Introduction to Java

V2V
CLASSES

RAJAN SHUKLA

Professor

V2V Classes

JAVA

1

Java programming language was originally developed by Sun Microsystems

2

It was initiated by James Gosling and released in 1995

3

It is core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

4

Sun Microsystems has renamed the new J2 versions as Java SE, Java EE & Java ME resp.

FEATURES OF JAVA

1. Object Oriented
2. Platform Independent
3. Secure
4. Robust
5. Simple
6. Portable

FEATURES OF JAVA

- 7 Multithreaded
- 8 Interpreted
- 9 Distributed
- 10 Dynamic
- 11 High Performance
- 12 Architectural neutral

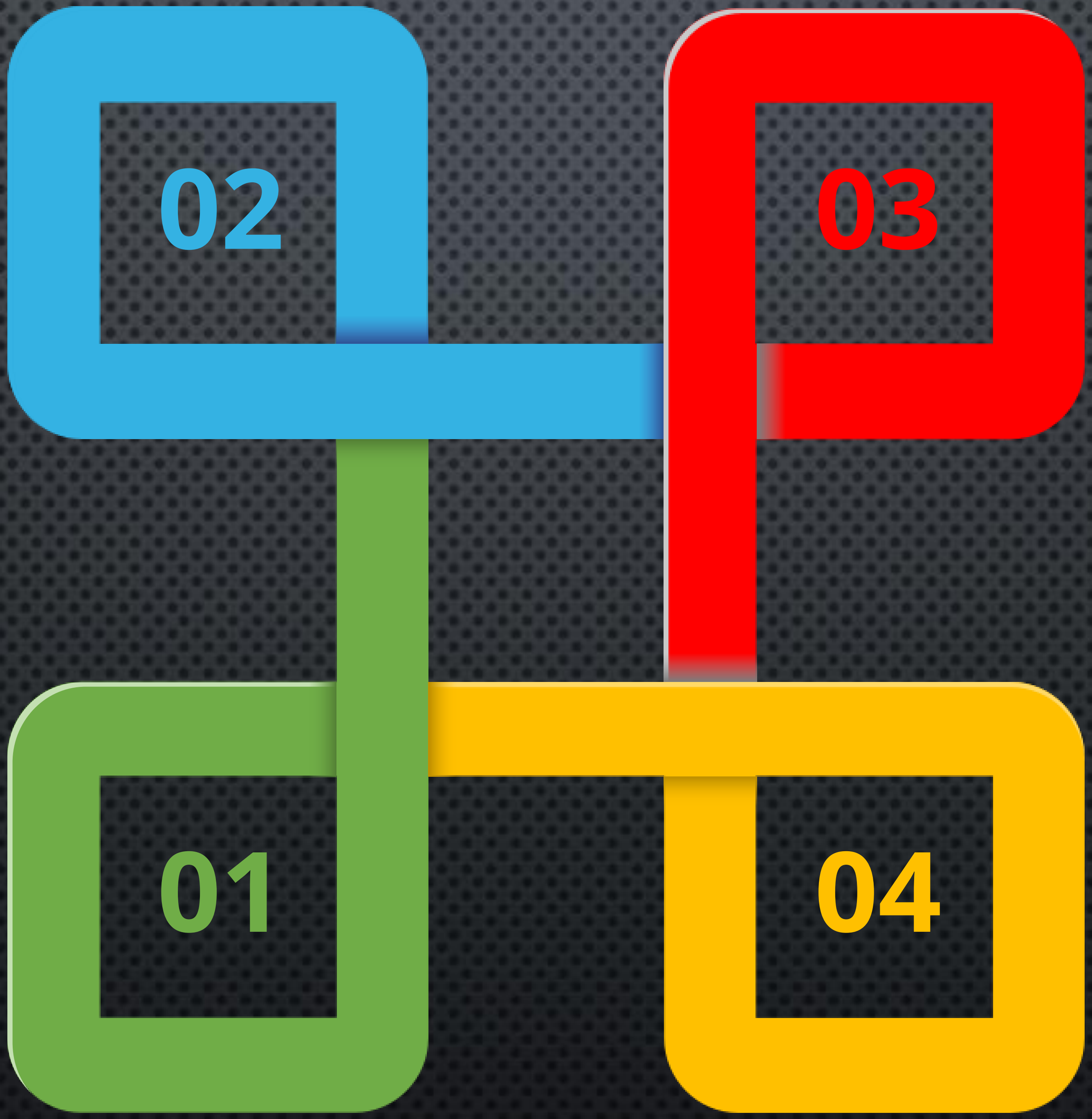
HISTORY OF JAVA

Sun released the first public implementation as Java 1.0 in 1995.

On 13 November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

A JAVA PROGRAM INCLUDES



CLASS

METHODS

OBJECT

INSTANCE VARIABLES

CLASSES IN JAVA

A class can be defined as a template/ blueprint.

Class is a collection of variable & function

CLASSES IN JAVA

Class is a logical Entity to create objects that share common properties and methods.

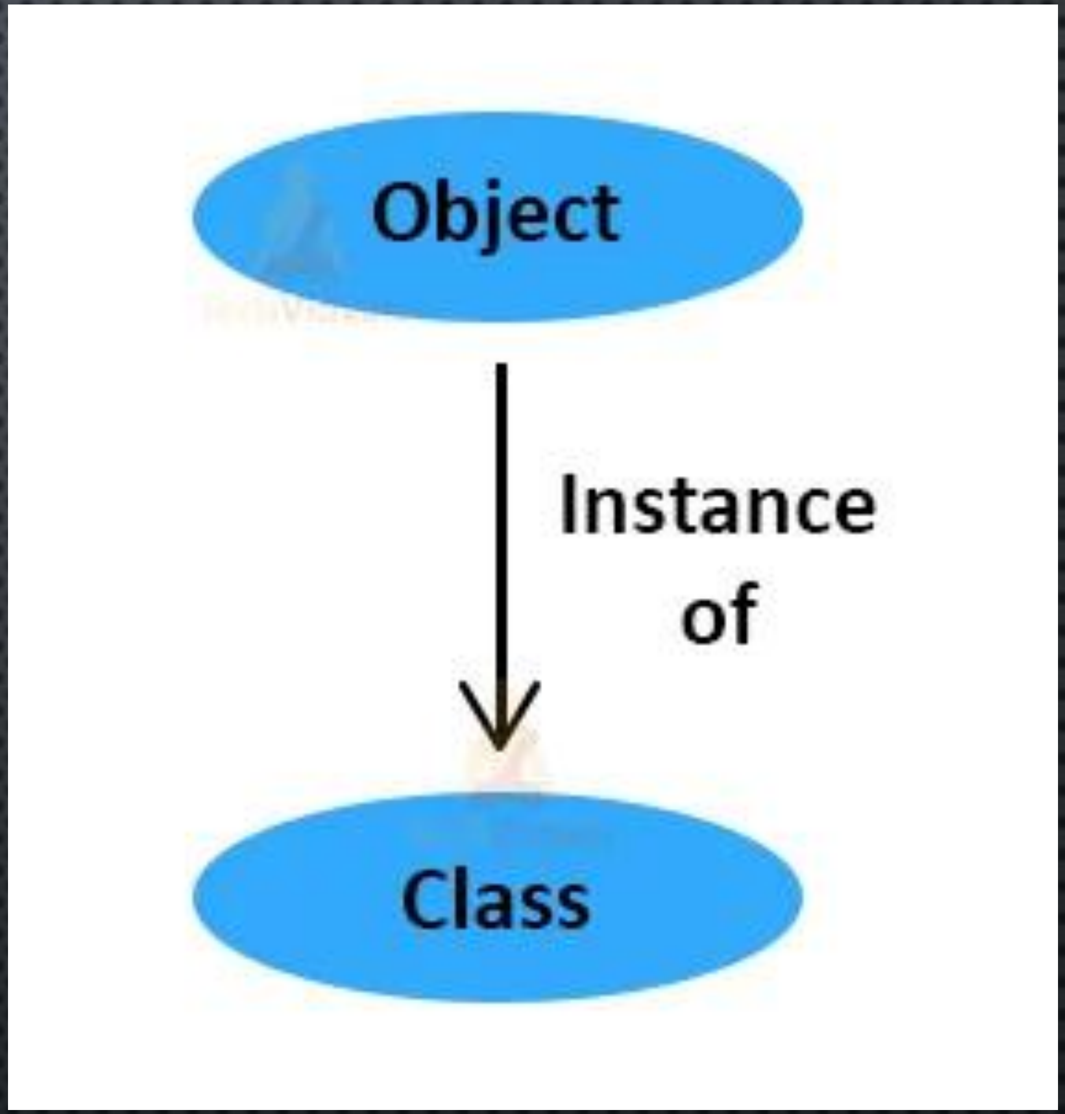
In Java, an object is created from a class.

OBJECT IN JAVA

An object is an instance of a class. It is real-world Entity. We can create N number of Objects from one class

Objects have all variable and function defined in class.Object is created from class

For ex, a dog has variable color, name, breed as well as behaviors like barking, eating.



METHODS IN JAVA

A method is set of statement that is used to do a task and it is reusable

A class can contain many methods. Method is written once and can be called n number of times

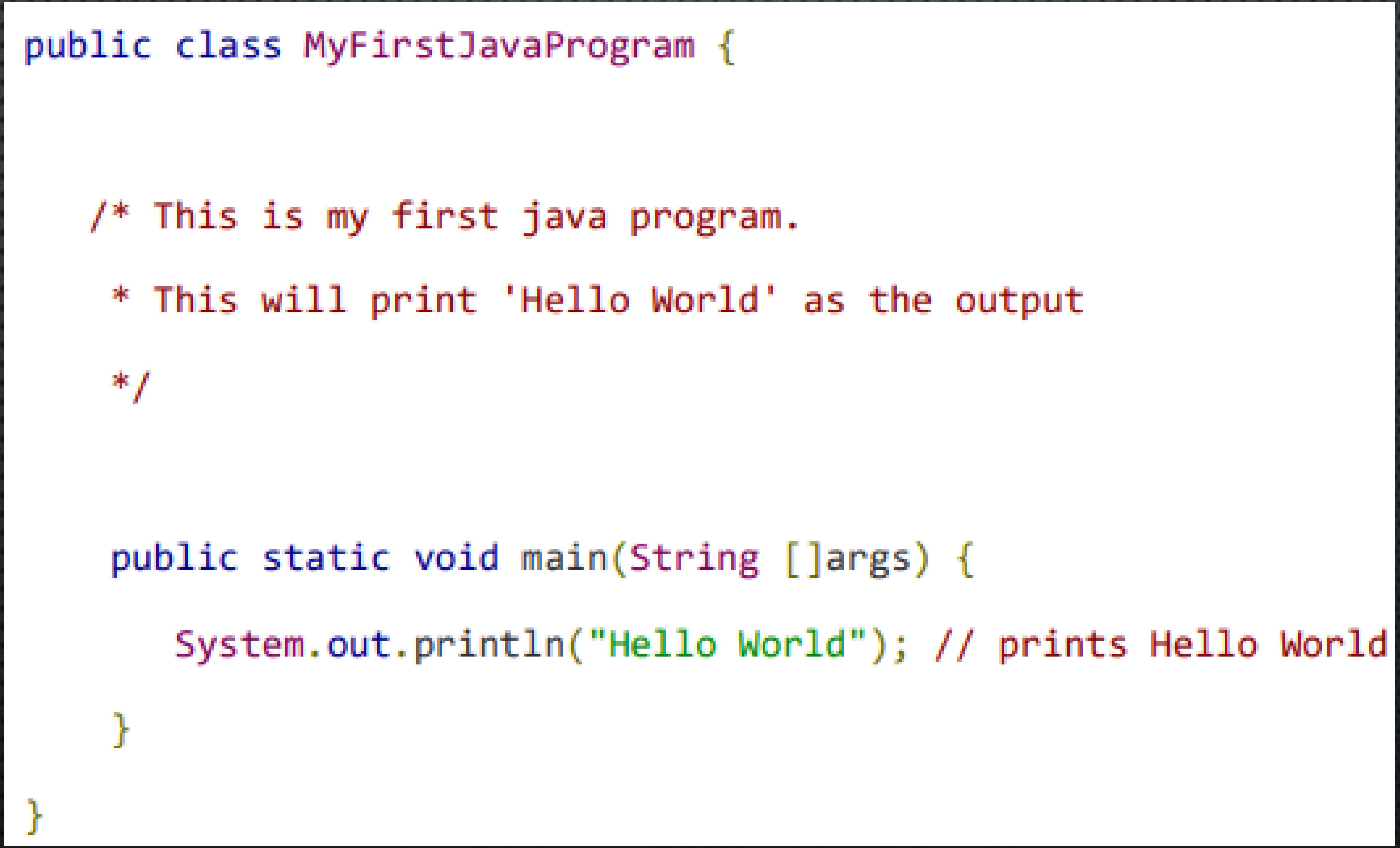
Method is identified by (). Method needs to be called as it cannot be invoked automatically

