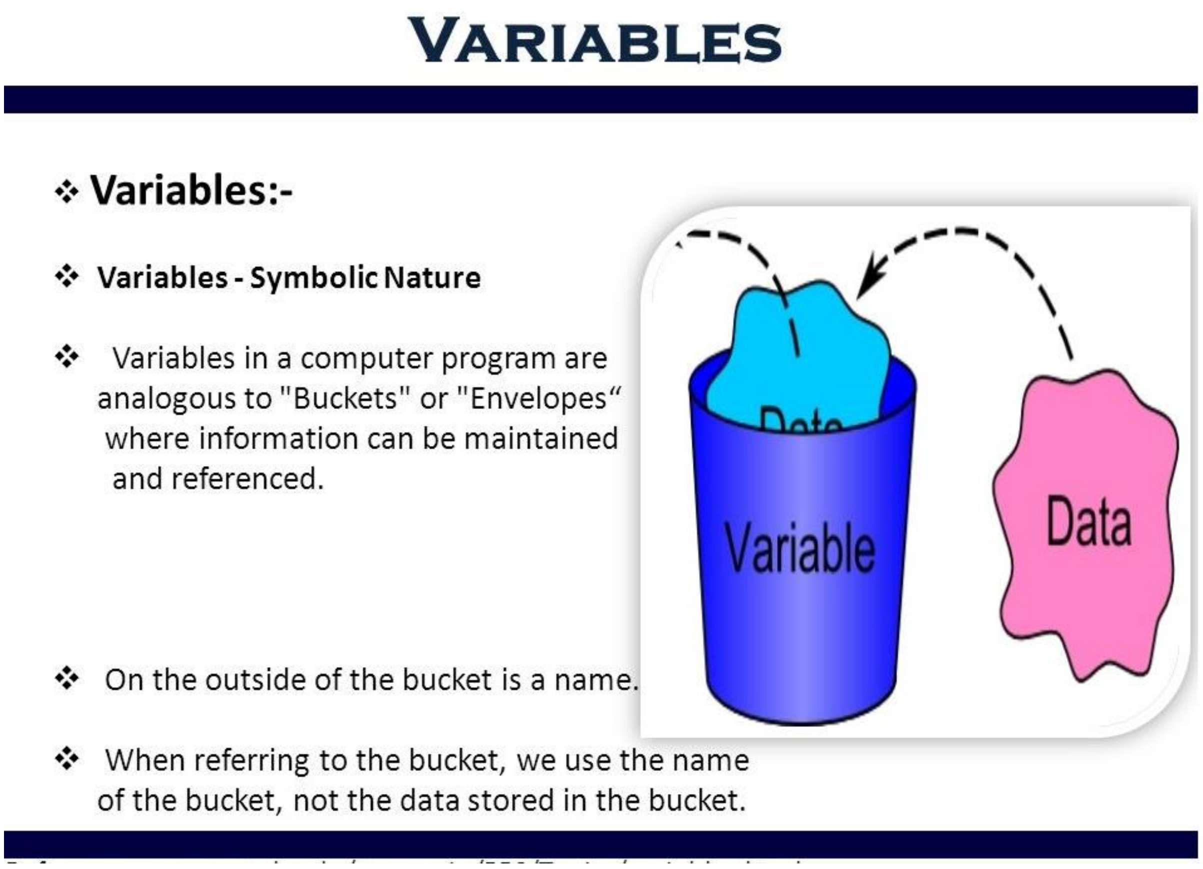
Variable, Data Types and Operators

❖ What is variable?



VARIABLES

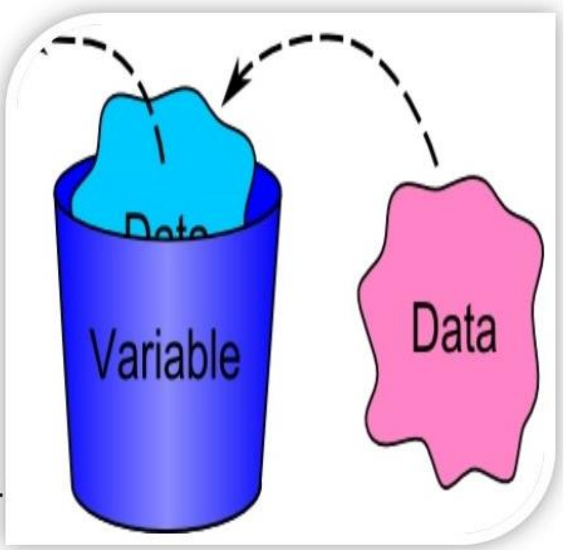
❖ **Variables:-**

❖ **Variables - Symbolic Nature**

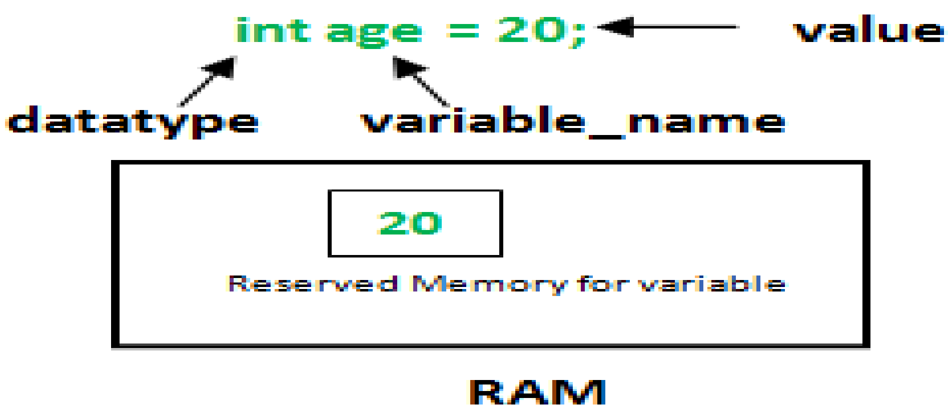
❖ Variables in a computer program are analogous to "Buckets" or "Envelopes" where information can be maintained and referenced.

❖ On the outside of the bucket is a name.

❖ When referring to the bucket, we use the name of the bucket, not the data stored in the bucket.

