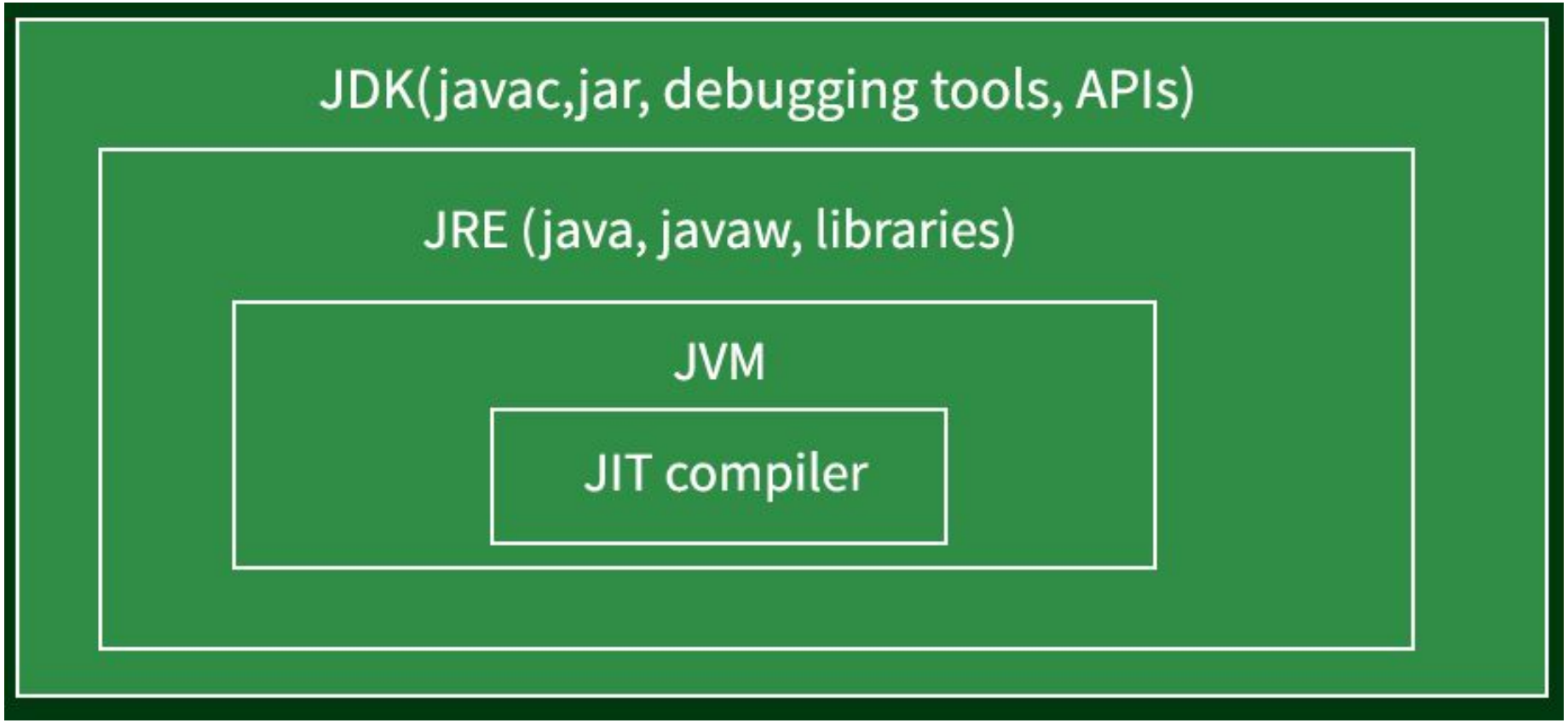
Agenda:

- Introduction to Java Architecture (JVM, JRE & JDK)
- Identifiers
- Reserved Words
- Variables
- Data Types
- Comments
- Literals
- Operators