01

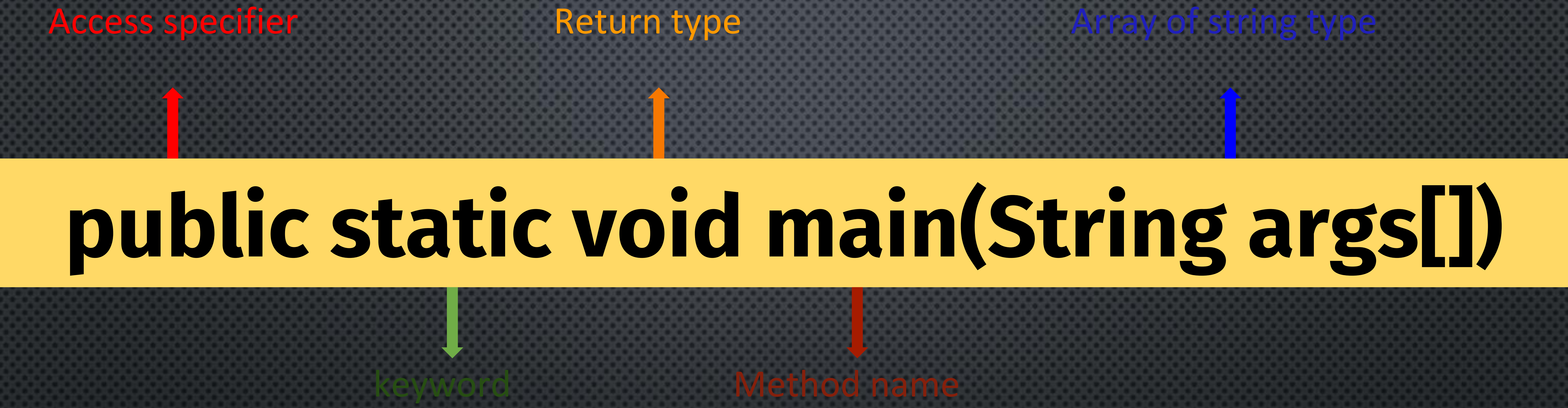
02

03

MY FIRST JAVA PROGRAM



```
public class MyFirstJavaProgram {


   /* This is my first java program.
    * This will print 'Hello World' as the output
    */

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



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

JAVA IDENTIFIERS

1 Definition

Names used for classes, variables and methods are called identifiers.

2 Examples

1. age
2. $salary
3. _value
4. __1_value

RULES FOR WRITING IDENTIFIER IN JAVA

RULE 01

All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).

RULE 02

After the first character identifiers can have any combination of characters.

RULE 03

A keyword cannot be used as an identifier.

RULE 04

Most importantly identifiers are case sensitive.

BASIC RULES IN JAVA

Case Sensitivity

Java is case sensitive, which means identifier Hello and hello would have different meaning in Java

Class Names

For all class names the first letter should be in Upper Case.

Example : class MyFirstJavaClass

Method Names

All method names should start with a Lower Case letter.

Example : public void myMethodName()

Case Sensitivity

Java is case sensitive, which means identifier Hello and hello would have different meaning in Java

Class Names

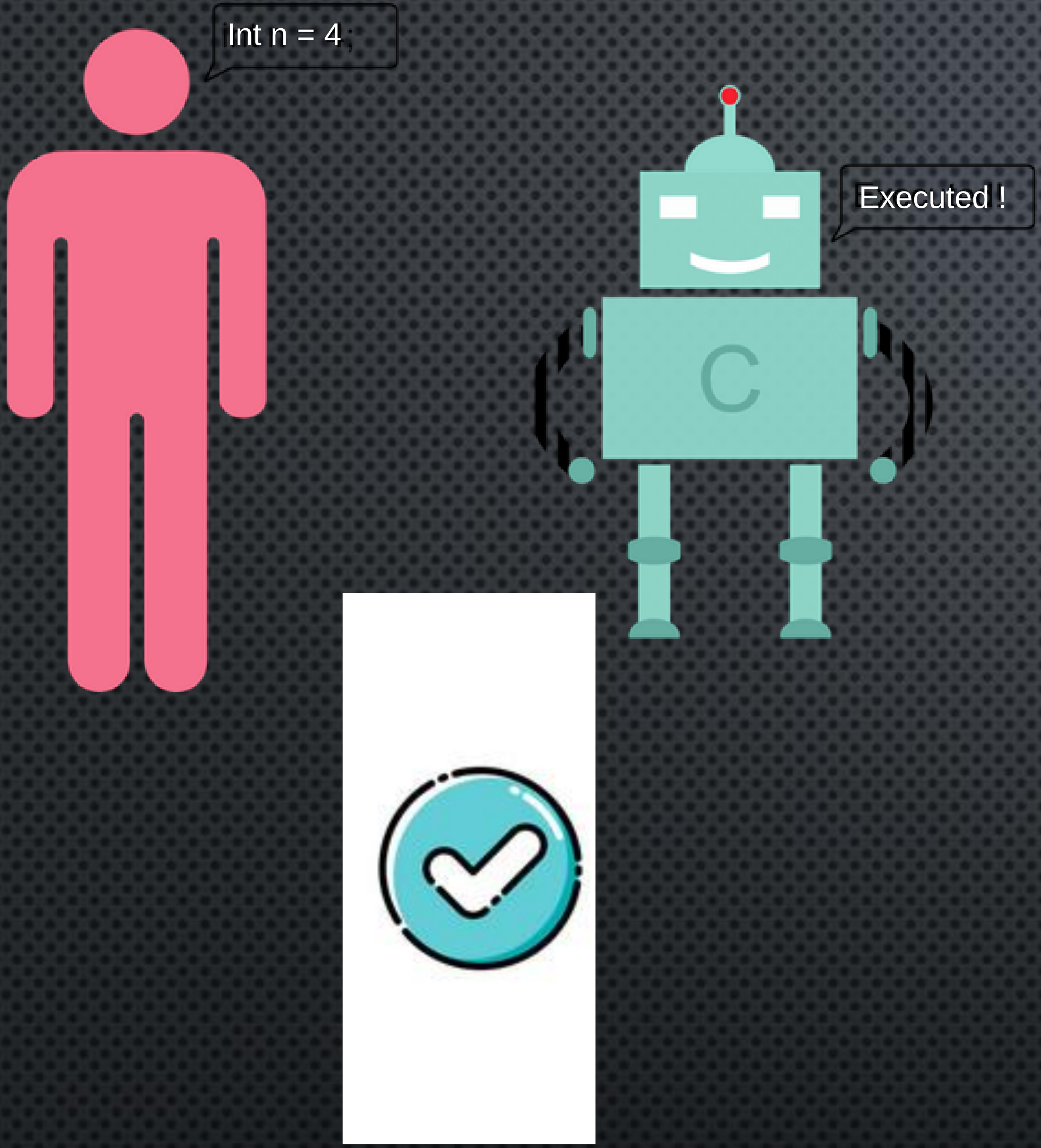
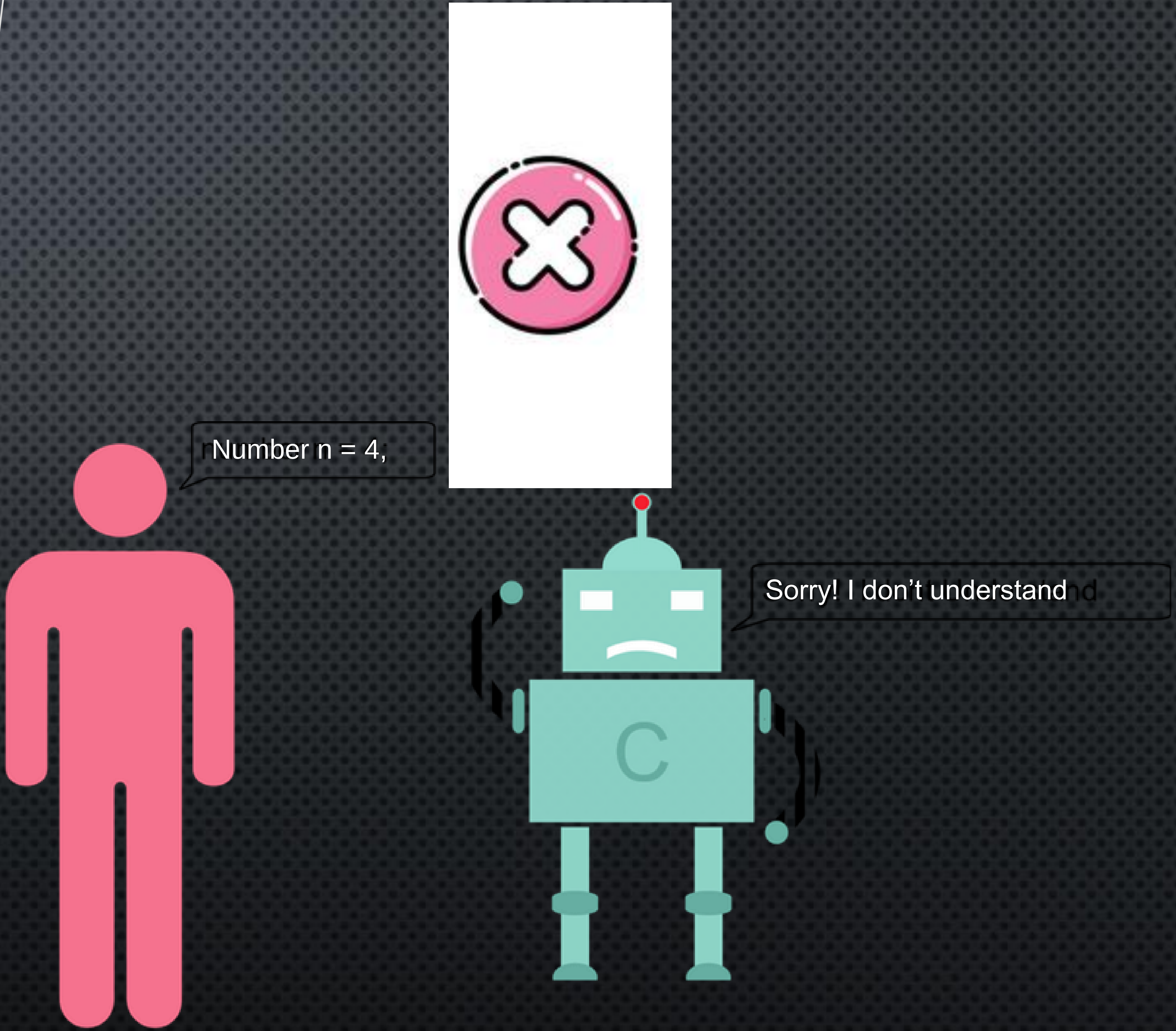
For all class names the first letter should be in Upper Case.

Example : class MyFirstJavaClass

JAVA KEYWORDS

1 Keywords are reserved words whose meaning is already known to compiler.

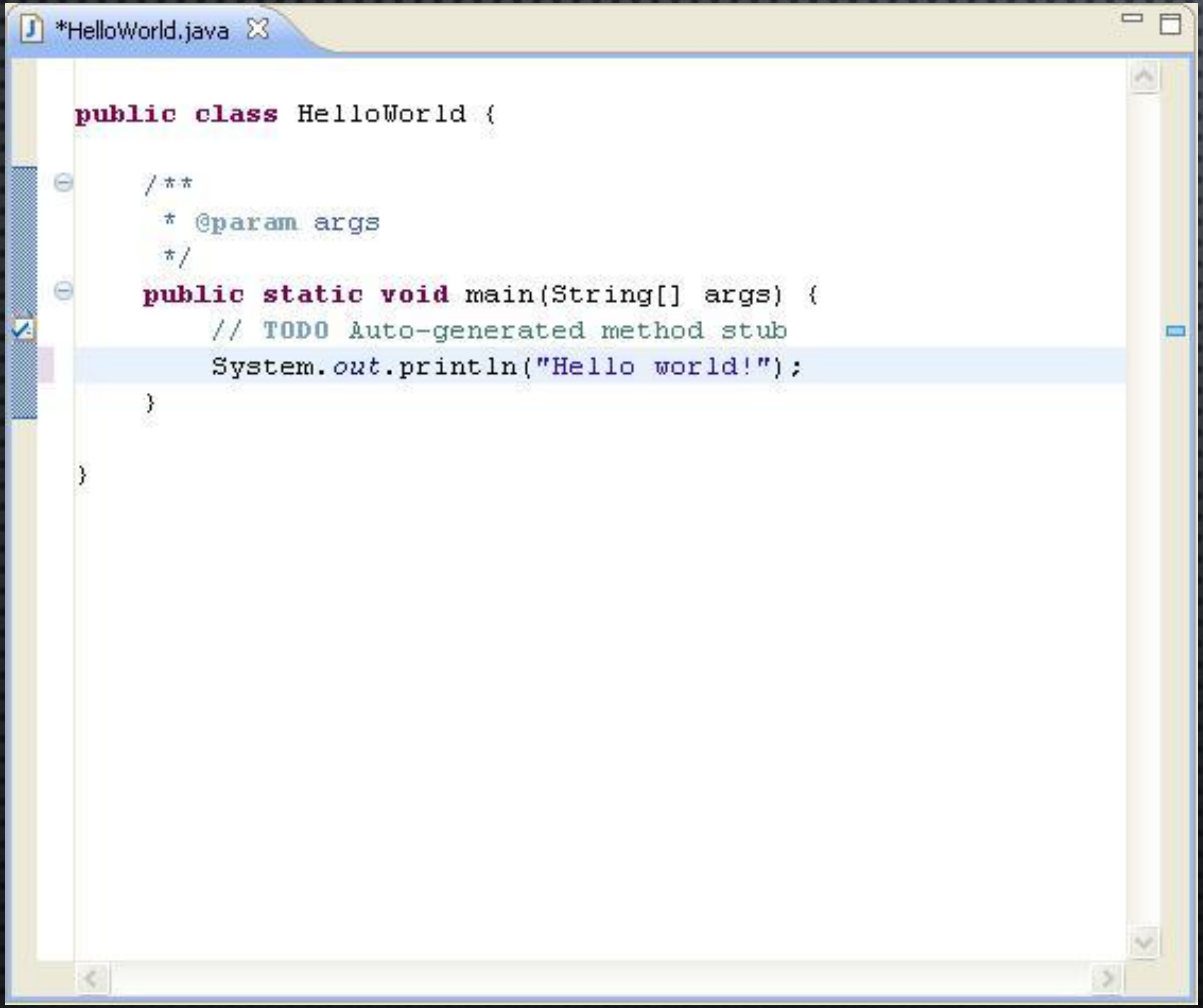
2 Keywords are part of the syntax and they cannot be used as an identifier.


Int n = 4
Executed !
C

Number n = 4,
Sorry! I don’t understand
C

LIST OF THE RESERVED KEYWORDS IN JAVA

abstract	assert	boolean	break	byte
A	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

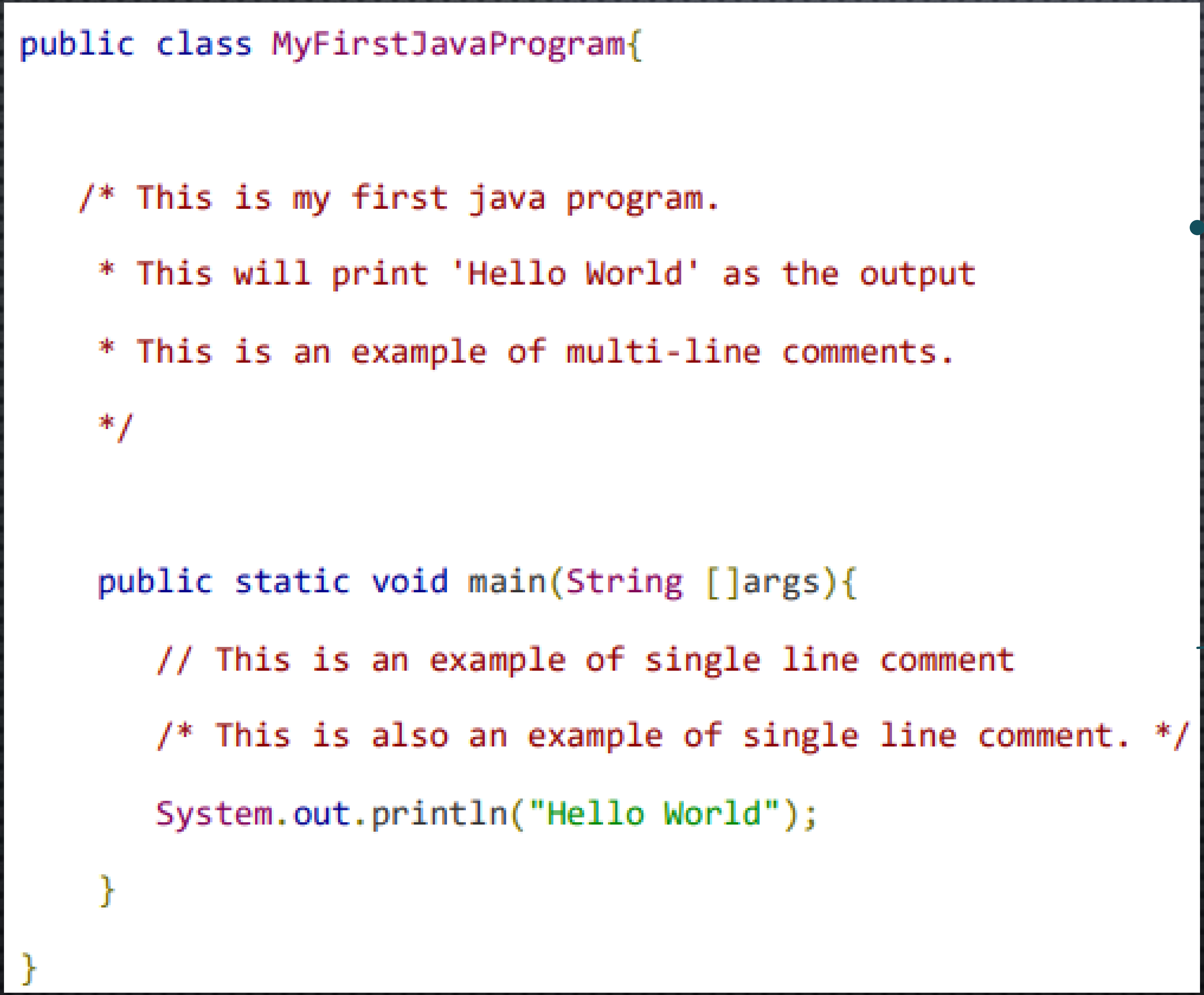
COMMENTS IN JAVA


*HelloWorld.java

