OBJECT ORIENTED PROGRAMMING

Abdullah Bin Kasem Bhuiyan

Topic 2: Data types, declarations, and variables in Java

VARIABLES

❑A variable is a named memory location capable of storing data

❑As we have already seen, ***object variables*** refer to objects, which are created by instantiating classes with the new operator

❑We can also store data in ***simple variables***, which represent data only, without any associated methods

DATA DECLARATION SYNTAX

❑ The syntax for the declaration of a variable is:

Data type identifier;

✓ "data type" may be the name of a class, as we have seen, or may be one of the simple types, which we'll see in a moment

✓ "identifier" is a legal Java identifier; the rules for simple variable identifiers are the same as those for object identifiers

VARIABLE DECLARATION: EXAMPLES

For example:

```
int age;                 // int means integer
double cashAmount;       // double is a real #
```

We can also declare multiple variables of the same type using a single instruction; for example:

```
int x, y, z; //     or
int      x,
              y,
              z;
```

The second way is preferable, because it’s easier to document the purpose of each variable this way.

DATA TYPES

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

- ❑**Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- ❑**Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

JAVA PRIMITIVE DATA TYPES

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in java.

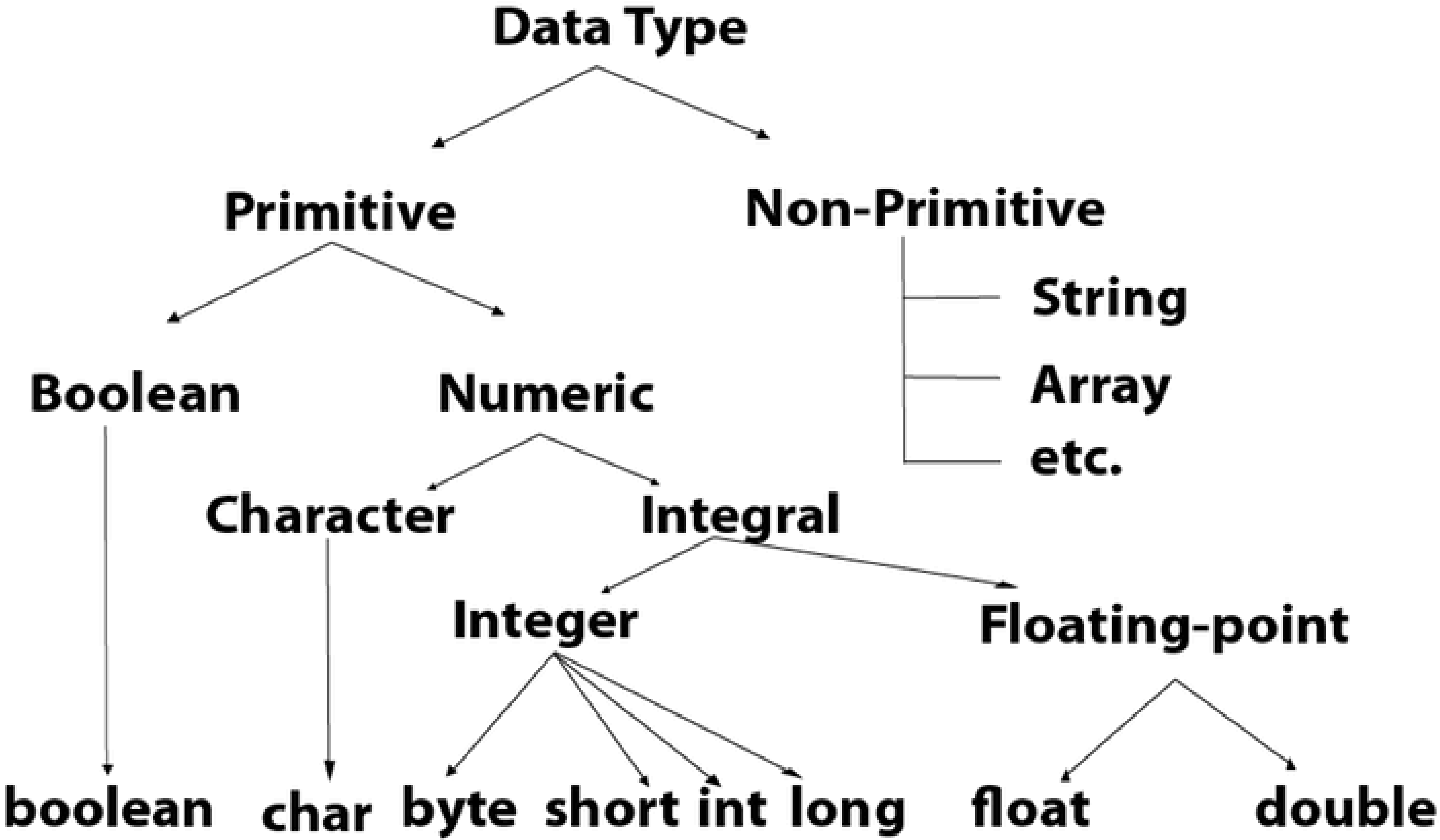
here are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

NON-PRIMITIVE DATA TYPE OR REFERENCE DATA TYPES

The **Reference Data Types** will contain a memory address of variable value because the reference types won't store the variable value directly in memory. They are strings, objects, arrays, etc.