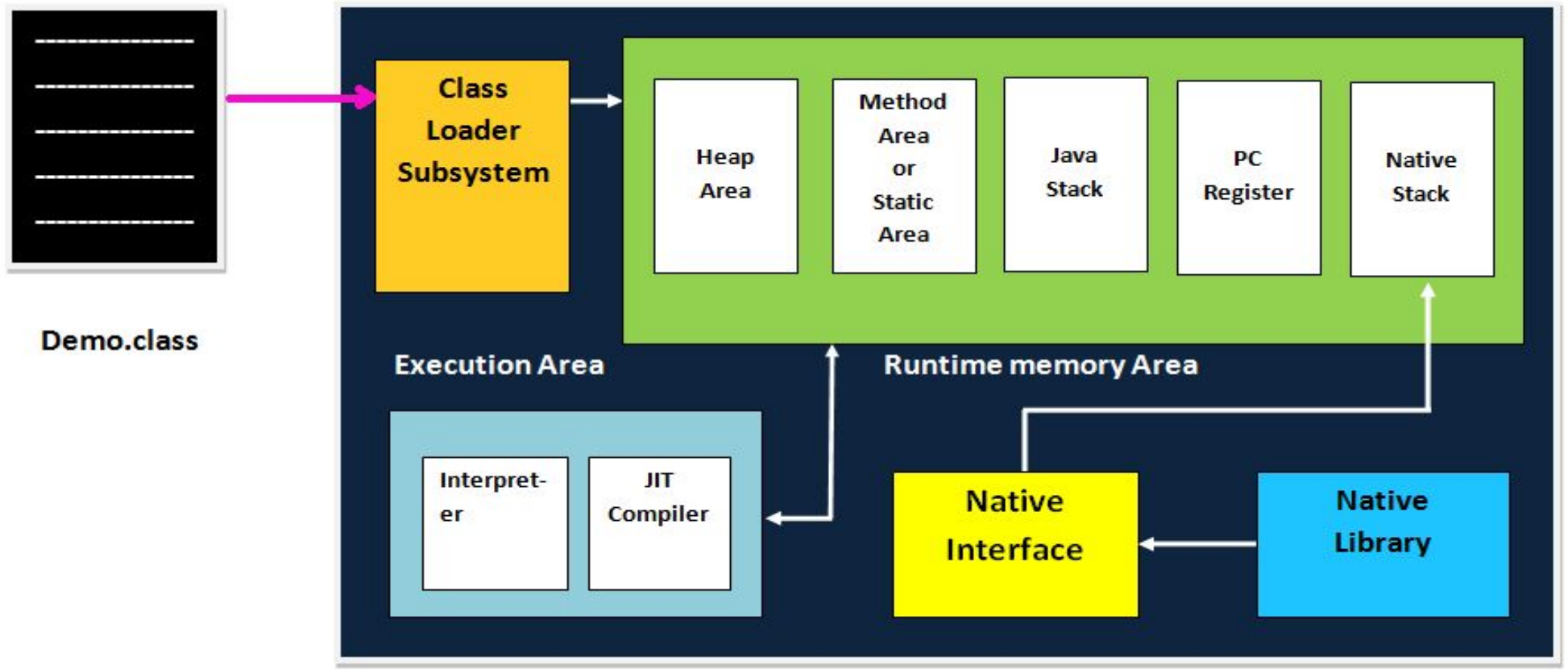

Features of Java
1 Object Oriented
2 Simple
3 Secured
4 Platform Independent
5 Robust
6 Portable
7 Architecture Neutral
8 Dynamic
9 Interpreted
10 High Performance
11 Multithreaded
12 Distributed

Introduction to Java Architecture (JVM, JRE & JDK)

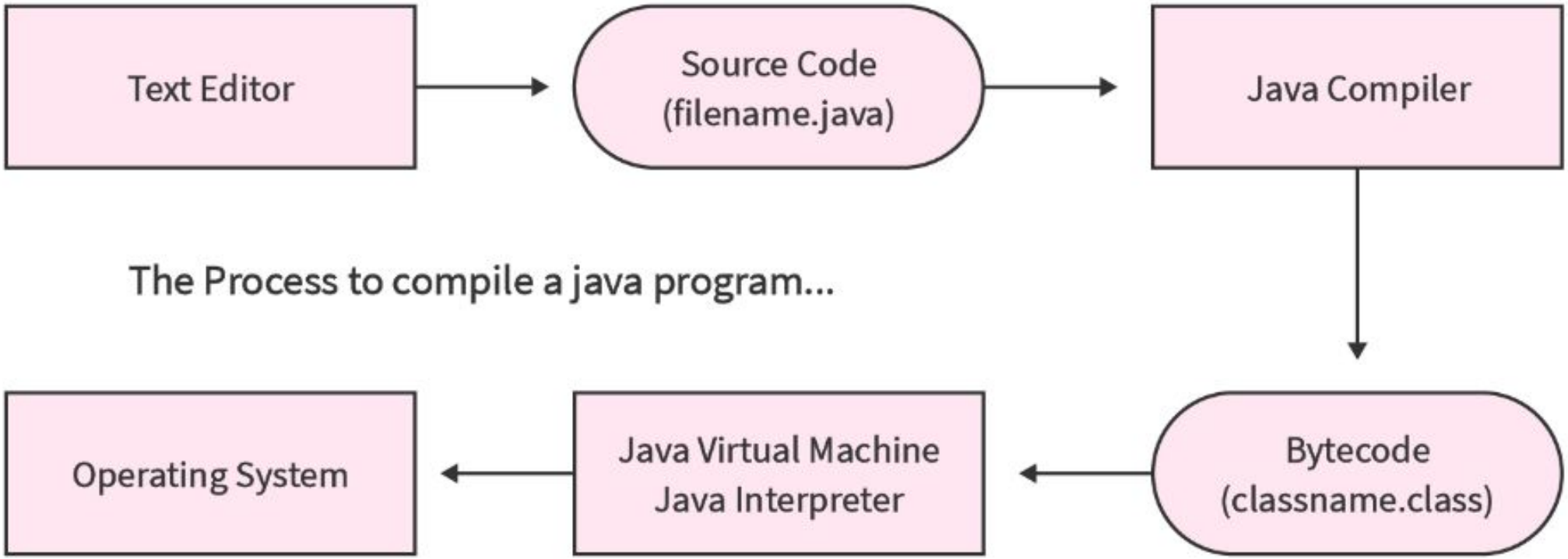
Difference between JIT , JVM , JRE , JDK