```
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hello world!");
    }

}
```

Java supports single-line and multi-line comments very similar to c and c++.

All characters available inside any comment are ignored by Java compiler.



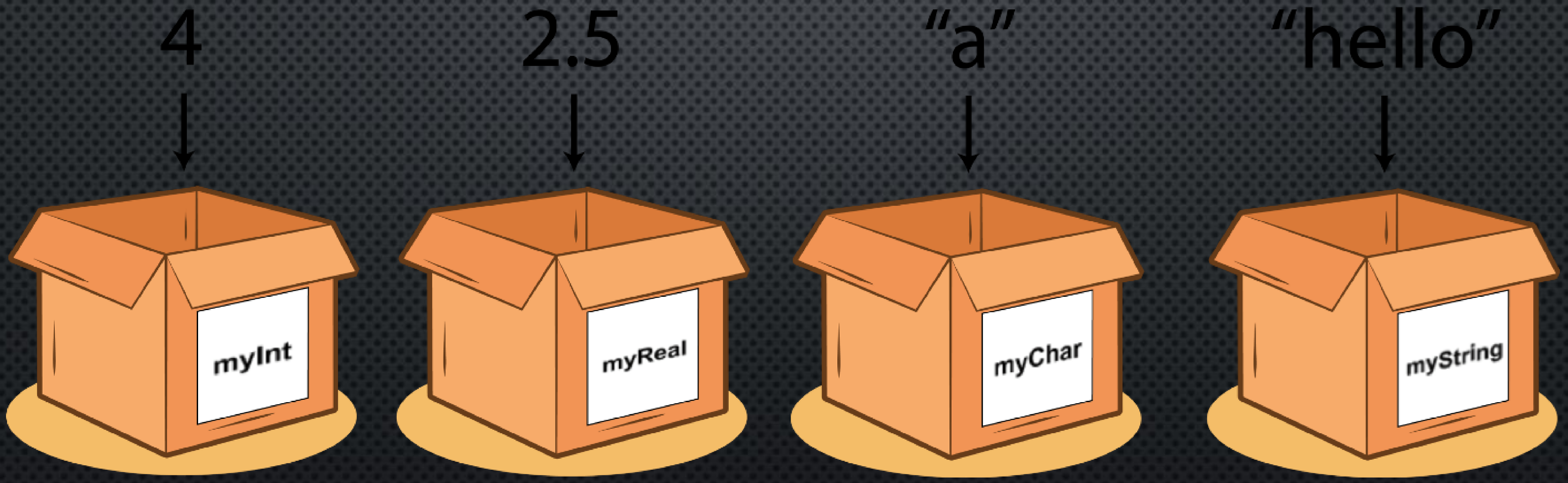
```java
public class MyFirstJavaProgram{


   /* This is my first java program.
    * This will print 'Hello World' as the output
    * This is an example of multi-line comments.
    */


    public static void main(String []args){
       // This is an example of single line comment
       /* This is also an example of single line comment. */
       System.out.println("Hello World");
    }
}
```

Multiline Comment

Single Line Comment

VARIABLES IN JAVA

A **variable** is a name given to memory location whose value can change throughout the program.Variable is a container that will store information.



TYPES OF VARIABLES

LOCAL VARIABLE 01

INSTANCE VARIABLE 02

CLASS OR STATIC VARIABLE 03

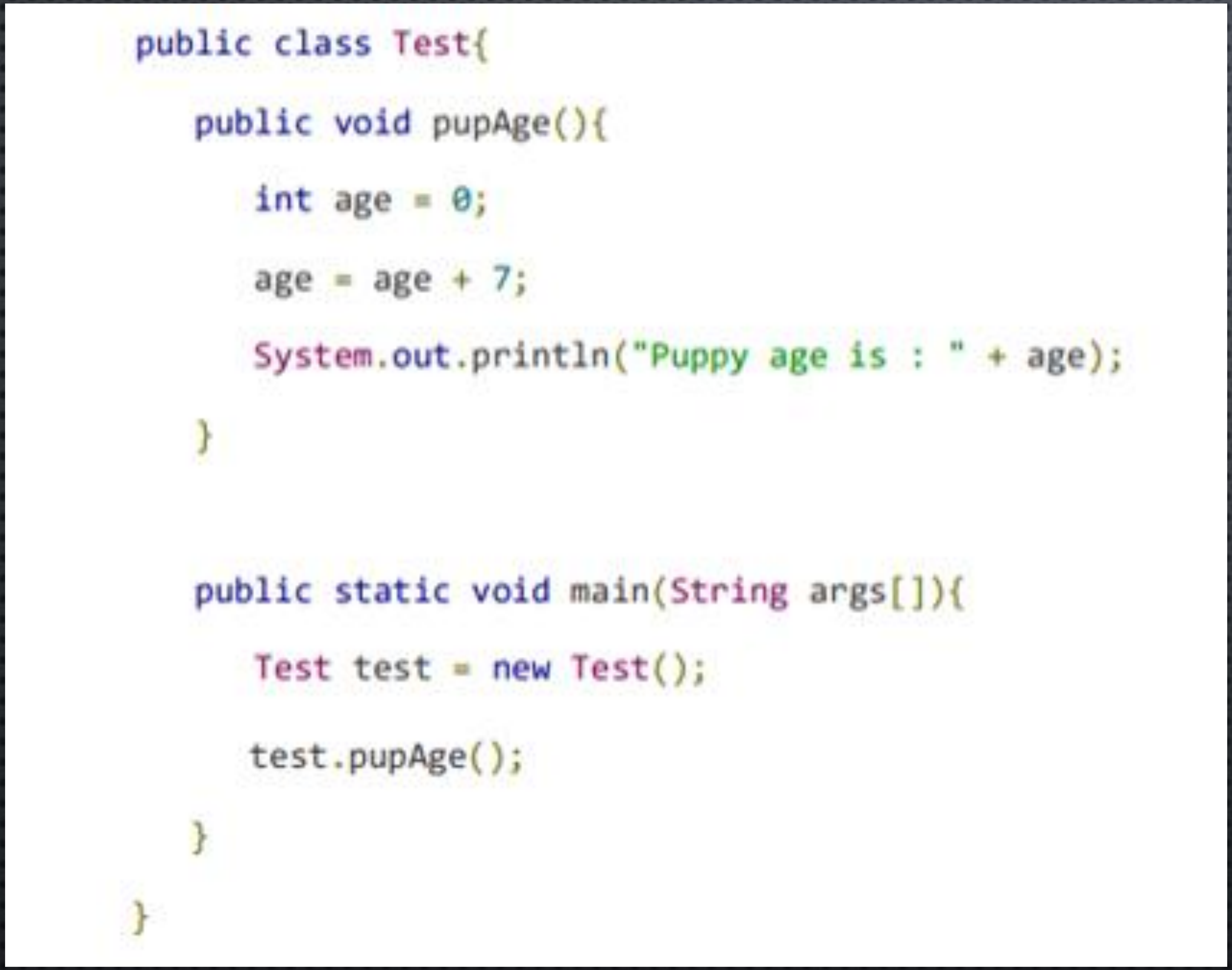
1.LOCAL VARIABLES

1

Local variables are declared inside methods, constructors, or blocks.
It is used for computation

2

Local variables will be destroyed once it exits the method, constructor or block.Local variables cannot be accessed from outside method, constructor or block