- ❑String
- ❑Array
- ❑Class
- ❑Object
- ❑Interface



Data Type

Primitive

Non-Primitive

- String
- Array
- etc.

Boolean

Numeric

Character

Integral

Integer

Floating-point

boolean

char

byte

short

int

long

float

double

TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	‘a’, ‘\u0041’, ‘\101’, ‘\\’, ‘\’,’\n’,’ β’	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

OPERATORS

Operator in Java is a symbol which is used to perform operations. For example: +, -, *, / etc.

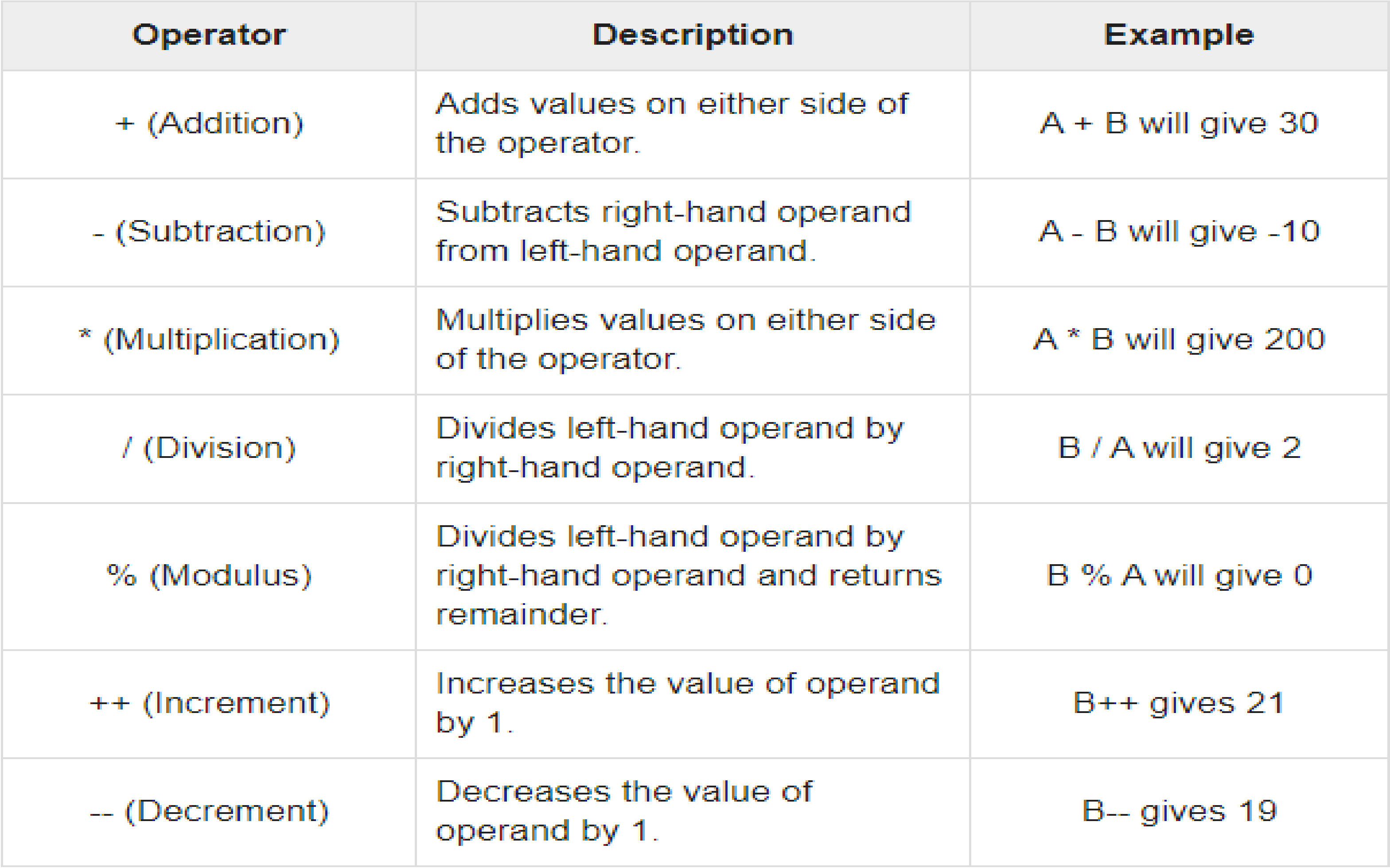
We can divide all the Java operators into the following groups −