JVM Architecture



Execution Engine



Hello, World! Program in Java

```
public class HelloWorld {

    public static void main(String[] args) {

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

    }
}
```

public static void main(String[] args) {

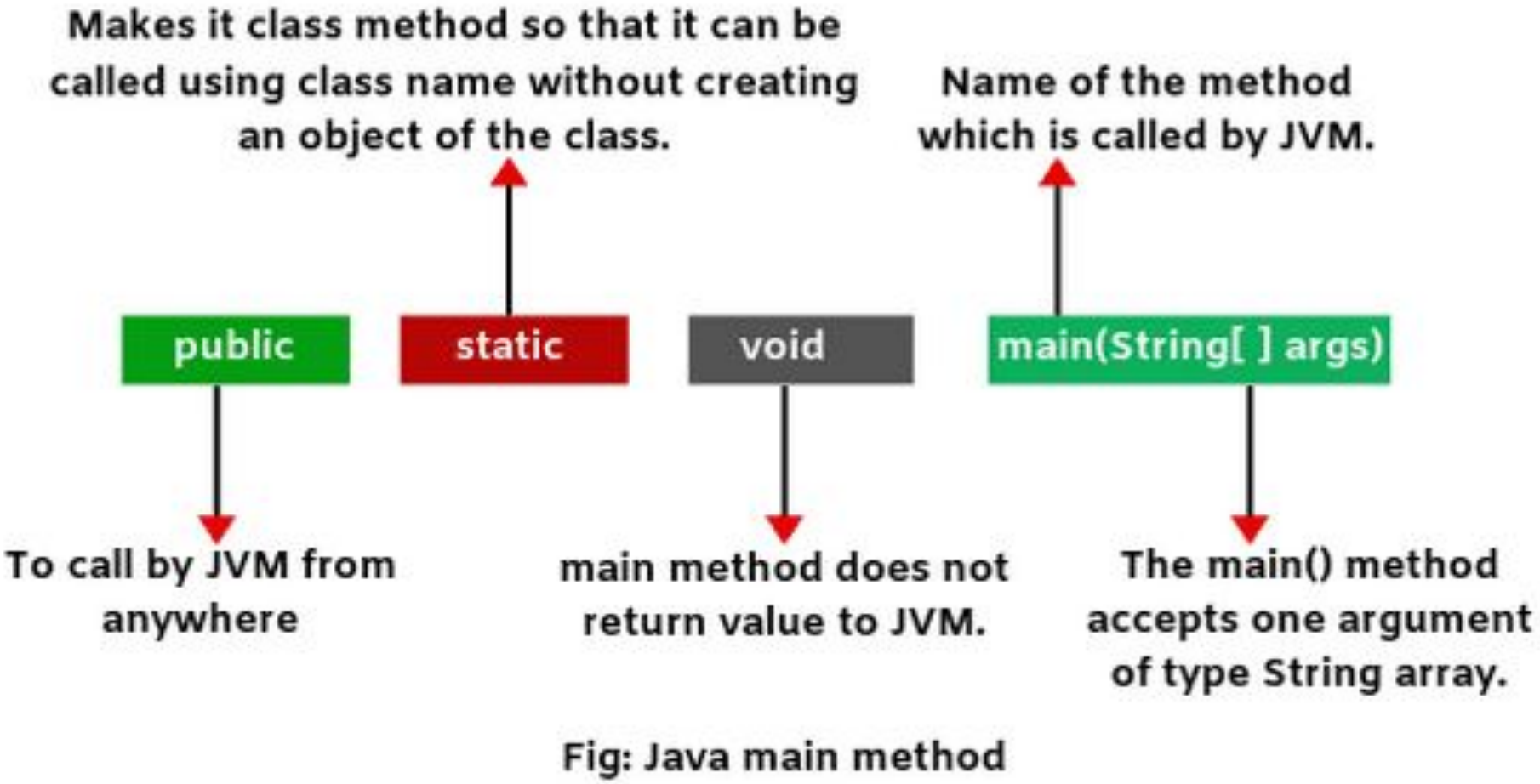
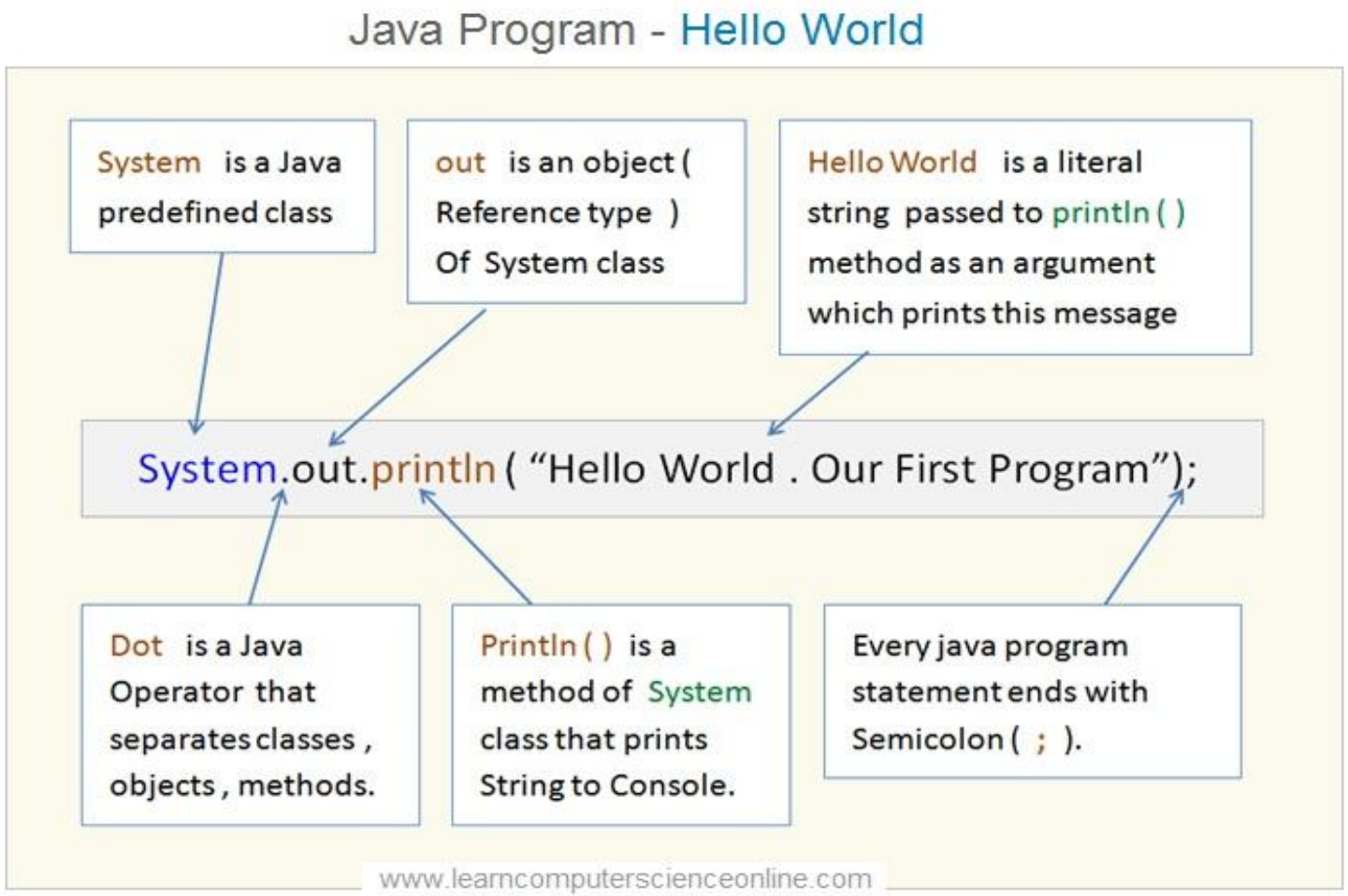