```
public class Test{

   public void pupAge(){

      int age = 0;

      age = age + 7;

      System.out.println("Puppy age is : " + age);

   }


   public static void main(String args[]){

      Test test = new Test();

      test.pupAge();

   }

}
```

EXAMPLE FOR LOCAL VARIABLE

Here, **age** is a **local variable.**

This is defined inside **pupAge()** method and its scope is limited to this method only.

INSTANCE VARIABLES IN JAVA

1

Each object has its unique set of instance variables.

2

An object's state is created by the values assigned to these instance variables.

2. INSTANCE VARIABLES

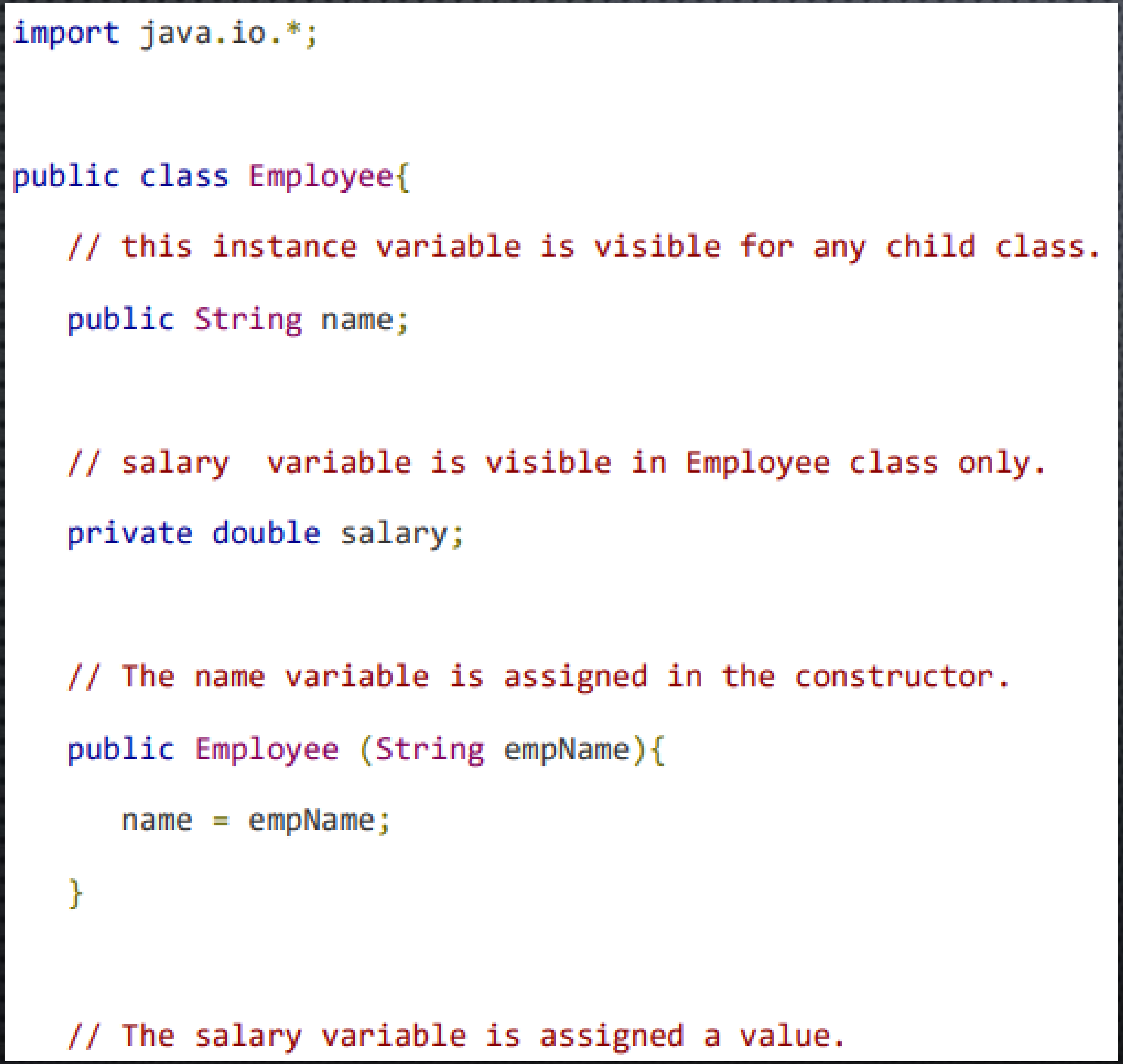
1

Instance variables are declared in a class, but outside a method, constructor or any block.

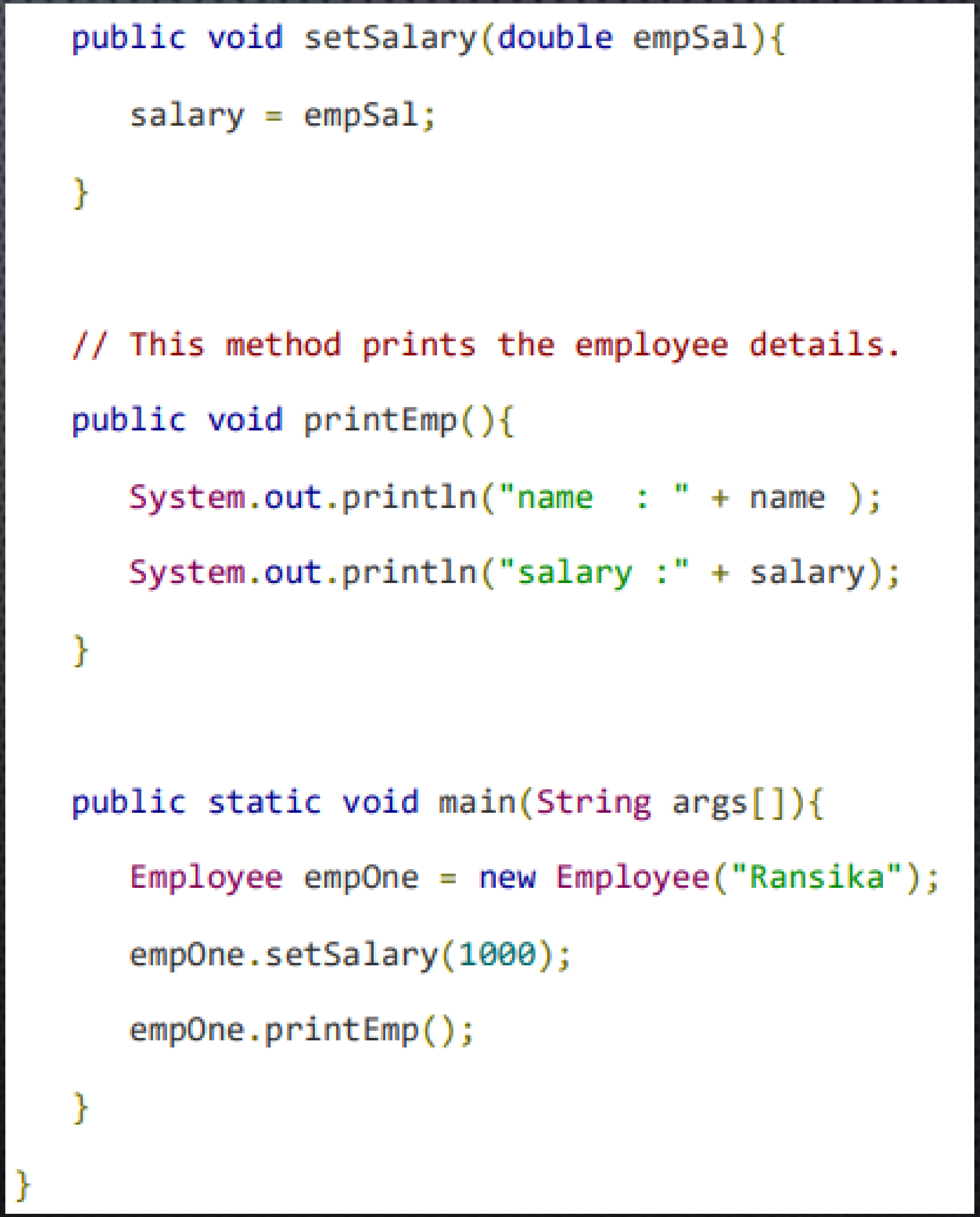
2

The instance variable is initialized when an object of the class is created using new keyword.Instance variable are accessed using object of class

EXAMPLE FOR INSTANCE VARIABLE



```java
import java.io.*;


public class Employee{

   // this instance variable is visible for any child class.

   public String name;


   // salary  variable is visible in Employee class only.

   private double salary;


   // The name variable is assigned in the constructor.

   public Employee (String empName){

      name = empName;

   }


   // The salary variable is assigned a value.
```



```java
   public void setSalary(double empSal){

      salary = empSal;

   }


   // This method prints the employee details.

   public void printEmp(){

      System.out.println("name  : " + name );

      System.out.println("salary :" + salary);

   }


   public static void main(String args[]){

      Employee empOne = new Employee("Ransika");

      empOne.setSalary(1000);

      empOne.printEmp();