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators

THE ARITHMETIC OPERATORS

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

Assume integer variable A holds 10 and variable B holds 20, then −

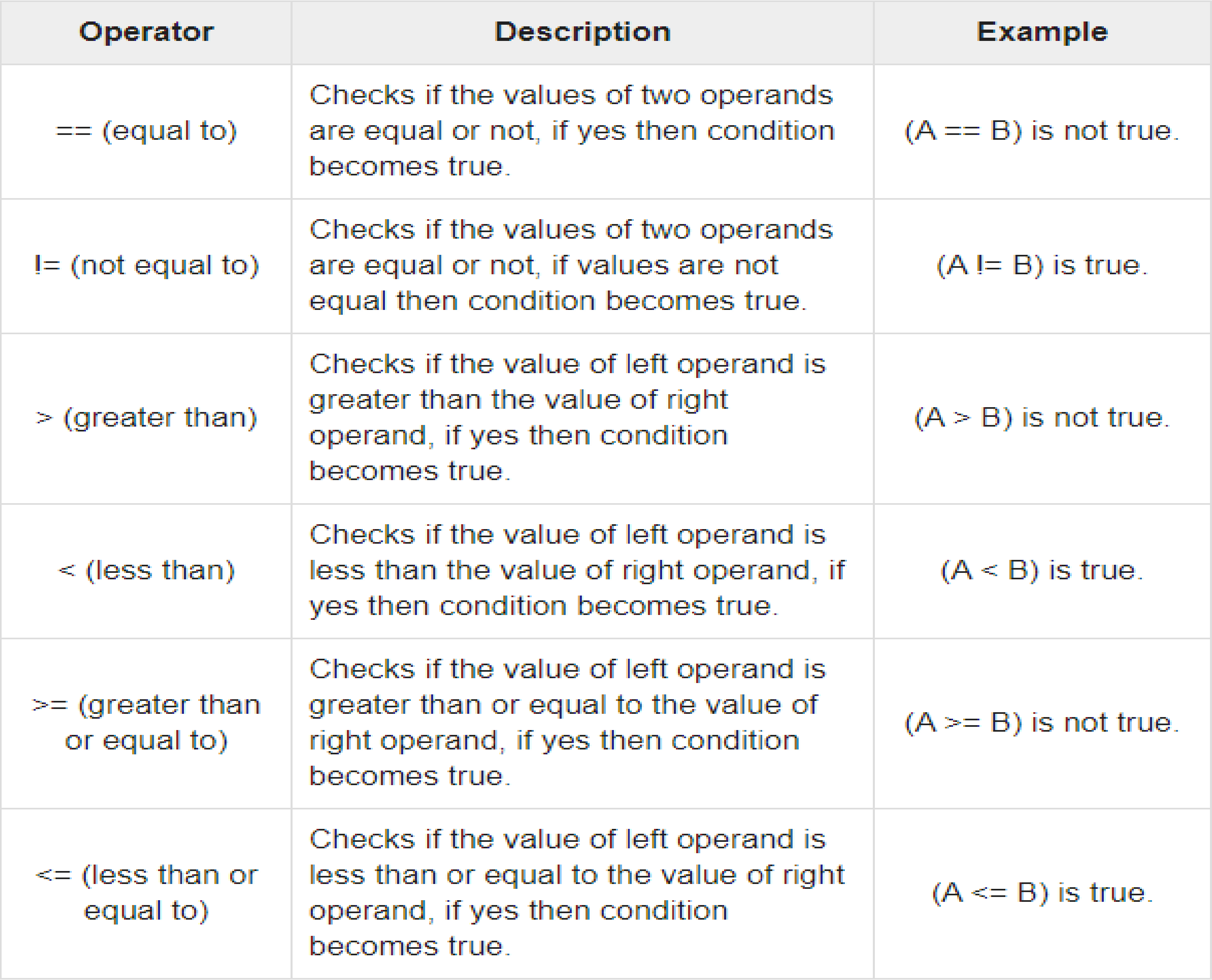


Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides 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

THE RELATIONAL OPERATORS

There are following relational operators supported by Java language.