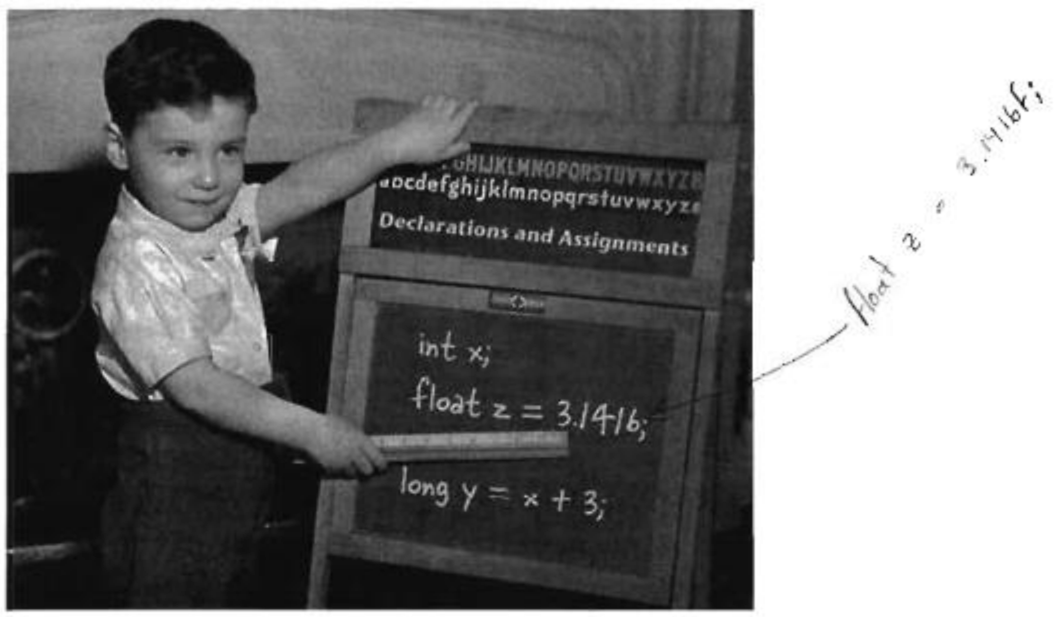


❖ How to initialize variable?

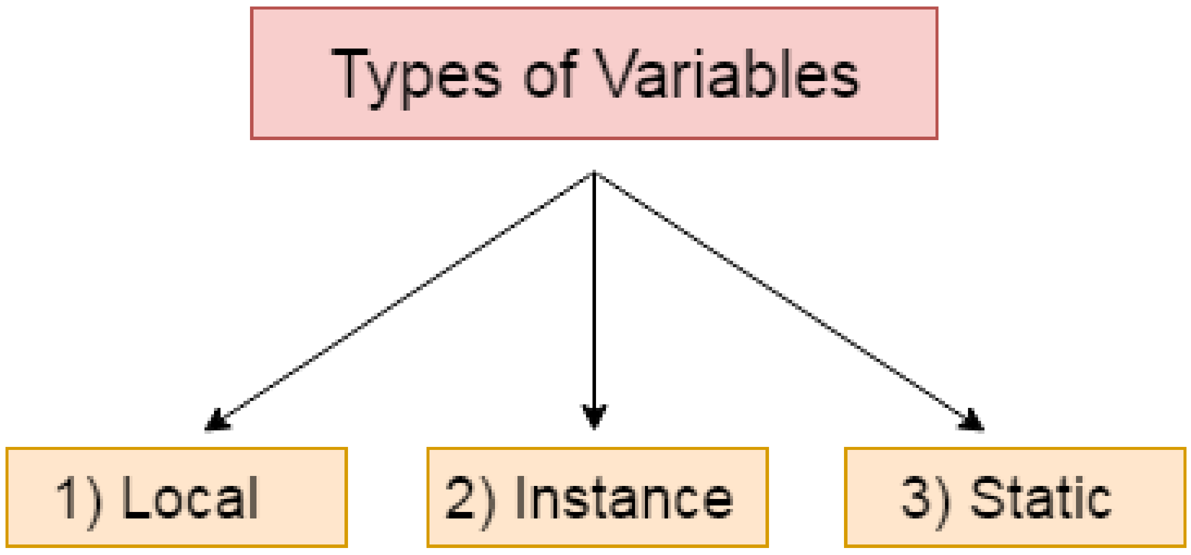


```
int age = 20;
```

Know Your Variables



❖ What are the types of variables?

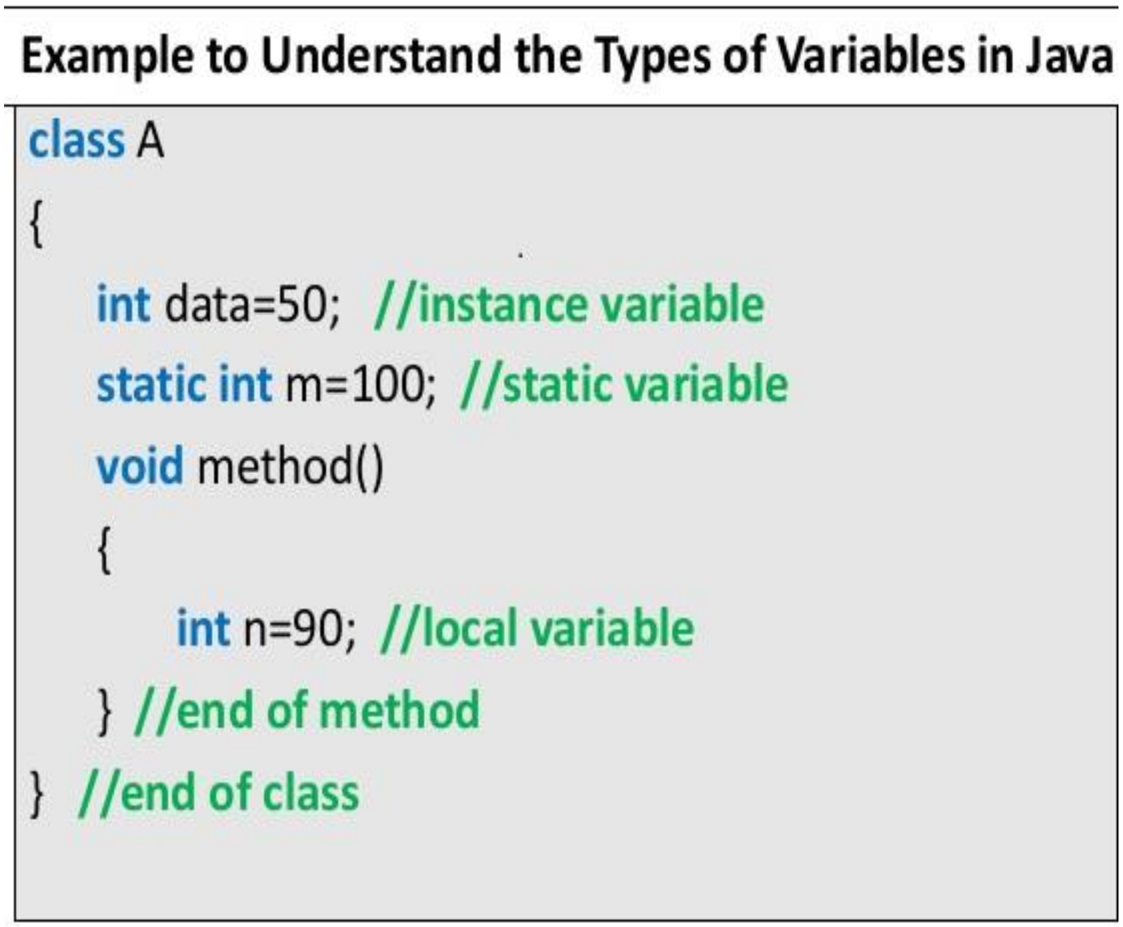


Static variables are shared.