   }

}
```

3. CLASS/STATIC VARIABLES

1

Class variables also known as static variables.Static variables only have one single copy which is shared by all.

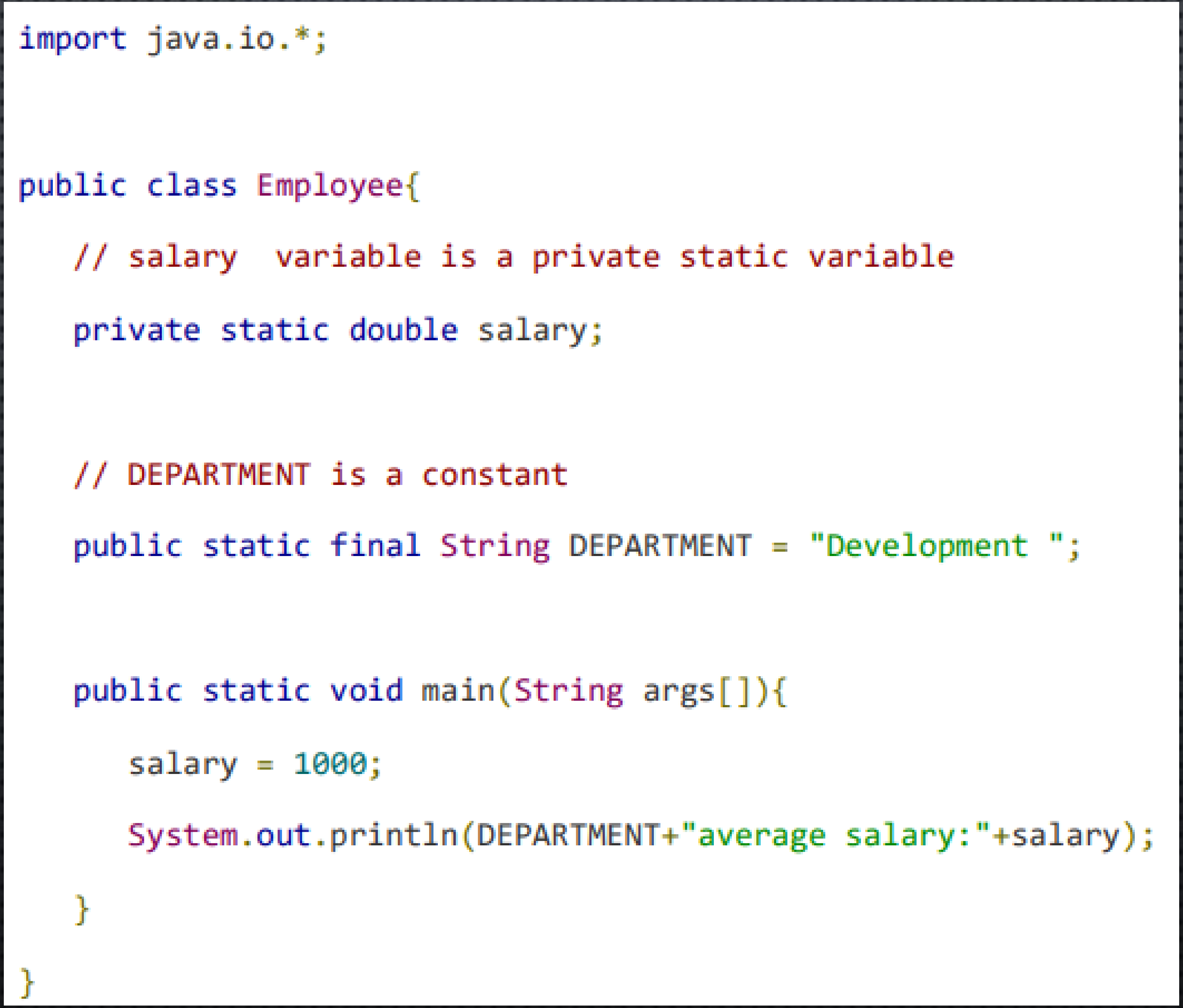
2

They are declared with the static keyword in a class, but outside a method, constructor or a block.

3

Static variables are created when the program starts and destroyed when the program stops.

EXAMPLE FOR CLASS/STATIC VARIABLE



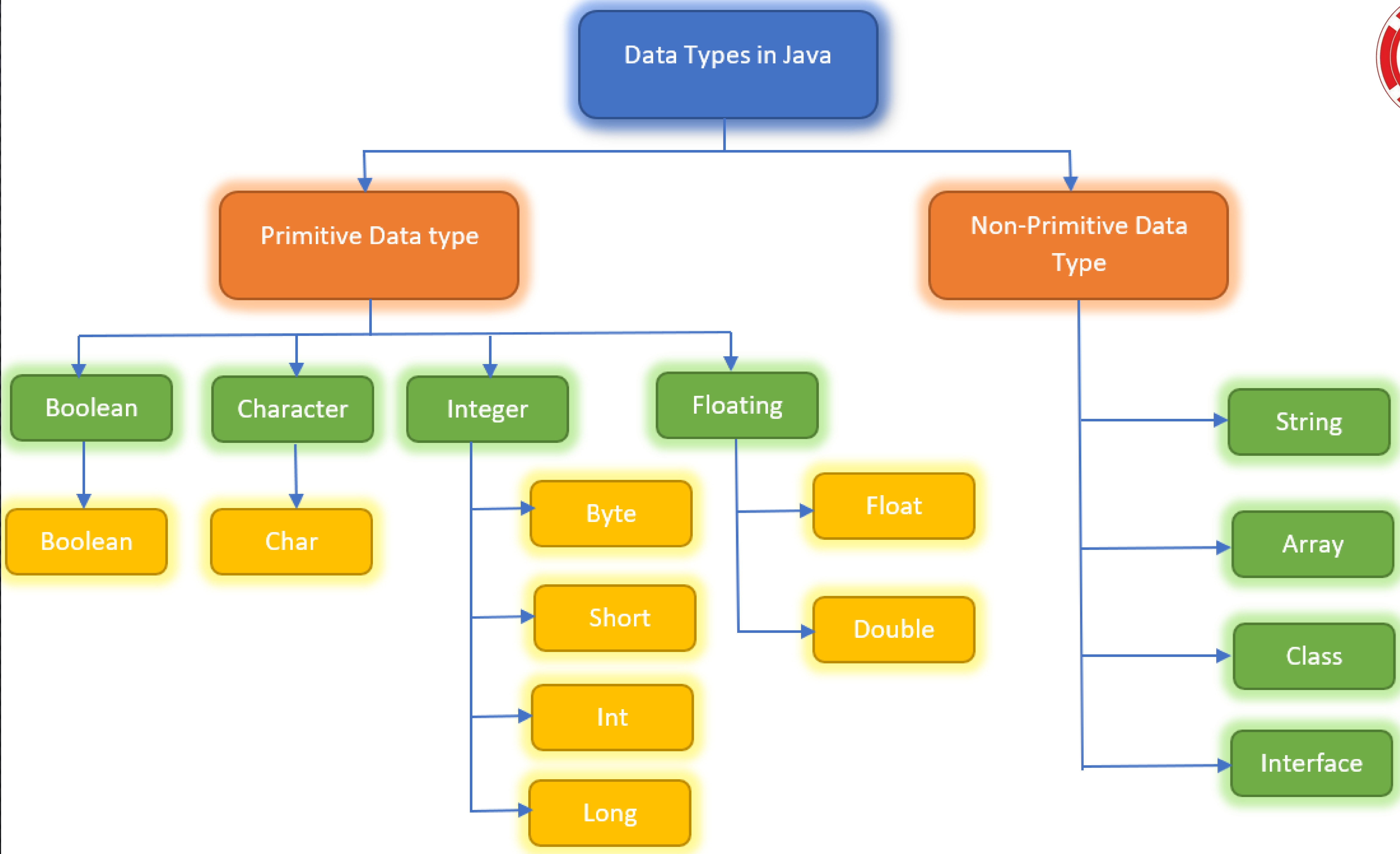
```
import java.io.*;


public class Employee{

   // salary  variable is a private static variable

   private static double salary;


   // DEPARTMENT is a constant

   public static final String DEPARTMENT = "Development ";


   public static void main(String args[]){

      salary = 1000;

      System.out.println(DEPARTMENT+"average salary:"+salary);