Assume variable A holds 10 and variable B holds 20, then −

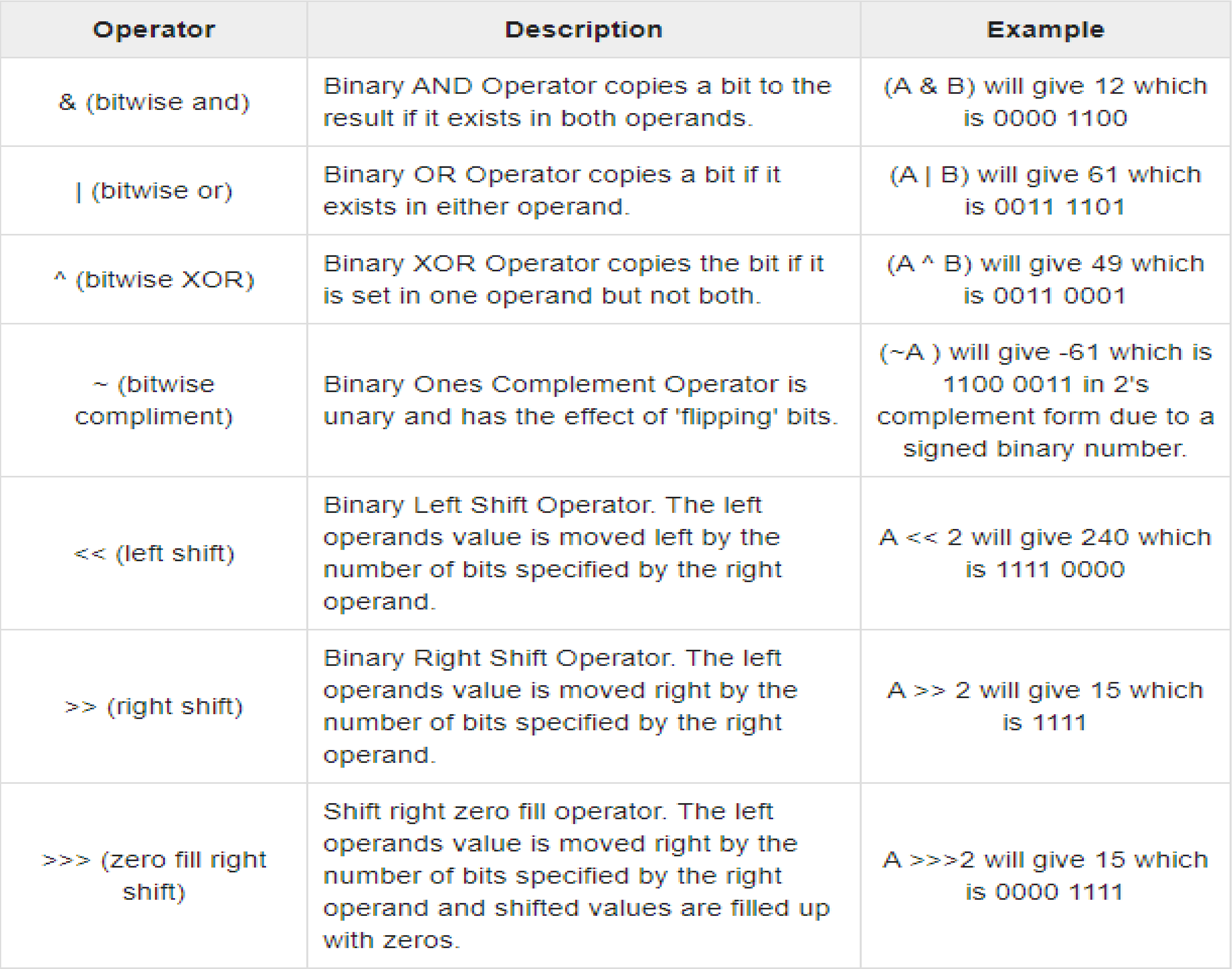


Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (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.

THE BITWISE OPERATORS

Bitwise operator works on bits and performs bit-by-bit operation.

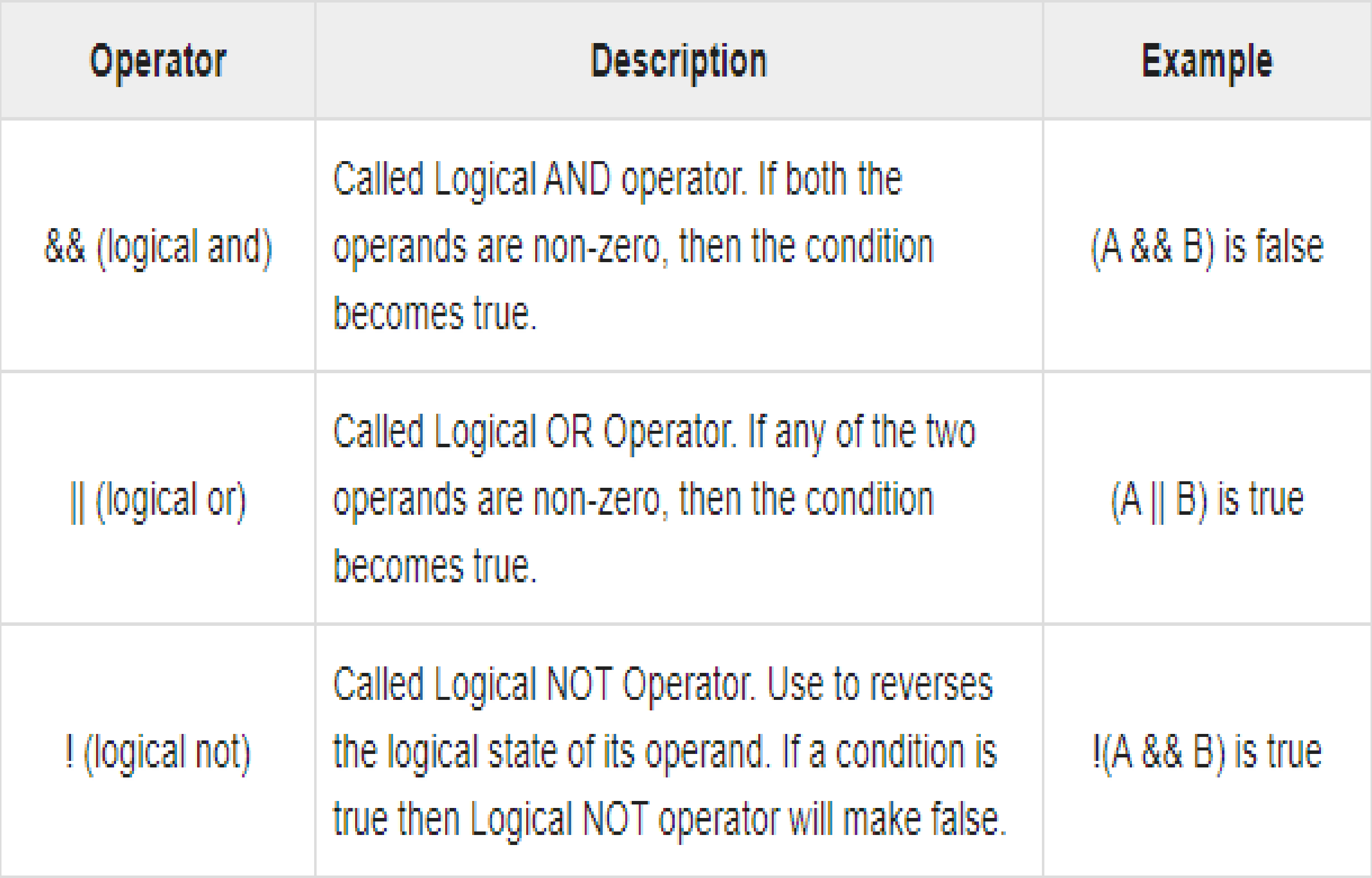
Assume integer variable A holds 60 and variable B holds 13 then −