All instances of the same class share a single copy of the static variables.

instance variables: 1 per **instance**

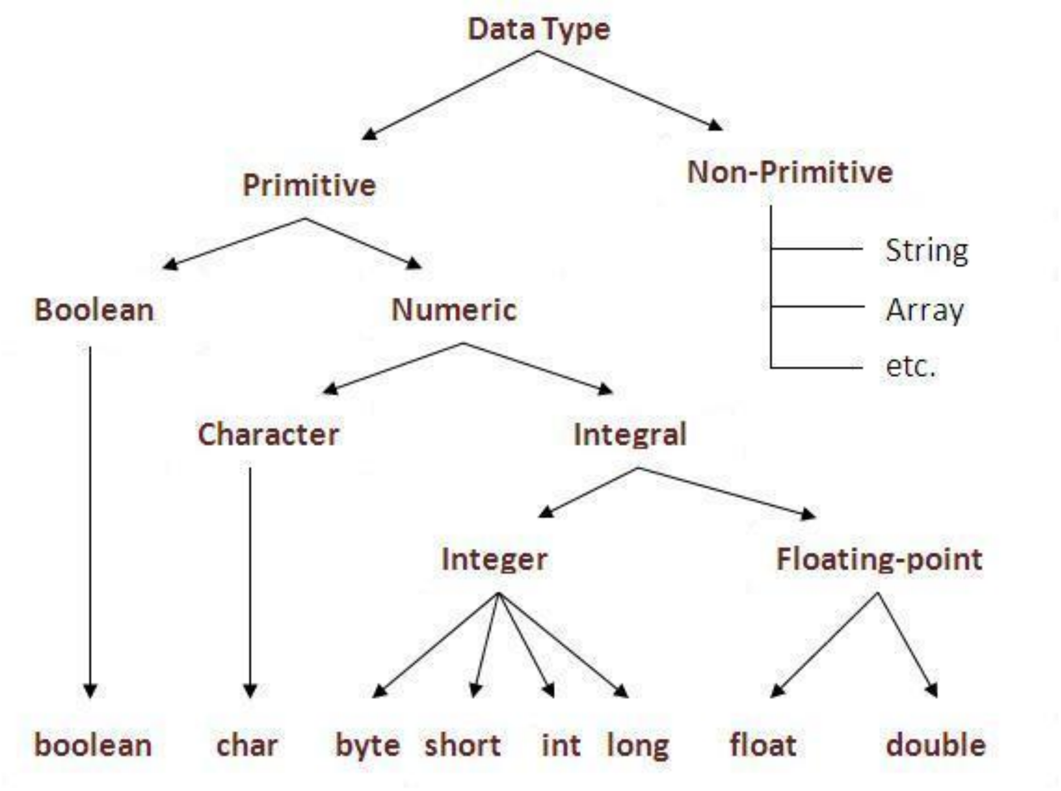
static variables: 1 per **class**



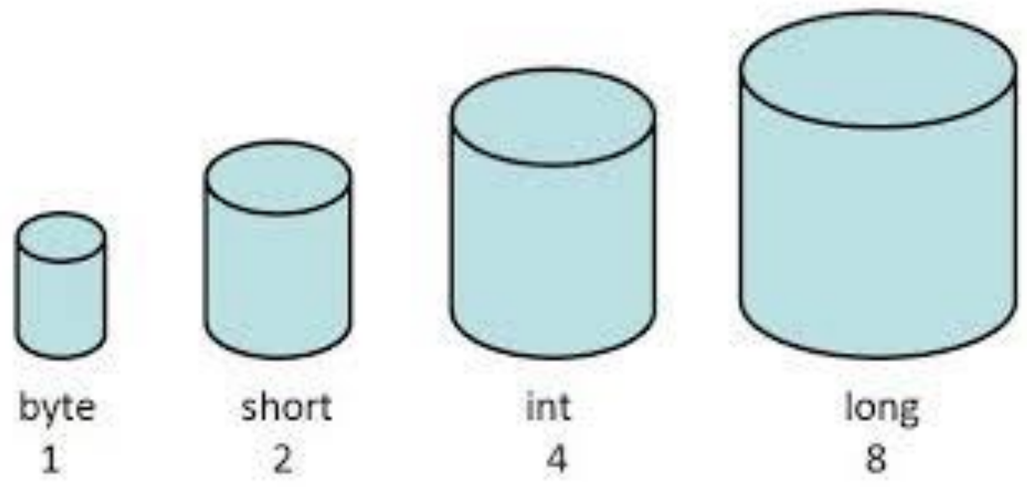
Example to Understand the Types of Variables in Java

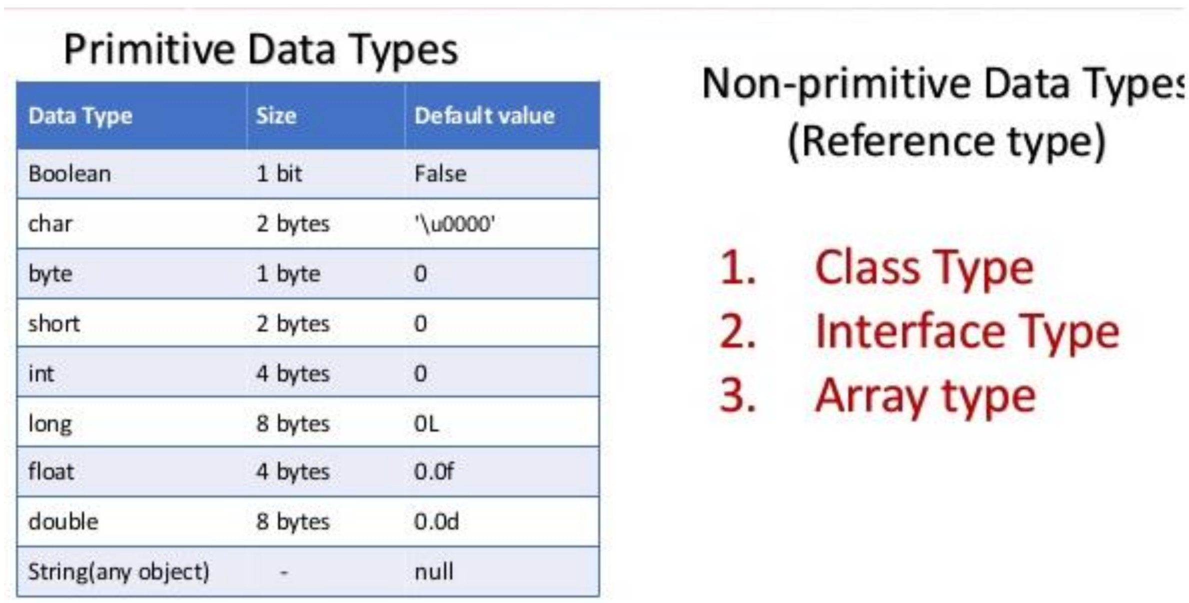
```
class A
{
     int data=50;   //instance variable
     static int m=100;  //static variable
     void method()
     {
          int n=90;  //local variable
     } //end of method
}  //end of class
```