Fig: Java main method

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



Identifiers in Java

What is Identifiers ?

- In Java, identifiers are names used to identify
  - => variables,
  - => methods,
  - => objects,
  - => classes,
  - => labels,
  - => packages,
  - => interfaces,..etc

Rules for naming identifiers in Java

- Identifiers can include letters (A-Z, a-z), digits (0-9), underscore (_), and dollar sign ($).
- Identifiers cannot start with a digit.
- Identifiers are case-sensitive.
- Identifiers cannot be Java reserved keywords.
- Identifiers can be of any length.
- Identifiers can use predefined class names (though it's not recommended for clarity).

In the following Java code snippet, identify all the identifiers used:

```
public class Example {
    public static void main(String[] args) {
        int number = 10;
        String message = "Hello";
        System.out.println(message + " World " + number);
    }
}
```

Answer:

```
public class Example {
    public static void main(String[] args) {
        int number = 10;
        String message = "Hello";
        System.out.println(message + " World " + number);
    }
}
```

Task:1

```
public class BankAccount {
    private double balance;

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        balance -= amount;
    }

    public double getBalance() {
        return balance;
    }
}
```

Answer:

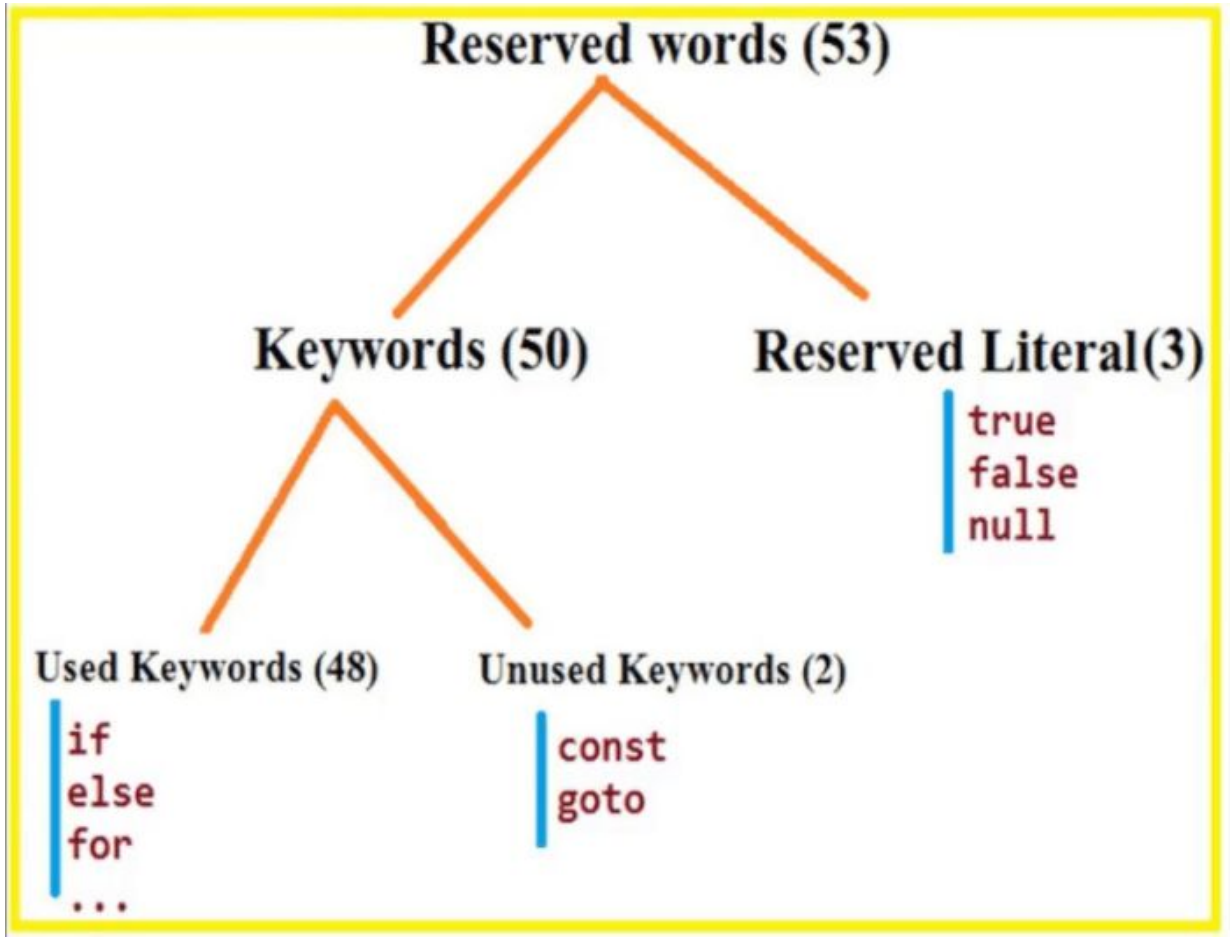
1. BankAccount (class name)
2. balance (field name)
3. deposit (method name)
4. withdraw (method name)
5. getBalance (method name)
6. amount (parameter name)

```
public class BankAccount {
    private double balance;

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        balance -= amount;
    }

    public double getBalance() {
        return balance;
    }
}
```

Reserved words in Java

- reserved words (or **keywords**) are **predefined** by the language and have **special meanings.**

- These words **cannot be used as identifiers** (such as variable names, method names, or class names).