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

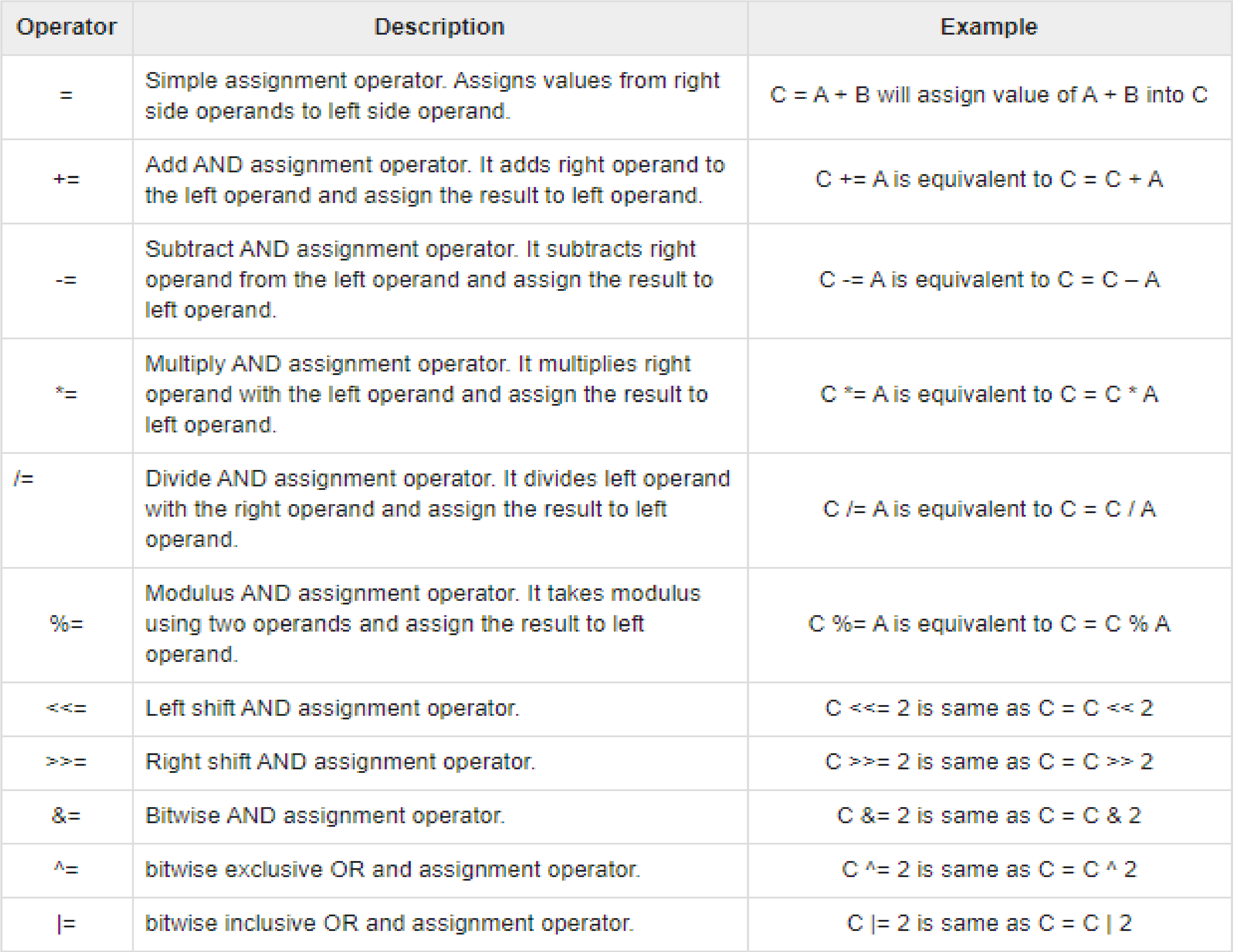
THE LOGICAL OPERATORS

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

THE ASSIGNMENT OPERATORS



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
<<=	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

EXPLICIT TYPE CAST

- When one real number is divided by another, the result is a real number; for example:

```
double x = 5.2, y = 2.0, z;
z = x / y;          // result is 2.6
```

- When dividing integers, we get an integer result

For example:

```
int x = 4, y = 9, z;
z = x / 2;          // result is 2
z = y / x;          // result is 2, again
z = x / y;          // result is 0
```

EXPLICIT TYPE CASTS - EXAMPLES

```
int x = 2, y = 5;
double z;

z = (double) y / z;            // z = 2.5
z = (double) (y / z);          // z = 2.0
```