❖ Data Types in Java



❖ Primitive Data Types



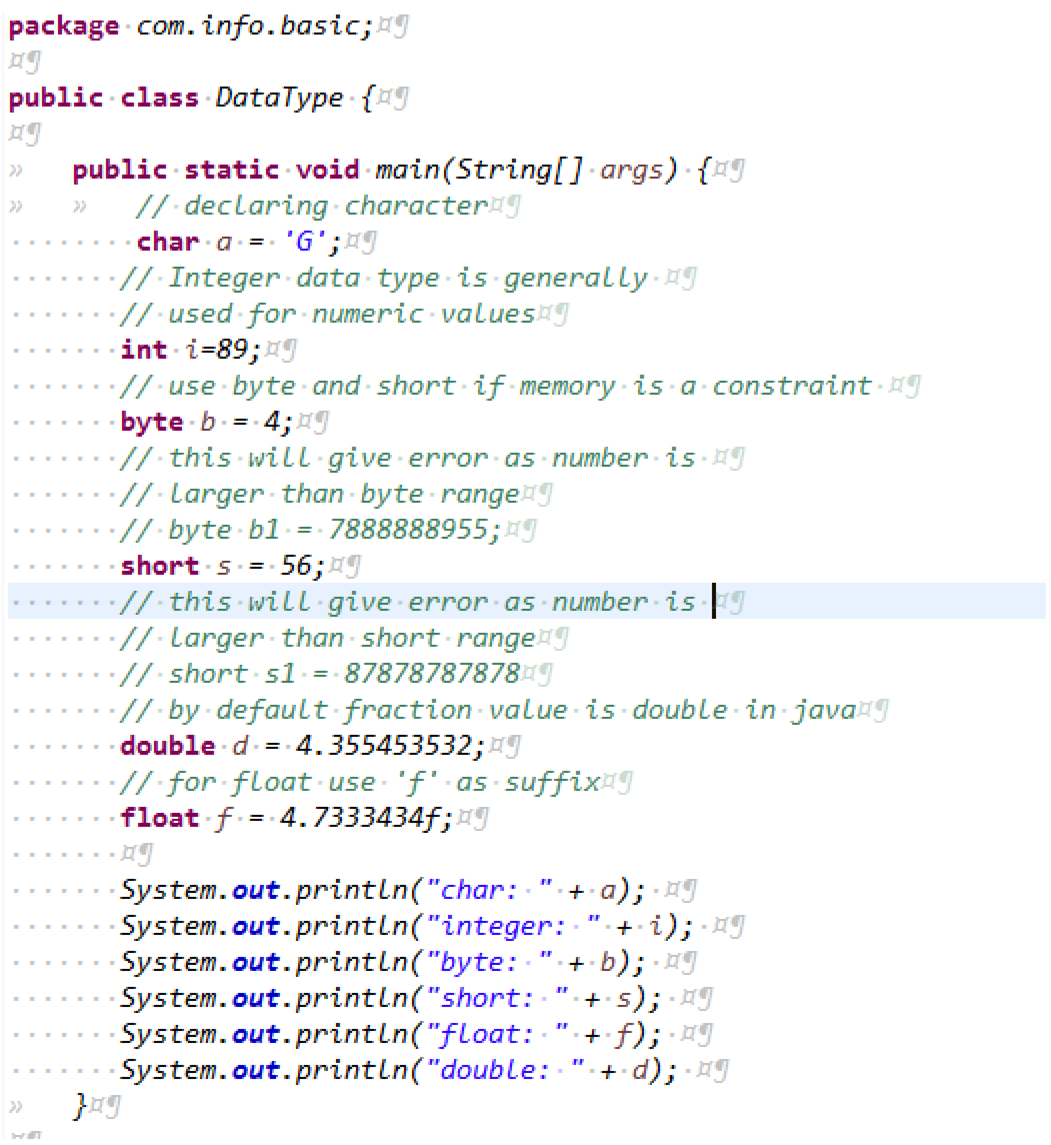


Primitive Data Types

Data Type	Size	Default value
Boolean	1 bit	False
char	2 bytes	'\u0000'
byte	1 byte	0
short	2 bytes	0
int	4 bytes	0
long	8 bytes	0L
float	4 bytes	0.0f
double	8 bytes	0.0d
String(any object)	-	null

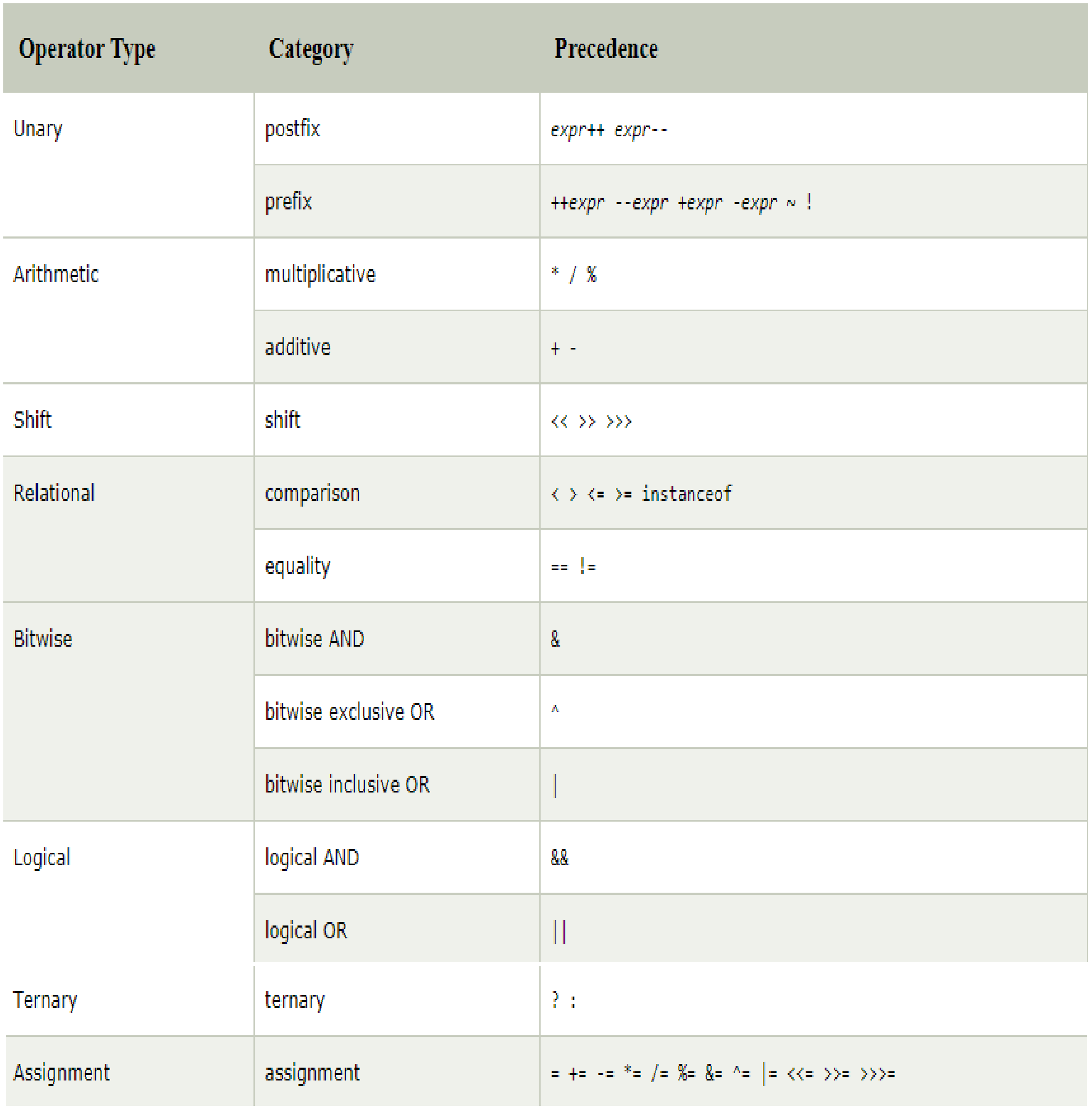
Non-primitive Data Types (Reference type)