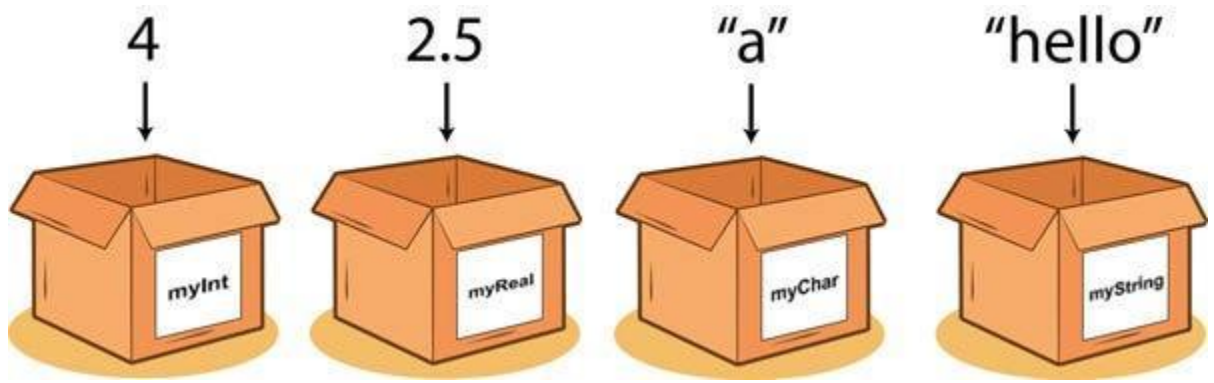
List of Keywords in Java

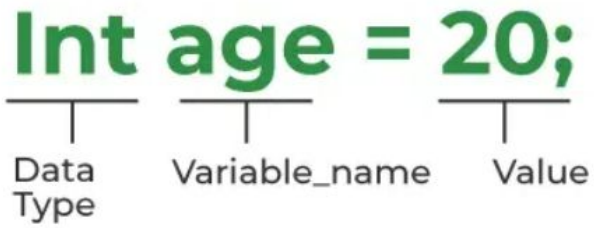
Primitive Types	Modifiers	Declarations	Control Flow	Miscellaneous
1. Boolean	1. public	1. class	1. if	1. this
2. byte	2. protected	2. interface	2. else	2. new
3. char	3. private	3. enum	3. try	3. super
4. short	4. abstract	4. extends	4. catch	4. import
5. int	5. static	5. implements	5. finally	5. instanceof
6. long	6. final	6. package	6. do	6. null
7. float	7. transient	7.throws	7. while	7. true
8. double	8. volatile		8. for	8. false
9. void	9. synchronized		9. continue	9. strictfp
	10. native		10. break	10. goto
			11. switch	11.assert
			12. case	
			13. default	12.const
			14. throw	
			15. return	

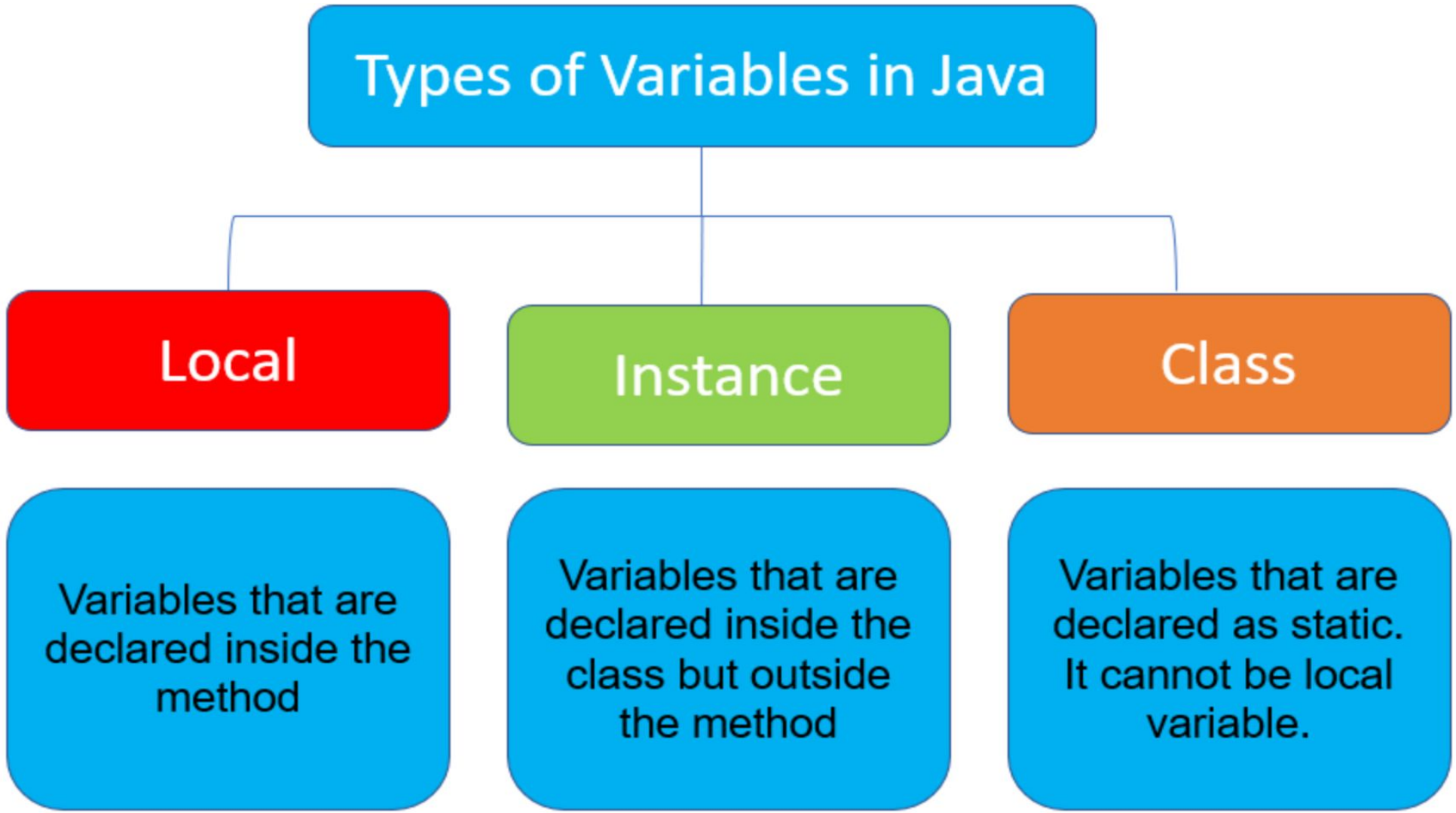
Variables In Java

Variables In Java

- A variable is a container which **holds the value** while the Java program is executed.
- A variable is assigned with a data type.
- Variable is a name of **memory location.**