NO DEMOTIONS IN ASSIGNMENT CONVERSIONS

- In Java we are not allowed to “demote” a higher-precision type value by assigning it to a lower-precision type variable
- Instead, we must do an explicit type cast. Some examples:

```
int x = 10;
double y = x;  // this is allowed; y = 10.0
x = y;         // error: can’t demote value to int
y = y / 3;                // y now contains 3.3333333333333333
x = (int)y;               // allowed; x = 3
```

VARIABLE

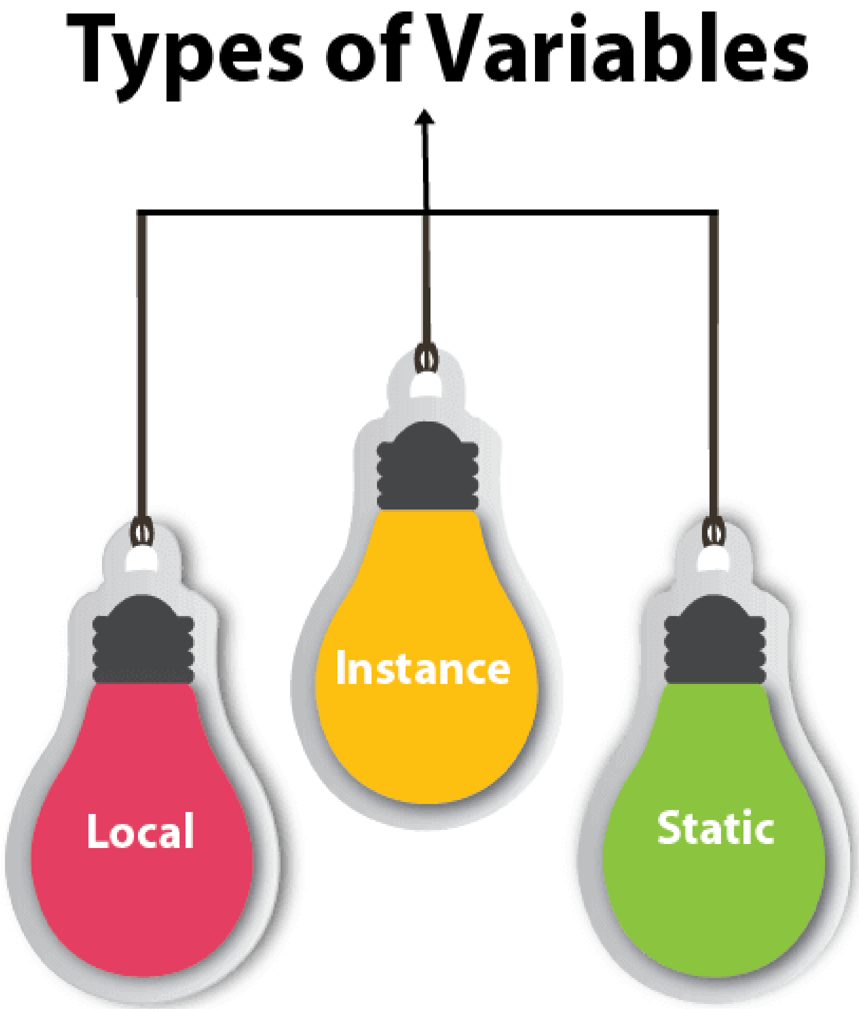
Variable is name of reserved area allocated in memory. In other words, it is a name of memory location. It is a combination of "vary + able" that means its value can be changed.

```
int data=50;//Here data is variable
```

Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable



LOCAL VARIABLES

✓Local variables are declared in methods, constructors, or blocks.