1. Class Type
2. Interface Type
3. Array type



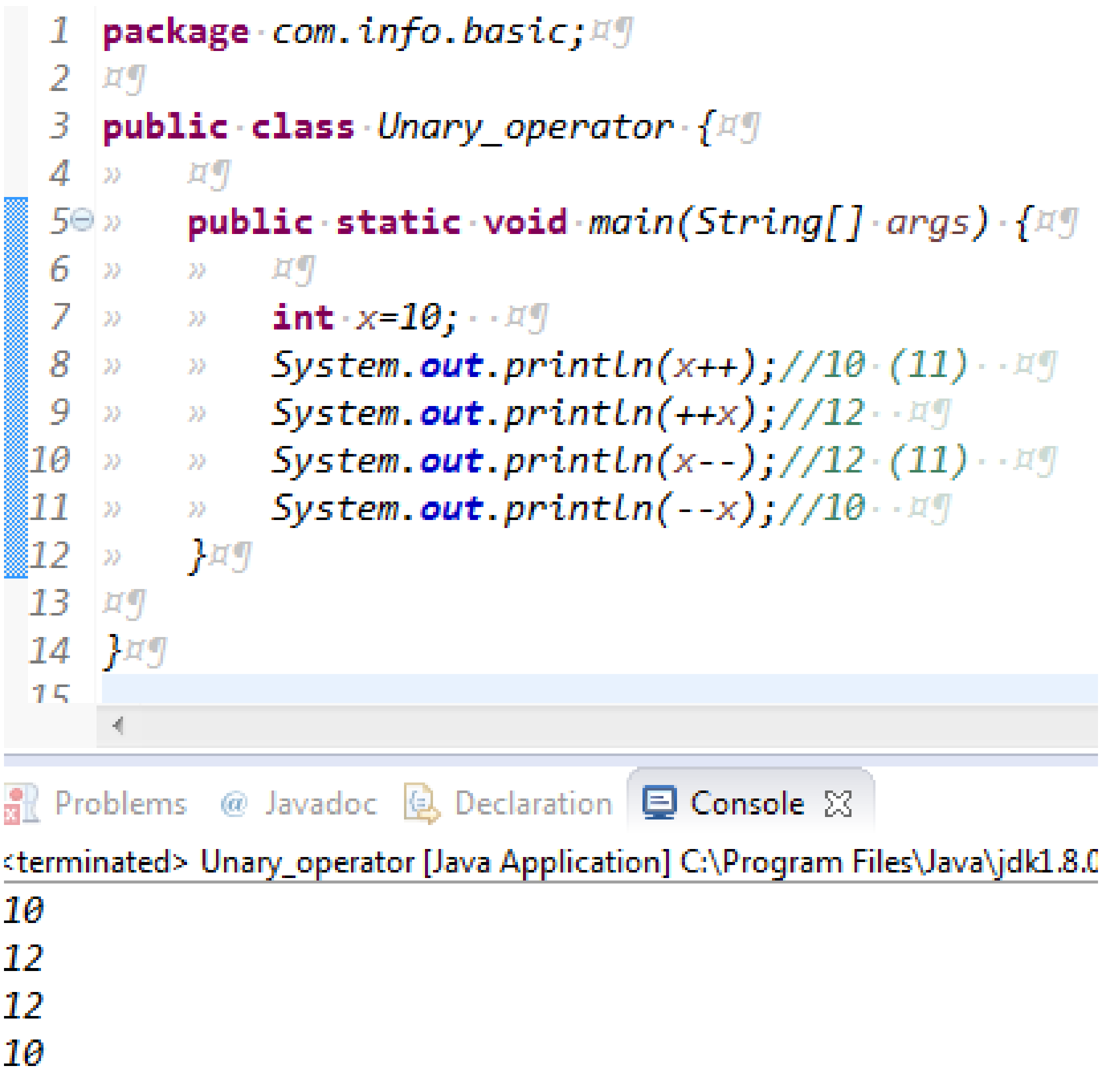
```java
package com.info.basic;

public class DataType {

    public static void main(String[] args) {
        // declaring character
        char a = 'G';
        // Integer data type is generally 
        // used for numeric values
        int i=89;
        // use byte and short if memory is a constraint 
        byte b = 4;
        // this will give error as number is 
        // larger than byte range
        // byte b1 = 7888888955;
        short s = 56;
        // this will give error as number is 
        // larger than short range
        // short s1 = 87878787878
        // by default fraction value is double in java
        double d = 4.355453532;
        // for float use 'f' as suffix
        float f = 4.7333434f;

        System.out.println("char: " + a); 
        System.out.println("integer: " + i); 
        System.out.println("byte: " + b); 
        System.out.println("short: " + s); 
        System.out.println("float: " + f); 
        System.out.println("double: " + d); 
    }