   }

}
```

DATA TYPES IN JAVA

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

DATA TYPES

- Primitive Data Types — 01
- Reference / Object Data Types — 02


Data Types in Java
Primitive Data type
Non-Primitive Data Type
Boolean
Character
Integer
Floating
Boolean
Char
Byte
Short
Int
Long
Float
Double
String
Array
Class
Interface

PRIMITIVE DATA TYPES IN JAVA

1 byte

Byte data type is an 8-bit signed two's complement integer.

2 short

Short data type is a 16-bit signed two's complement integer.

3 int

Int data type is a 32-bit signed two's complement integer.

4 long

Long data type is a 64-bit signed two's complement integer.

PRIMITIVE DATA TYPES IN JAVA

5 float

Float data type is a single-precision 32-bit IEEE 754 floating point.

6 double

Double data type is a double-precision 64-bit IEEE 754 floating point.

7 boolean

There are only two possible values: true and false.

8 char

Char data type is a single 16-bit Unicode character.

REFERENCE/OBJECT DATA TYPES IN JAVA

01 Array

02 String

03 Class

04 Interface

05 Enum

1 Array

- An array is used to hold elements of the same type.
- It is an object in java, and the array name is a reference value that carries the base address of the continuous location of elements of an array.

2. Class

- A class is a user-defined data type from which objects are created.
- It contains fields and methods that represent the behaviour of an object.
- A class gets invoked by the creation of the respective object.

3. Enum

- An enum, similar to a class has attributes and methods.
- However, in contrast to classes, enum constants are public, static, and final.
- The enum can implement interfaces.

4. String

- The String data type stores a sequence or array of characters

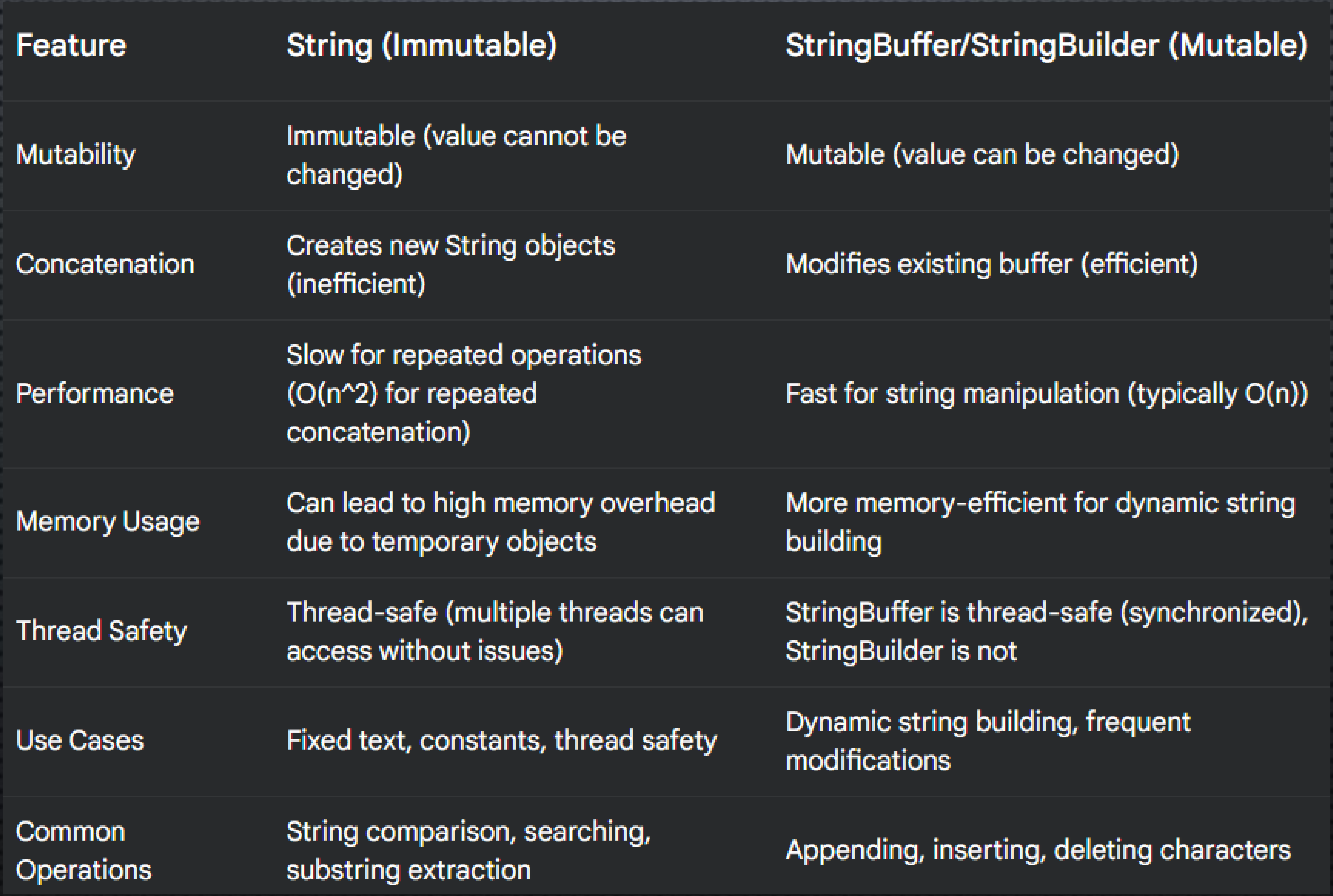
.

- A string is a non-primitive data type but it is predefined in Java. String literals are enclosed in double-quotes.

5. Interface

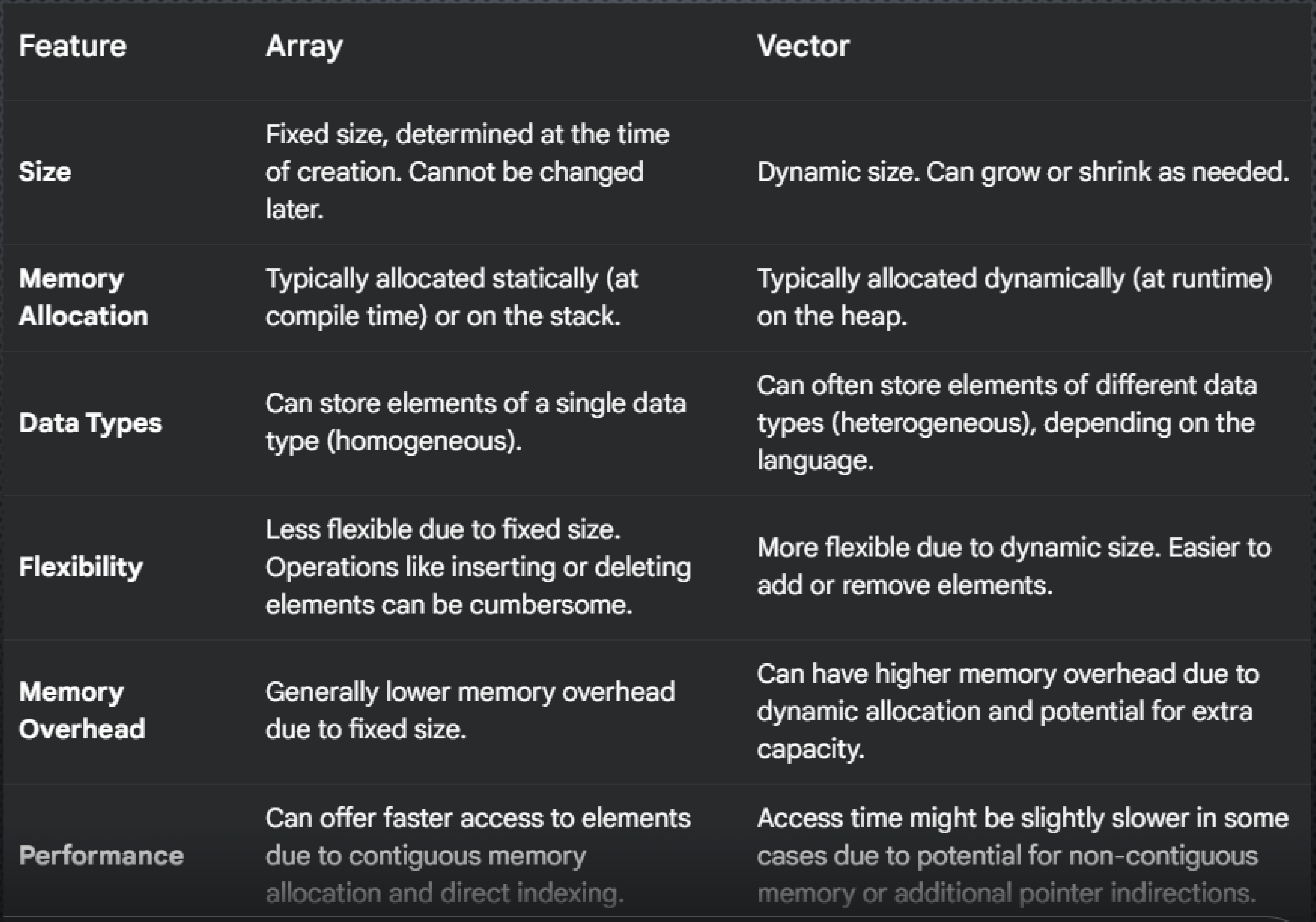
- An interface is declared like a class.
- The key difference is that the interface contains methods that are abstract by default; they have nobody.

Difference between String and StringBuffer



Feature	String (Immutable)	StringBuffer/StringBuilder (Mutable)
Mutability	Immutable (value cannot be changed)	Mutable (value can be changed)
Concatenation	Creates new String objects (inefficient)	Modifies existing buffer (efficient)
Performance	Slow for repeated operations (O(n^2) for repeated concatenation)	Fast for string manipulation (typically O(n))
Memory Usage	Can lead to high memory overhead due to temporary objects	More memory-efficient for dynamic string building
Thread Safety	Thread-safe (multiple threads can access without issues)	StringBuffer is thread-safe (synchronized), StringBuilder is not
Use Cases	Fixed text, constants, thread safety	Dynamic string building, frequent modifications
Common Operations	String comparison, searching, substring extraction	Appending, inserting, deleting characters

Difference between array and vector