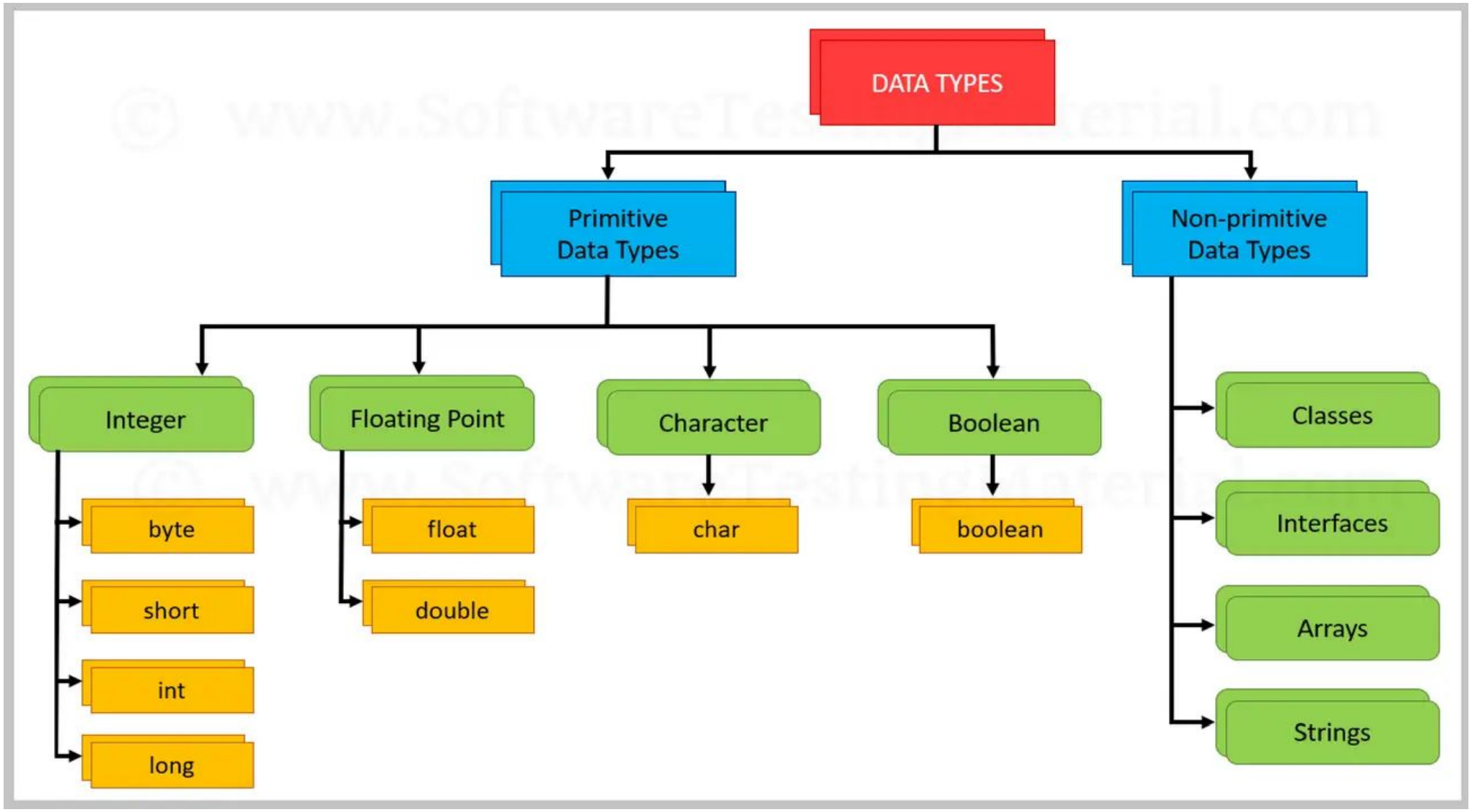





Types of Variables in Java
Local
Instance
Class
Variables that are declared inside the method
Variables that are declared inside the class but outside the method
Variables that are declared as static. It cannot be local variable.