```

Operators in Java



Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

❖Unary Operator:-



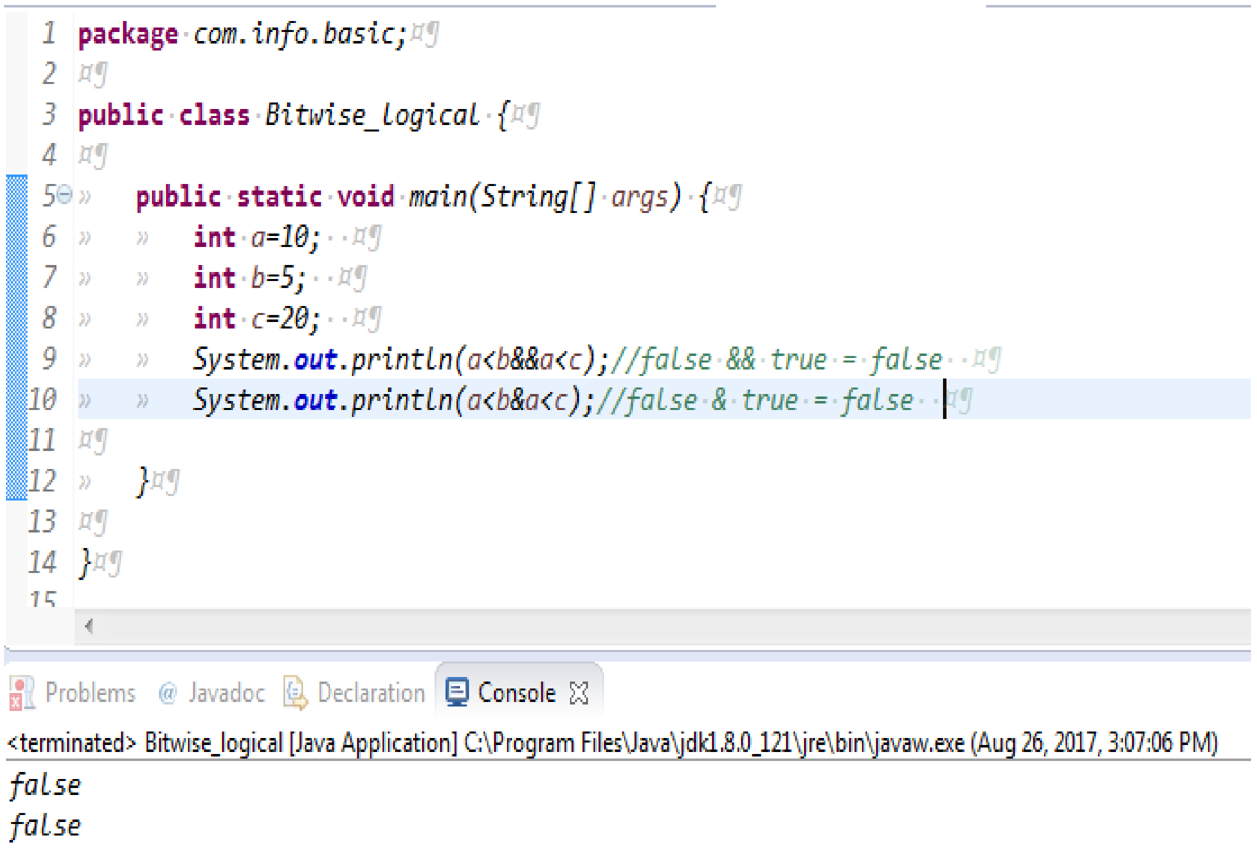
```java
package com.info.basic;

public class Unary_operator {
	
	public static void main(String[] args) {
		
		int x=10;  
		System.out.println(x++);//10 (11)  
		System.out.println(++x);//12  
		System.out.println(x--);//12 (11)  
		System.out.println(--x);//10  
	}

}
```

Problems @ Javadoc Declaration Console

```
<terminated> Unary_operator [Java Application] C:\Program Files\Java\jdk1.8.0
10
12
12
10
```

❖ Java AND Operator Example: Logical && and Bitwise &

- **&&, Logical AND:** returns true when both conditions are true.
- **||, Logical OR:** returns true if at least one condition is true.



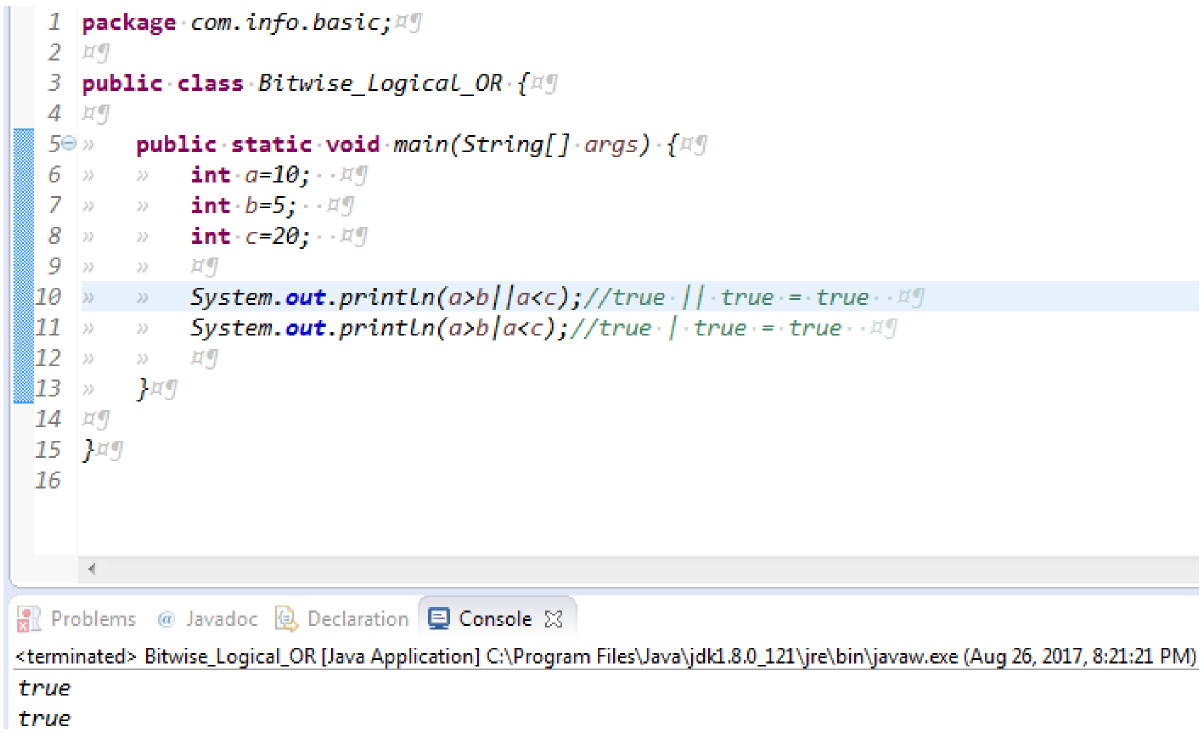
```
package com.info.basic;

public class Bitwise_logical {

    public static void main(String[] args) {
        int a=10;
        int b=5;
        int c=20;
        System.out.println(a<b&&a<c);//false && true = false
        System.out.println(a<b&a<c);//false & true = false

    }

}
```

Problems @ Javadoc Declaration Console

```
<terminated> Bitwise_logical [Java Application] C:\Program Files\Java\jdk1.8.0_121\jre\bin\javaw.exe (Aug 26, 2017, 3:07:06 PM)
false
false
```

❖ Java OR Operator Example: Logical || and Bitwise |

- The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.
- The bitwise | operator always checks both conditions whether first condition is true or false.