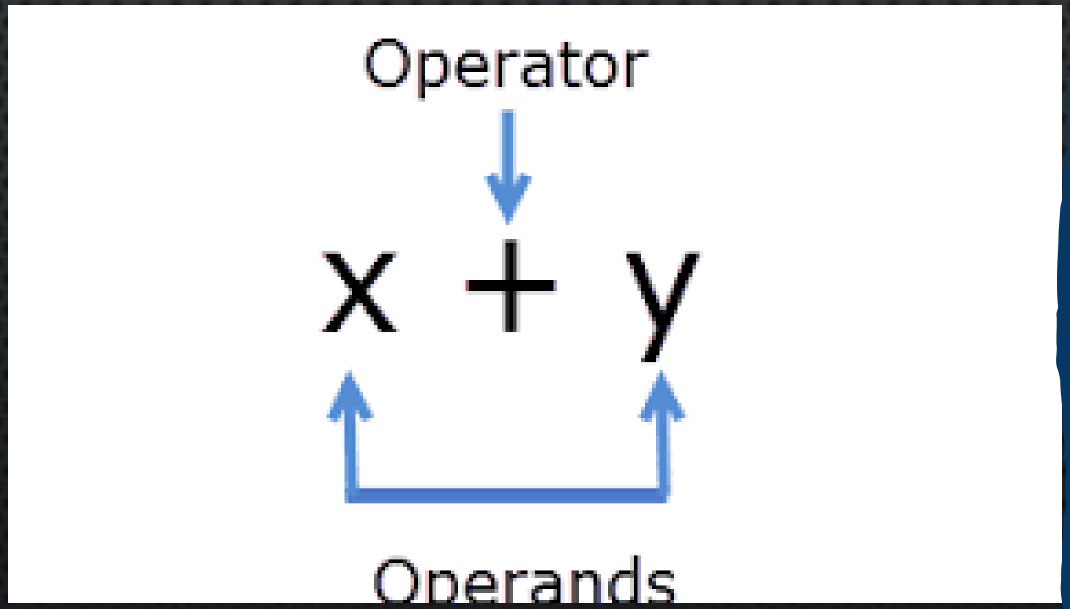
Feature	Array	Vector
Size	Fixed size, determined at the time of creation. Cannot be changed later.	Dynamic size. Can grow or shrink as needed.
Memory Allocation	Typically allocated statically (at compile time) or on the stack.	Typically allocated dynamically (at runtime) on the heap.
Data Types	Can store elements of a single data type (homogeneous).	Can often store elements of different data types (heterogeneous), depending on the language.
Flexibility	Less flexible due to fixed size. Operations like inserting or deleting elements can be cumbersome.	More flexible due to dynamic size. Easier to add or remove elements.
Memory Overhead	Generally lower memory overhead due to fixed size.	Can have higher memory overhead due to dynamic allocation and potential for extra capacity.
Performance	Can offer faster access to elements due to contiguous memory allocation and direct indexing.	Access time might be slightly slower in some cases due to potential for non-contiguous memory or additional pointer indirections.

OPERATORS IN JAVA

1

Definition

The symbols which are used to perform logical and mathematical operations in a Java program are called Operators.



2

Example

For example:-

'+' is an operator to perform addition.

TYPES OF OPERATORS

Arithmetic Operators

Logical Operators

Relational Operators

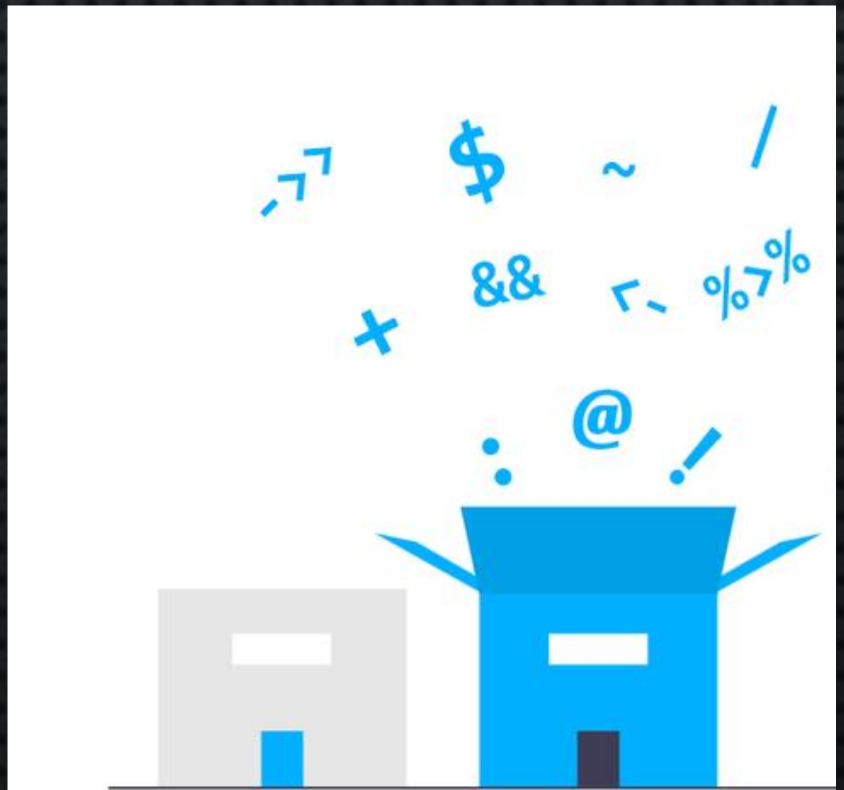
Assignment Operators

Bitwise Operators

Miscellaneous Operators

ARITHMETIC OPERATORS

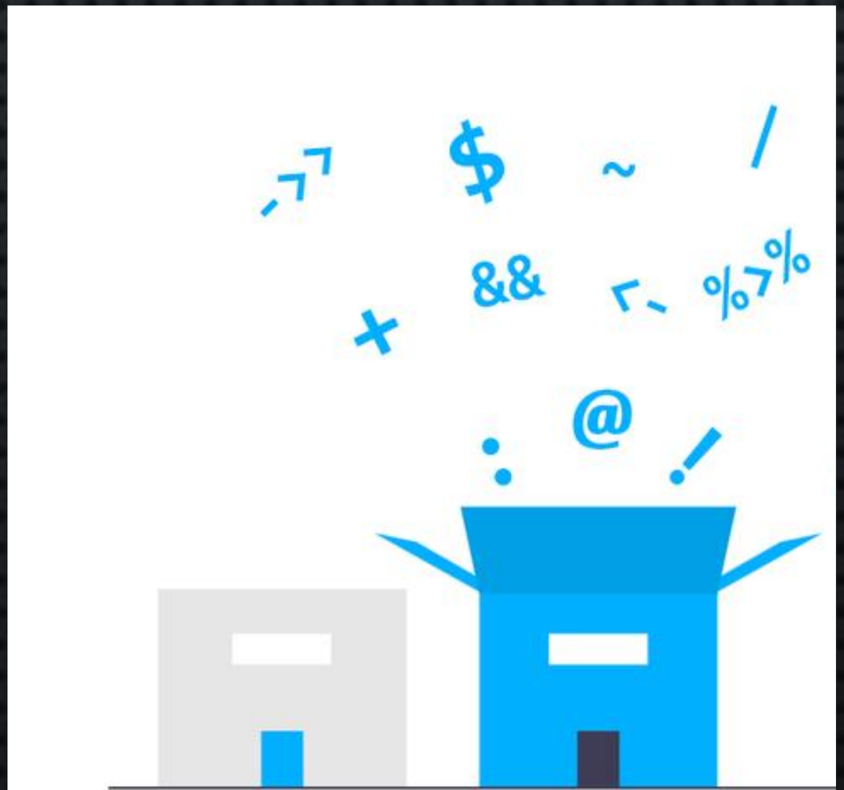
Arithmetic operators are used in **mathematical expressions** in the same way that they are used in algebra.



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

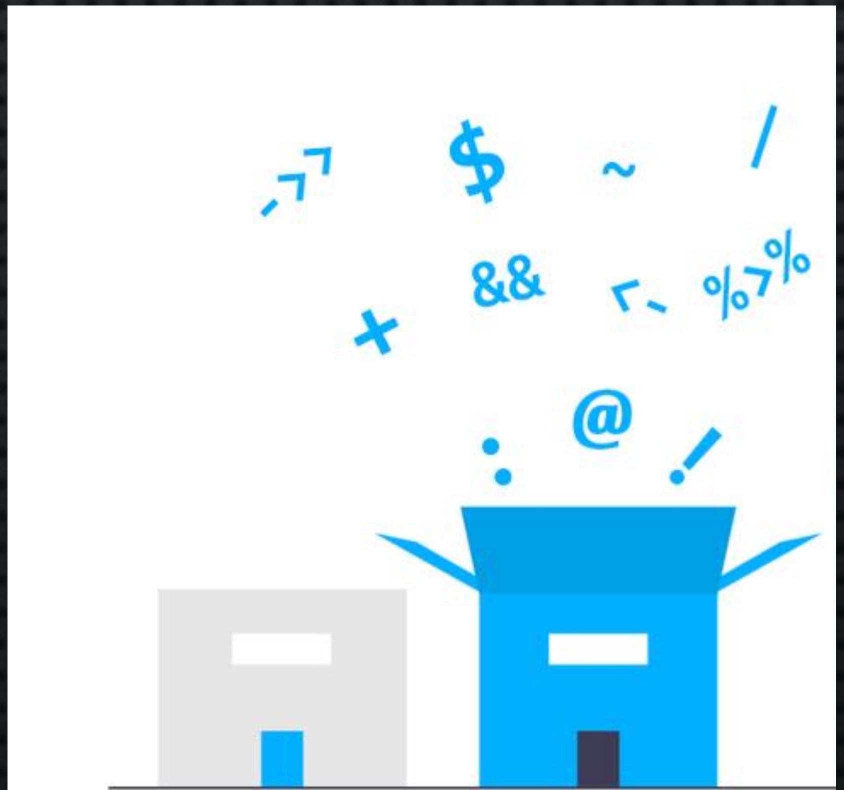
RELATIONAL OPERATORS

Relational operators are used **to check** the **relationship** between two operands.



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.

BITWISE OPERATORS

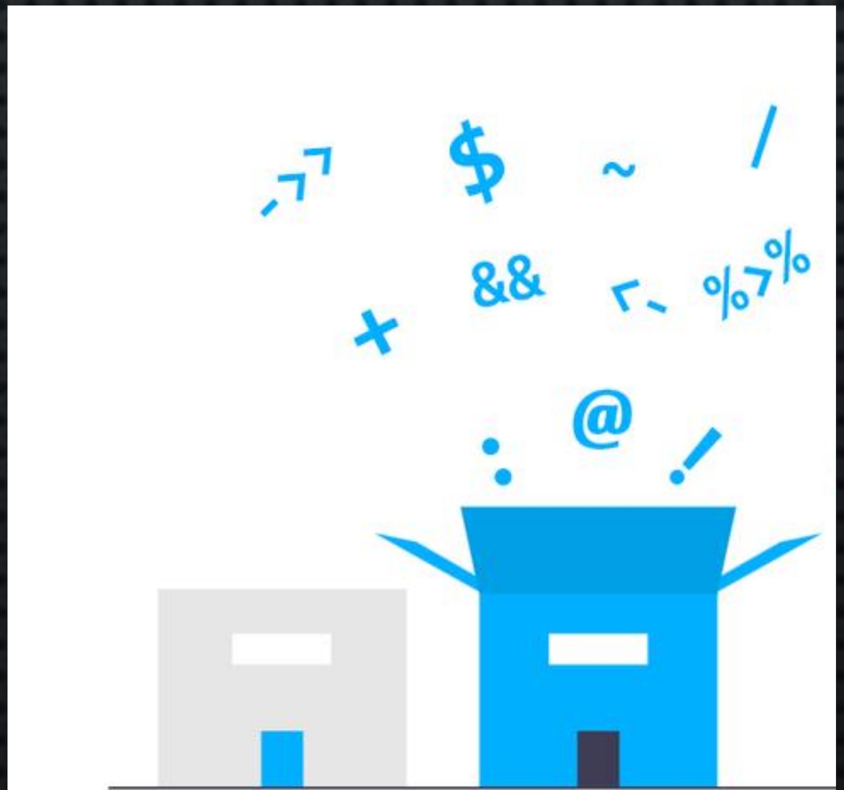


Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte. **Bitwise operator** works on bits and **performs bit-by-bit operation.**

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

LOGICAL OPERATORS

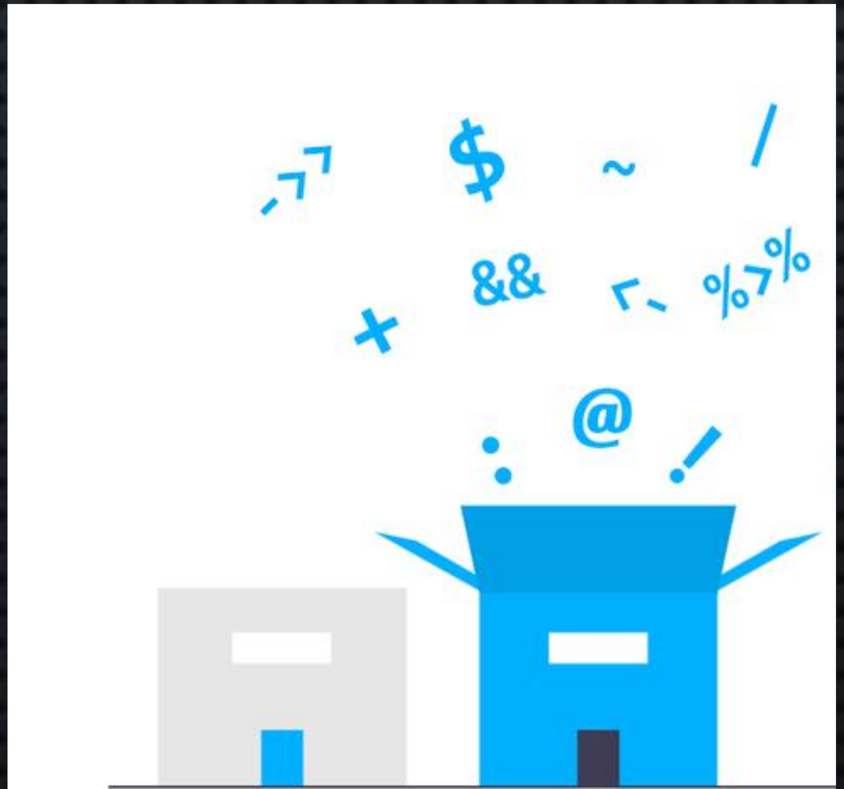
Logical operators are used **to check** whether the expression is **true or false.**



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

ASSIGNMENT OPERATORS

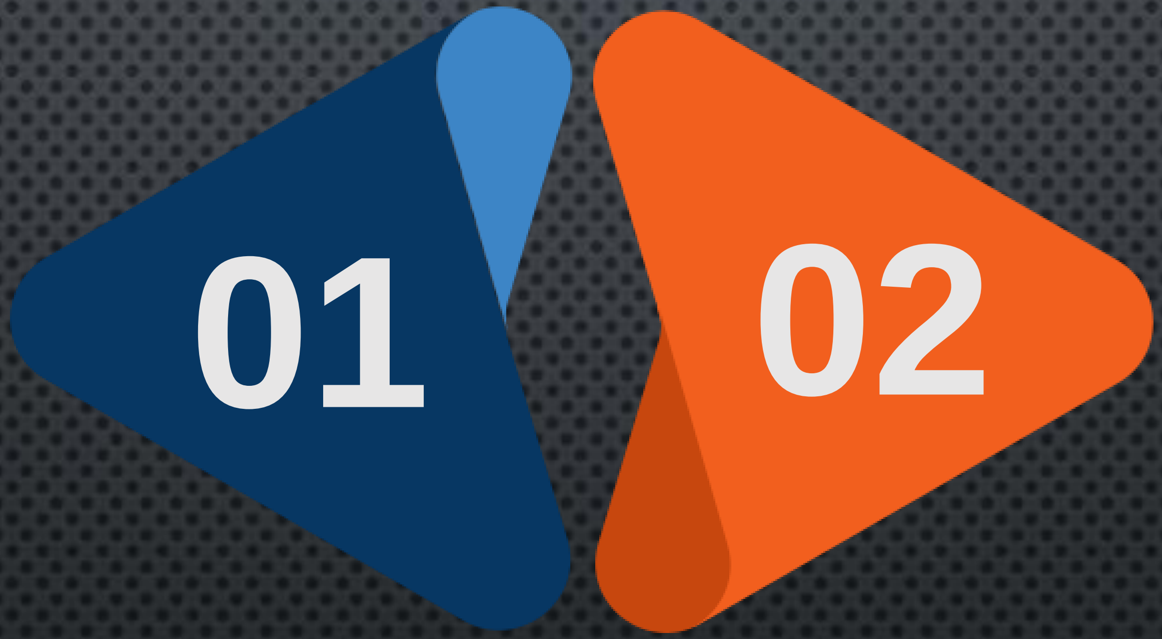
Assignment operators are used in Java **to assign values to** variables.



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

Operator	Description	Example
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

MISCELLANEOUS OPERATORS



Conditional Operator (? :)

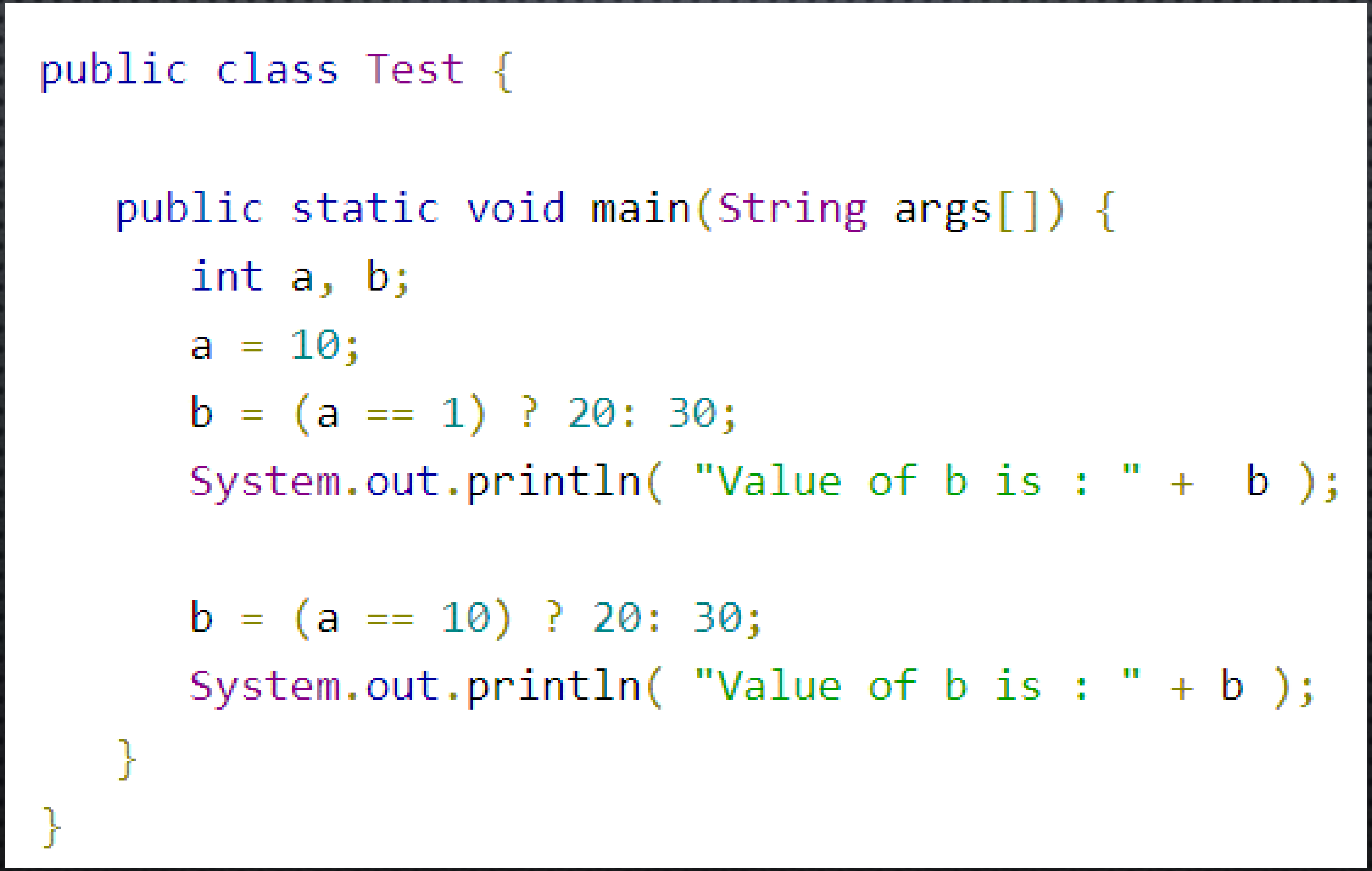
Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions.

instanceof Operator

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type).

Syntax for conditional operator:

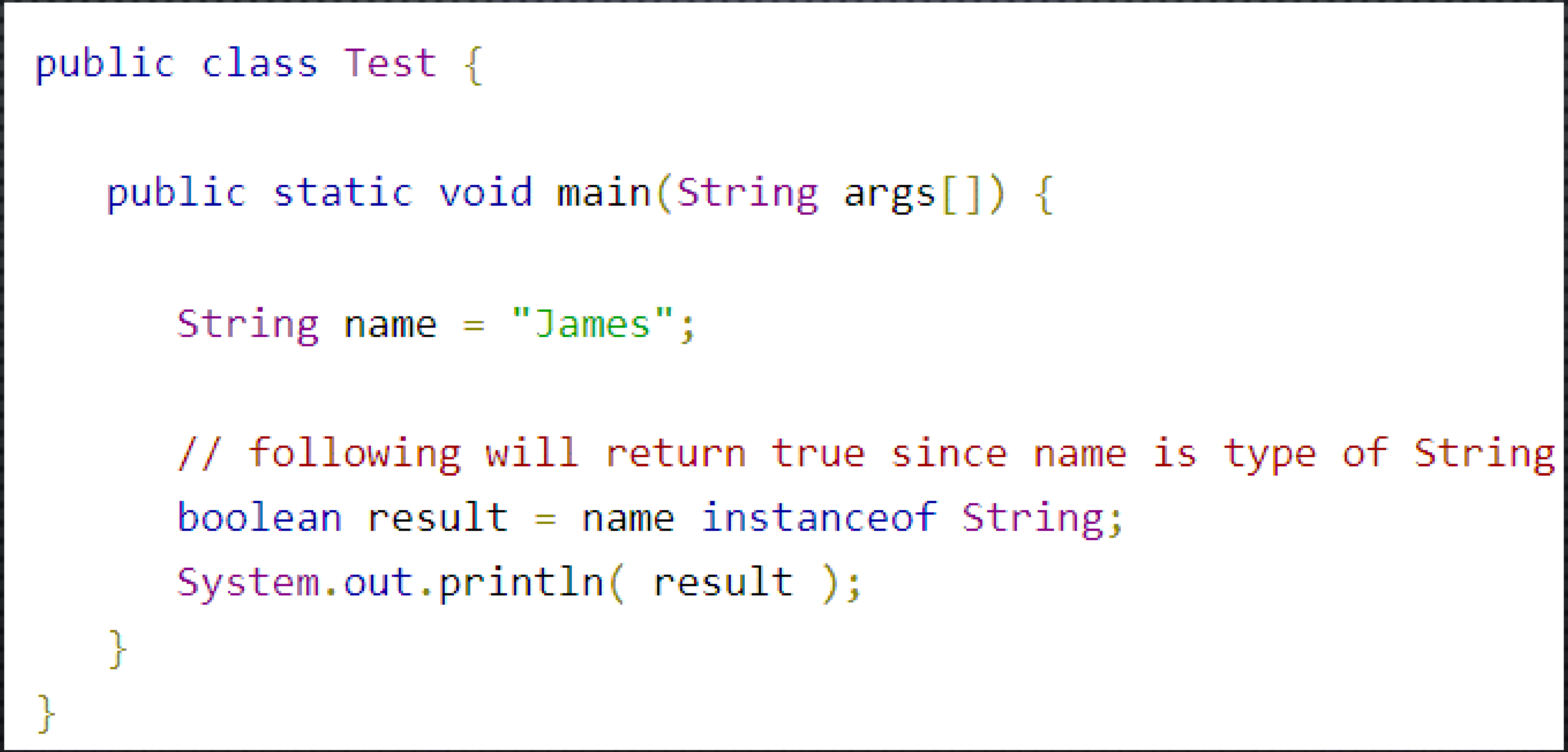
variable x = (expression) ? value if true : value if false