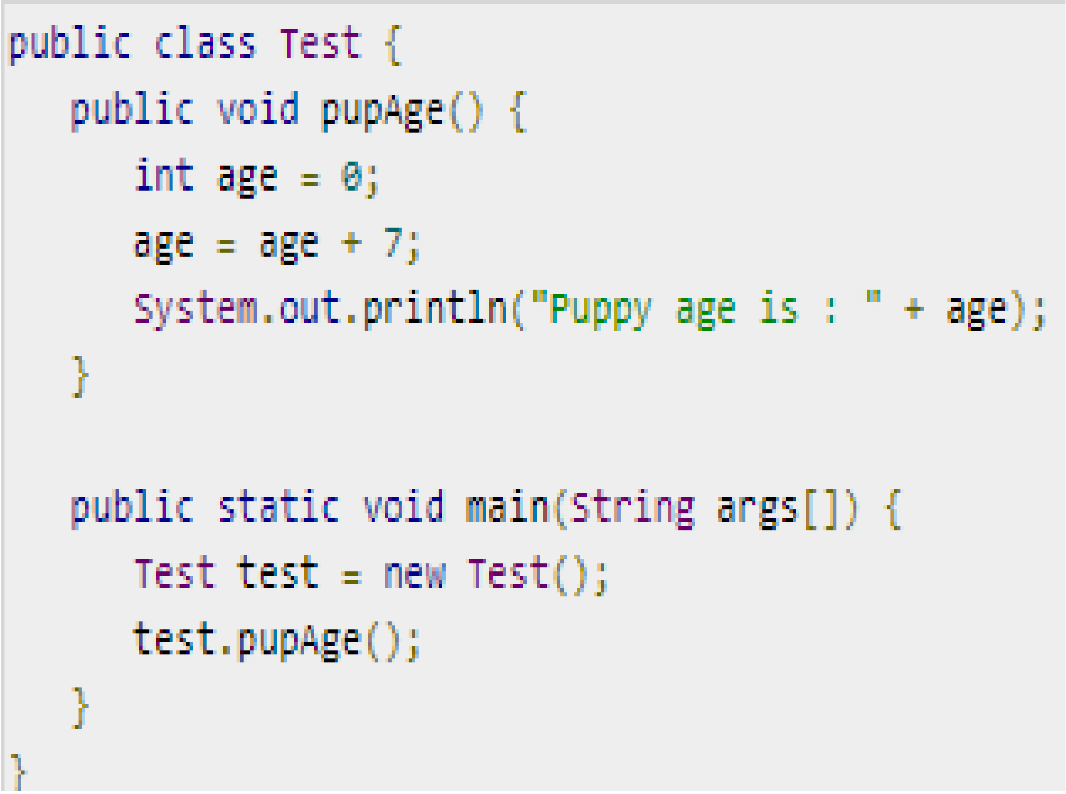
✓Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.

✓Access modifiers cannot be used for local variables.

✓Local variables are visible only within the declared method, constructor, or block.

✓Local variables are implemented at stack level internally.

✓There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.



```
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();
   }
}
```