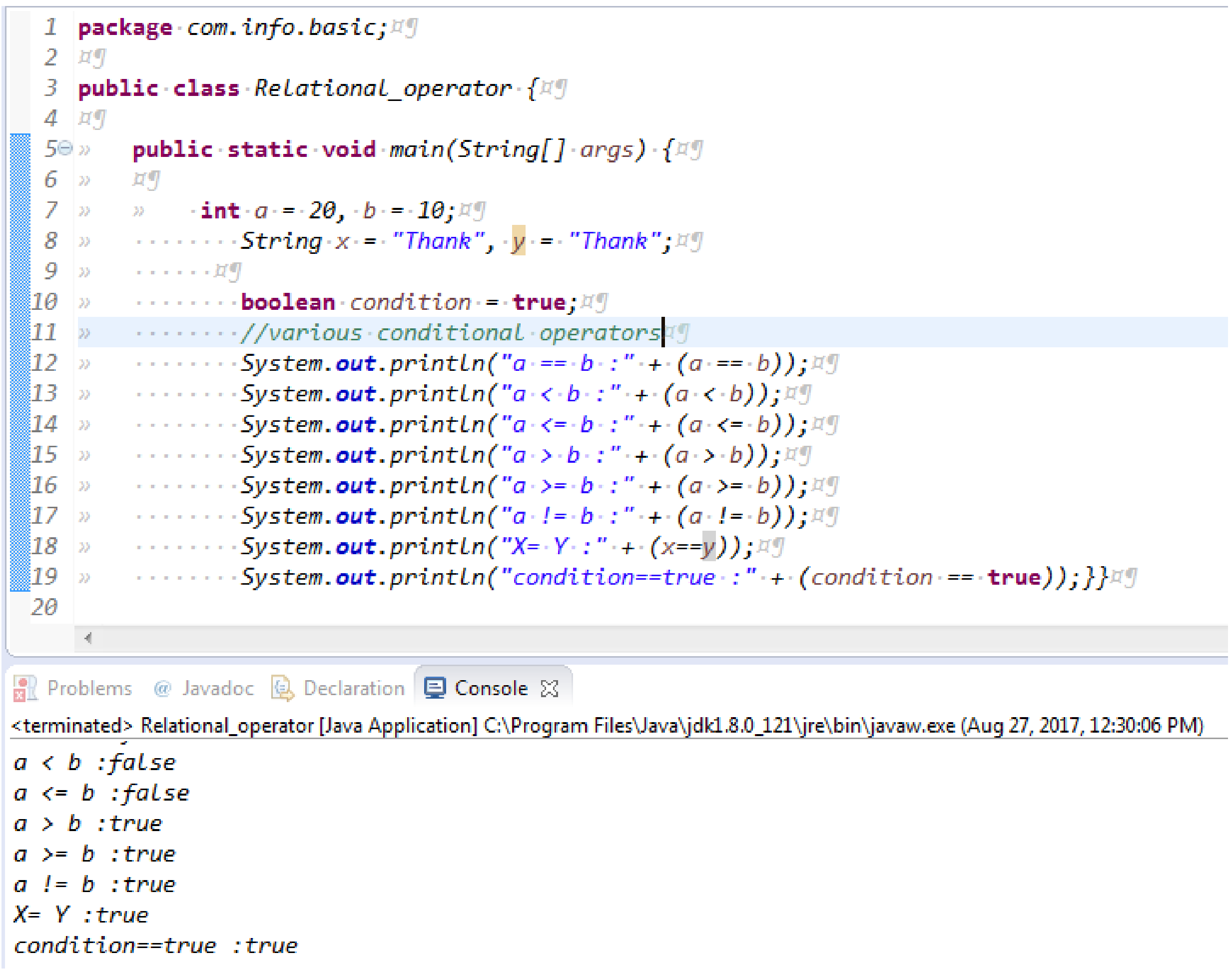
```java
package com.info.basic;

public class Bitwise_Logical_OR {

	public static void main(String[] args) {
		int a=10;
		int b=5;
		int c=20;

		System.out.println(a>b||a<c);//true || true = true
		System.out.println(a>b|a<c);//true | true = true

	}

}
```

```
<terminated> Bitwise_Logical_OR [Java Application] C:\Program Files\Java\jdk1.8.0_121\jre\bin\javaw.exe (Aug 26, 2017, 8:21:21 PM)
true
true
```

❖ Relational Operators:-

➢ These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison and are extensively used in looping statements as well as conditional if else statements. General format is,

Syntax:-variable **relation_operator** value



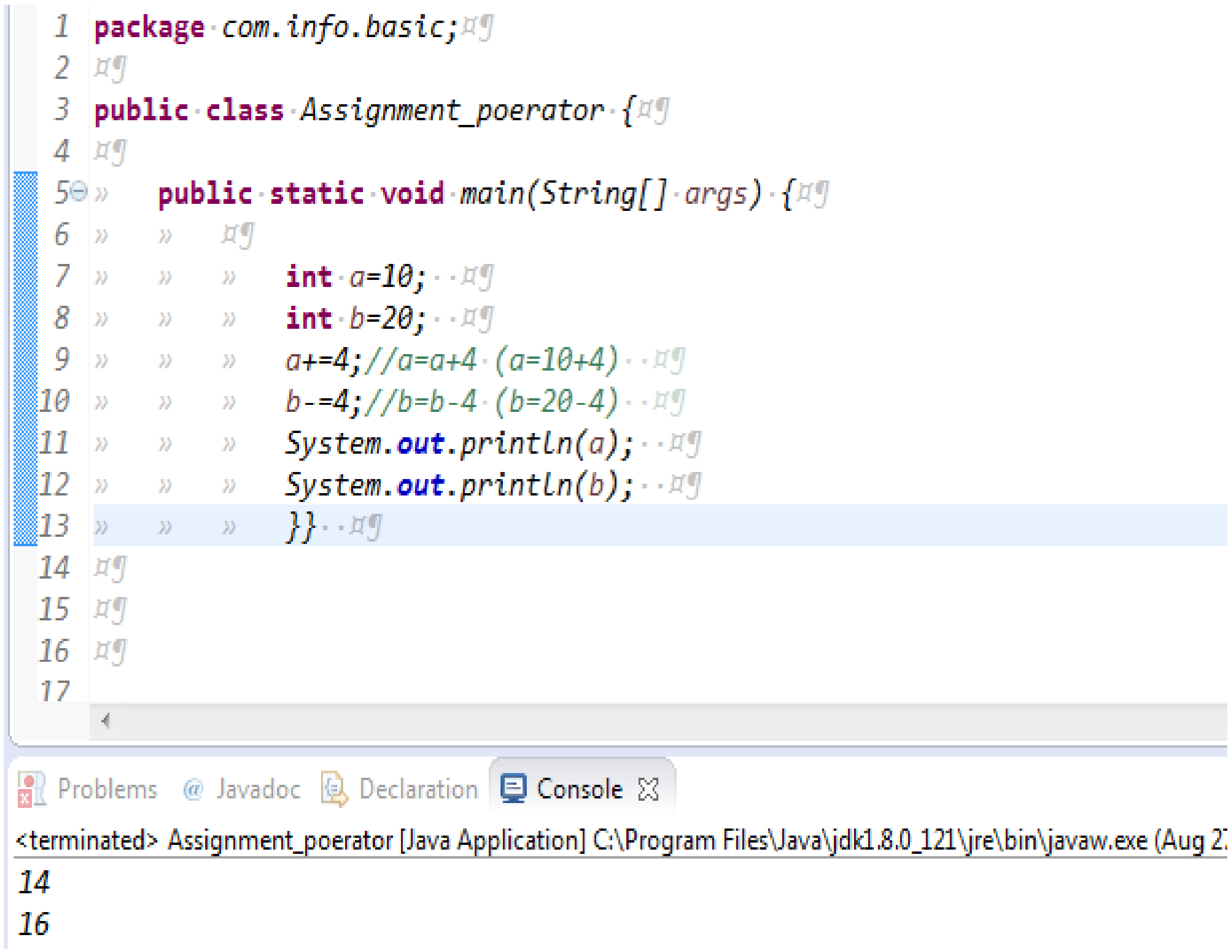
```
package com.info.basic;

public class Relational_operator {

    public static void main(String[] args) {

        int a = 20, b = 10;
        String x = "Thank", y = "Thank";

        boolean condition = true;
        //various conditional operators
        System.out.println("a == b :" + (a == b));
        System.out.println("a < b :" + (a < b));
        System.out.println("a <= b :" + (a <= b));
        System.out.println("a > b :" + (a > b));
        System.out.println("a >= b :" + (a >= b));
        System.out.println("a != b :" + (a != b));
        System.out.println("X= Y :" + (x==y));
        System.out.println("condition==true :" + (condition == true));}}
```