```java
public class Test {

   public static void main(String args[]) {
      int a, b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

Syntax for instanceOf operator:

(Object reference variable) instanceof (class/interface type)



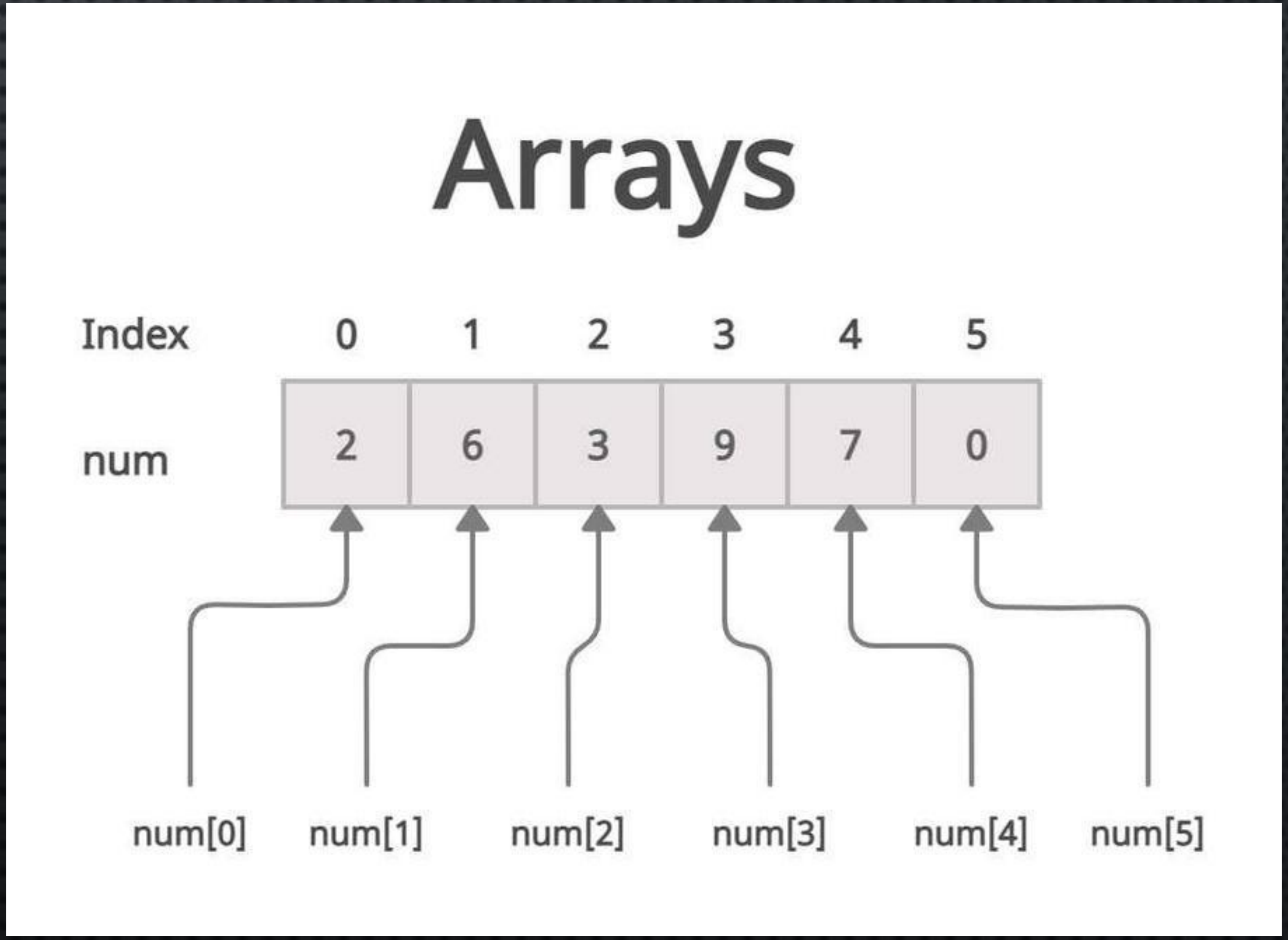
```
public class Test {

   public static void main(String args[]) {

      String name = "James";

      // following will return true since name is type of String
      boolean result = name instanceof String;
      System.out.println( result );
   }
}
```

ARRAY

- Definition: An array is a collection of elements of the same data type stored in contiguous memory locations.
- Key Points:
- Fixed size, zero-based indexing.
- Types: One-dimensional, Two-dimensional, and Multi-dimensional arrays.

```
int[] numbers = {1, 2, 3, 4};
```

- Advantages: Easy to access and manage data.
- Limitations: Fixed size, cannot store heterogeneous data.



STRING

- Definition: A sequence of characters treated as a single data type in Java.
- Key Features: Immutable in Java.
- Common Methods: length(), charAt(), substring(), concat().
- Example:

```
String str = "Hello World";
System.out.println(str.length());
```

Uses: Storing and manipulating text data.

STRING BUFFER CLASSES

- Definition: Mutable sequences of characters.
- Key Features:
- Modifiable string.
- Thread-safe.
- Common Methods: append(), insert(), replace(), reverse().
- Example:

```
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World");
System.out.println(sb);
```

VECTOR

- Definition: A dynamic array that can grow or shrink in size.
- Key Features:
- Can store heterogeneous elements.
- Synchronized (thread-safe).
- Common Methods: add(), remove(), size(), capacity().
- Example:

```
Vector<Integer> v = new Vector<>();
v.add(1);
v.add(2);
```

WRAPPER CLASSES

- Definition: Convert primitive data types into objects.
- Examples:
- Integer for int.
- Double for double.
- Common Methods: valueOf(), parseInt().
- Usage: Autoboxing, unboxing.
- Example:

```
Integer num = Integer.valueOf(5);
int val = num.intValue();
```

Constructors and Methods

- Constructors: Special methods to initialize objects.
  - No return type.
  - Same name as the class.
- Methods: Perform specific actions. Can return values.
  - Syntax: returnType methodName(parameters).

Types of Constructors

- Default Constructor: No parameters.
- Parameterized Constructor: Accepts parameters to initialize fields.
- Copy Constructor: Copies values from another object.

Method and Constructor Overloading

- Definition: Multiple methods or constructors with the same name but different parameters.
- Purpose: Increase flexibility and code readability.
- Example:

```
void display(int a);
void display(int a, int b);
```

PROGRAM

```
class OverloadingExample {
    // Constructor Overloading
    OverloadingExample() {
        System.out.println("Default Constructor");
    }

    OverloadingExample(int x) {
        System.out.println("Parameterized Constructor: " + x);
    }

    // Method Overloading
    void display() {
        System.out.println("No arguments");
    }

    void display(int x) {
        System.out.println("With one argument: " + x);
    }

public static void main(String[] args) {
 OverloadingExample obj1 = new OverloadingExample();
 OverloadingExample obj2 = new OverloadingExample(10);

 obj1.display();
 obj1.display(20);
 }
}
```

Nesting of methods

- Definition: One method calls another method within the same class.
- Purpose: Code reusability and modularity.
- Example:

```
void method1() {
    method2();
}
void method2() {
    System.out.println("Method called");
}
```

Command-Line Arguments

- Definition: Arguments passed to the main method when the program is executed.
- Syntax: public static void main(String[] args).
- Example: System.out.println(args[0]);

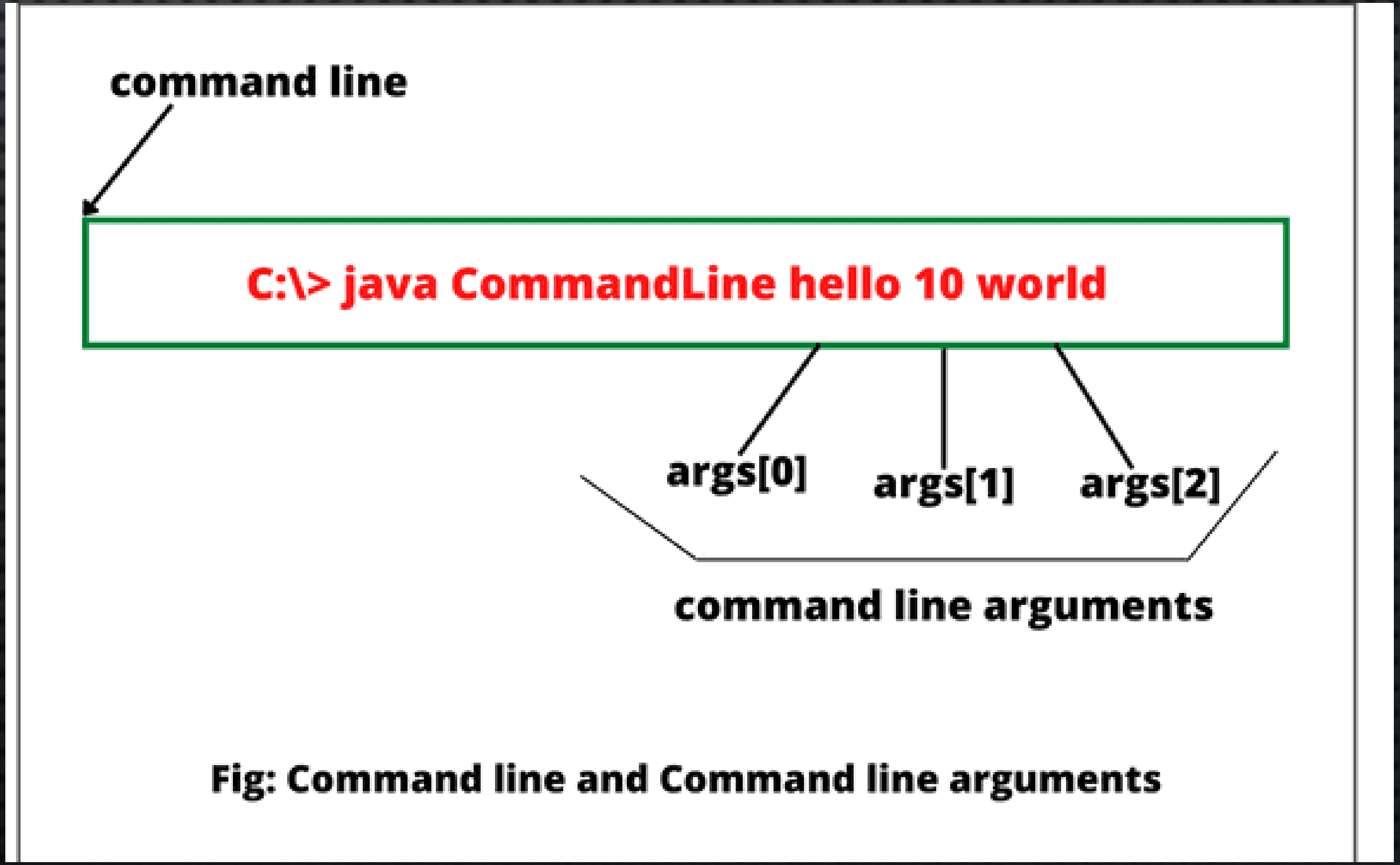


Fig: Command line and Command line arguments

Garbage Collection

- Definition: Automatic memory management by reclaiming unused objects.
- Key Features:
- Performed by the JVM.
- Method: System.gc().
- Advantages: Prevents memory leaks.

Visibility Control

- Definition: Controls access to class members.

Access Modifiers:

- private: Accessible only within the class.
- default: Accessible within the package.
- protected: Accessible within the package and subclasses.
- public: Accessible everywhere.
- private protected: is an access modifier that allows a member to be accessible only within its declaring class and by derived classes that are in the same assembly.

Access control for members of class and interface in java

Access Specifier \ Accessibility Location	Same Class	Same Package		Other Package	
		Child class	Non-child class	Child class	Non-child class
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No

Thankyou :)