INSTANCE VARIABLES

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

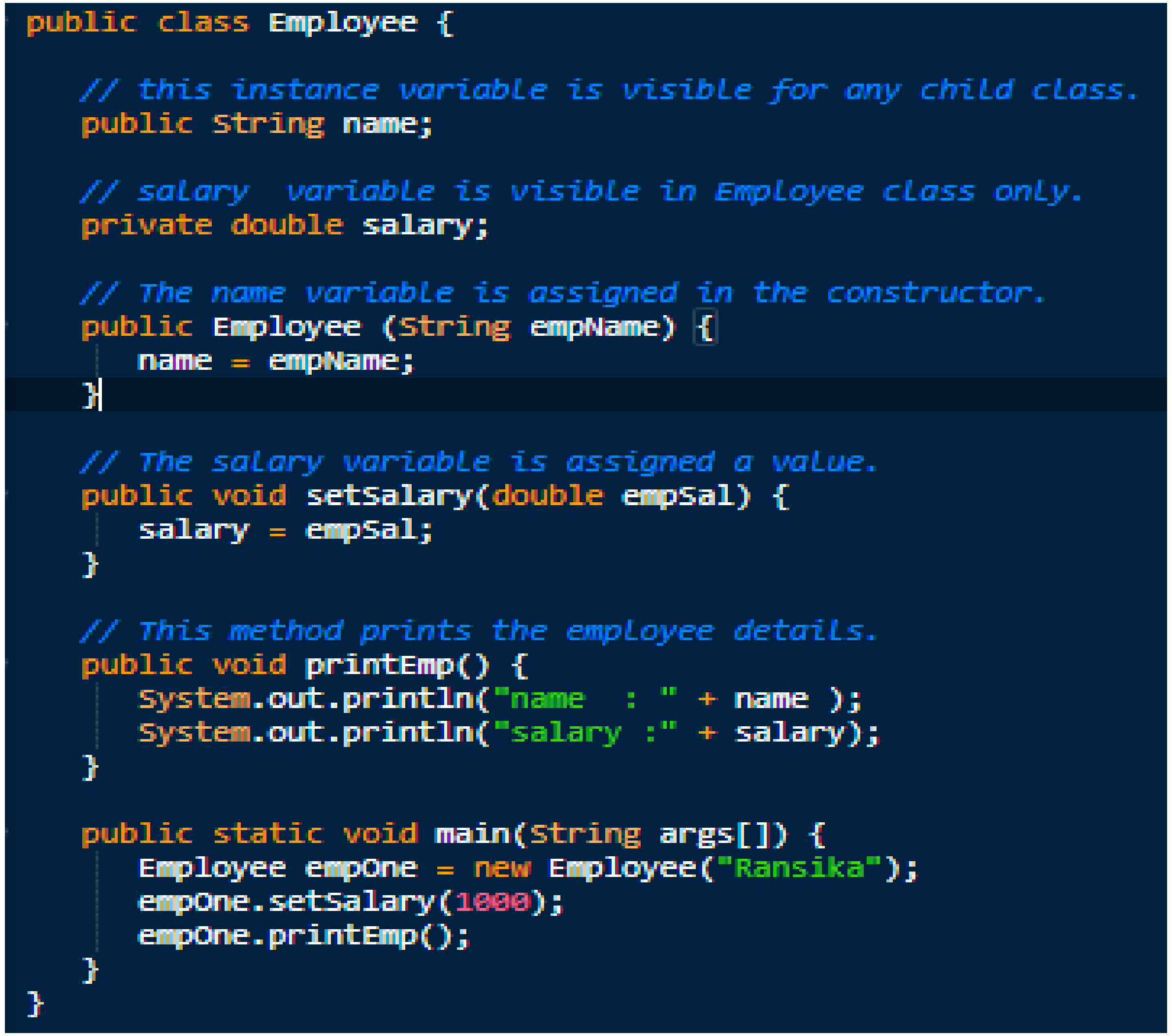
✓When a space is allocated for an object in the heap, a slot for each instance variable value is created.

✓Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

✓Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.

✓Instance variables can be declared in class level before or after use.

✓Access modifiers can be given



```java
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.
    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();
    }
}
```

CLASS/STATIC VARIABLES

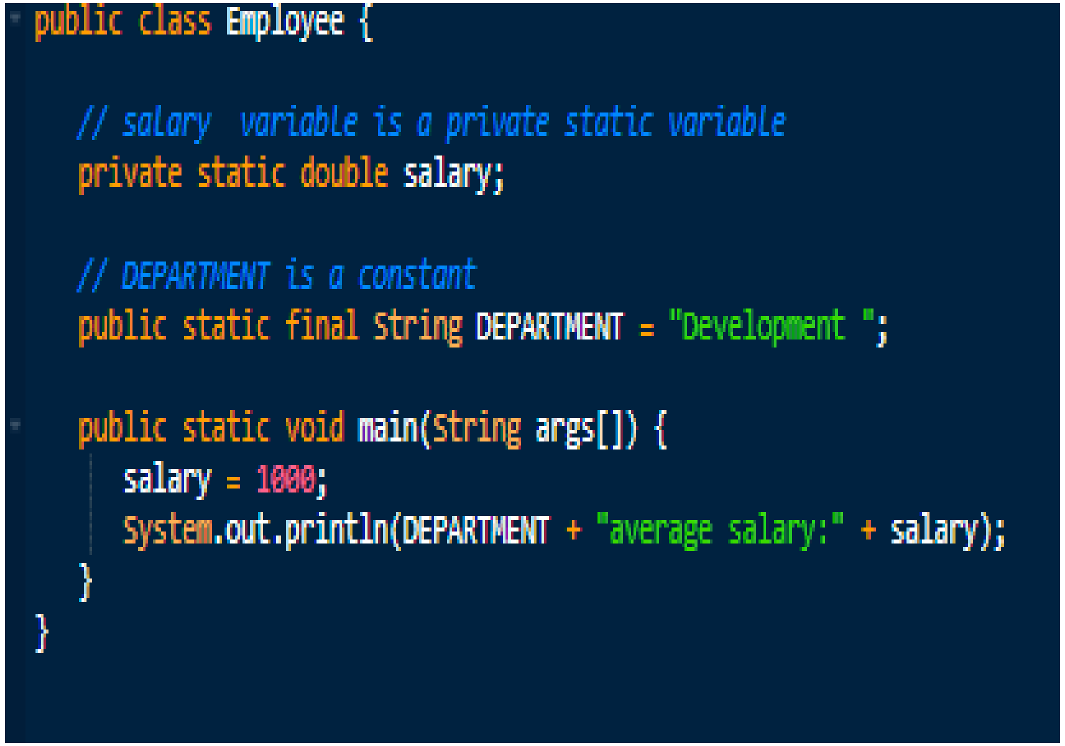
✓Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.

✓There would only be one copy of each class variable per class, regardless of how many objects are created from it.

✓Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.

✓Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.

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



```java
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);
   }
}
```