Data Types In Java

Data Types In Java



byte

Size: 1 byte

Range: -128 to 127

```
public class ByteExample {
    public static void main(String[] args) {
        byte b = 100;
        System.out.println(b);
    }
}
```

```
100
```

short

Size: 2 bytes

Range: -32,768 to 32,767

```
public class ShortExample {
    public static void main(String[] args) {
        short s = 15000;
        System.out.println(s);
    }
}
```

```
15000
```

int

Size: 4 bytes

Range: -2^31 to 2^31 - 1

```
public class IntExample {
    public static void main(String[] args) {
        int i = 100000;
        System.out.println(i);
    }
}
```

```
100000100000
```

long

Size: 8 bytes

Range: -2^63 to 2^63 - 1

```
public class LongExample {
    public static void main(String[] args) {
        long l = 10000000000L;
        System.out.println(l);
    }
}
```

```
10000000000
```

float

Size: 4 bytes

Range: ±3.40282347E+38F(7 decimal digits)

```
public class FloatExample {
    public static void main(String[] args) {
        float f = 3.14f;
        System.out.println(f);
    }
}
```

```
3.14
```

double

Size: 8 bytes

Range: ±1.79769313486231570E+308 (15 decimal digits)

```
public class DoubleExample {
    public static void main(String[] args) {
        double d = 3.141592653589793;
        System.out.println(d);
    }
}
```

```
3.141592653589793
```

char

Size: 2 bytes

Range: 0 to 65,535 (Unicode characters)

```
public class CharExample {
    public static void main(String[] args) {
        char c = 'A';
        System.out.println(c);
    }
}
```

```
A
```

boolean

Size: Not precisely defined; often 1 byte or 1 bit per value

Range: true or false

```
public class BooleanExample {
    public static void main(String[] args) {
        boolean b = true;
        System.out.println(b);
    }
}
```

```
true
```

Commends in java

- comments are used to add explanatory notes to the code, making it easier to understand and maintain.

Types:

- Single line
- Multi line

Single-Line Comments

- **Syntax:** Start with **//** and continue to the end of the line.
- **Usage:** Used for short explanations or notes.

```
public class Demo {
    public static void main(String[] args) {
        int x = 10; // This is a single-line comment
        System.out.println(x);
    }
}
```

Multi-Line Comments

- **Syntax:** Start with **/*** and end with ***/**. Can span multiple lines.
- **Usage:** Used for longer explanations or to comment out multiple lines of code.

```
public class Demo {
    public static void main(String[] args) {
        /*
         * This is a multi-line comment.
         * It can span multiple lines.
         */
        int y = 20;
        System.out.println(y);
    }
}
```

Literals In Java

Operators In Java


Java Operators
Arithmetic Operators
Logical Operators
Comparison Operators
Unary Operators
Bitwise Operators
Assignment Operators
Ternary Operators
+
-
*
/
%
&&
(AND)
||
(OR)
!
(NOT)
==
!=
>
<
>=
<=
++
--
+
-
!
&
|
^
~
<<
>>
>>>
=
+=
-=
*=
/=
%=
&=
|=
^=
>>=
<<=
?:

Arithmetic operators:

```
public class ArithmeticDemo {
    public static void main(String[] args) {
        int x = 8, y = 3;
        System.out.println("Sum: " + (x +y));
        System.out.println("Difference: " + (x - y));
        System.out.println("Product: " + (x * y));
        System.out.println("Quotient: " + (x / y));
        System.out.println("Remainder: " + (x % y));
    }
}
```

```
Sum: 11
Difference: 5
Product: 24
Quotient: 2
Remainder: 2
```

Relational operators:

```
public class RelationalDemo {
    public static void main(String[] args) {
        int x = 10, y = 5;

        System.out.println(x > y);
        System.out.println(x < y);
        System.out.println(x >= y);
        System.out.println(x <= y);
        System.out.println(x == y);
        System.out.println(x != y);
    }
}
```

```
true
false
true
false
false
true
```

Logical operators:

```
public class LogicalDemo {
    public static void main(String[] args) {
        boolean a = true, b = false;

        System.out.println(a && b);
        System.out.println(a || b);
        System.out.println(!a);
        System.out.println(!b);
    }
}
```

```
false
true
false
true
```

Unary operators:

```
public class UnaryDemo {
    public static void main(String[] args) {
        int x = 5;
        System.out.println(++x);
        System.out.println(x--);
        System.out.println(!true);
    }
}
```

```
6
6
false
```

Bitwise operators:

```
public class BitwiseDemo {
    public static void main(String[] args) {
        int x = 5, y = 3;
        System.out.println(x & y);
        System.out.println(x | y);
        System.out.println(x ^ y);
        System.out.println(~x);
    }
}
```

```
1
7
6
-6
```

Ternary operators:

```
public class TernaryDemo {
    public static void main(String[] args) {
        int a = 10, b = 20;
        System.out.println(a > b ? "a is greater" : "b is greater");

        System.out.println(a == b ? "a equals b" : "a does not equal b");
    }
}
```

b is greater
a does not equal b

Shift operators:

```
public class ShiftDemo {
    public static void main(String[] args) {
        int x = 8;
        System.out.println(x << 2);
        System.out.println(x >> 2);
        System.out.println(x >>> 2);
    }
}
```

```
32
2
2
```