Problems Javadoc Declaration Console

```
<terminated> Relational_operator [Java Application] C:\Program Files\Java\jdk1.8.0_121\jre\bin\javaw.exe (Aug 27, 2017, 12:30:06 PM)
a < b :false
a <= b :false
a > b :true
a >= b :true
a != b :true
X= Y :true
condition==true :true
```

❖ Assignment Operator :

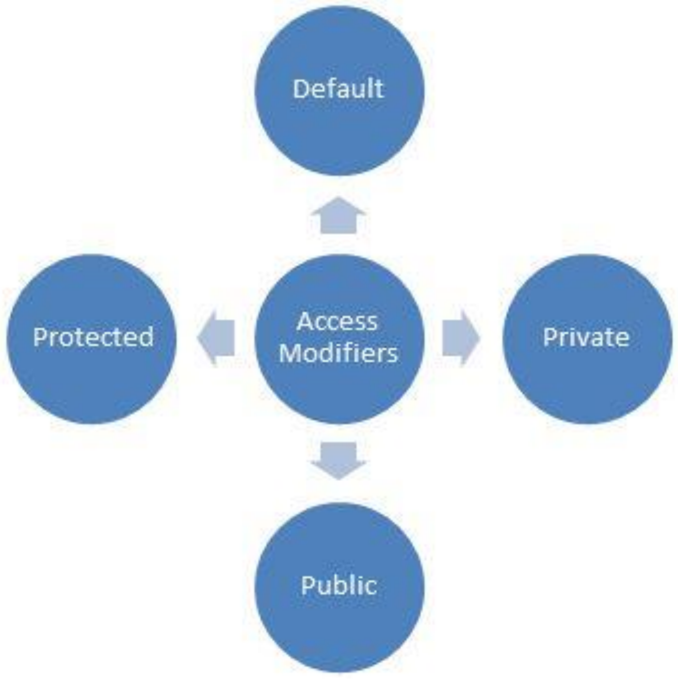
'=' Assignment operator is used to assign a value to any variable. It has a right to left associativity,



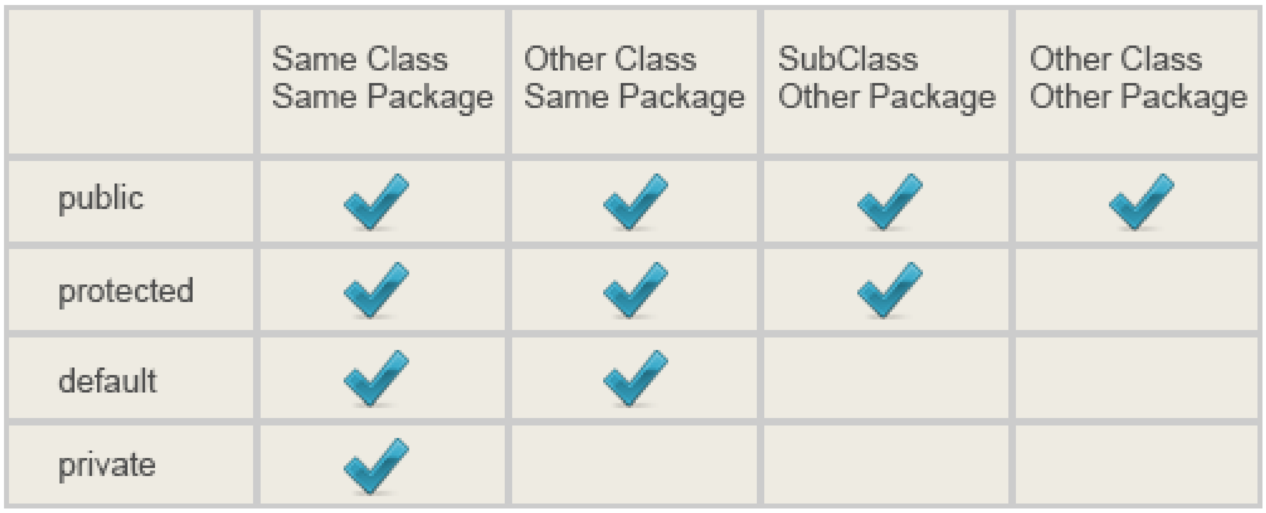
```
package com.info.basic;

public class Assignment_poerator {

	public static void main(String[] args) {
		
			int a=10;  
			int b=20;  
			a+=4;//a=a+4 (a=10+4)  
			b-=4;//b=b-4 (b=20-4)  
			System.out.println(a);  
			System.out.println(b);  
			}}  



```

```
<terminated> Assignment_poerator [Java Application] C:\Program Files\Java\jdk1.8.0_121\jre\bin\javaw.exe (Aug 2
14
16
```

Access Modifiers in Java



❖ Modifiers Scope



	Same Class Same Package	Other Class Same Package	SubClass Other Package	Other Class Other Package
public	✓	✓	✓	✓
protected	✓	✓	✓	
default	✓	✓		
private	✓			

➢ The access modifier in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

- There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

1) Private access modifier

The private access modifier is accessible only within class.

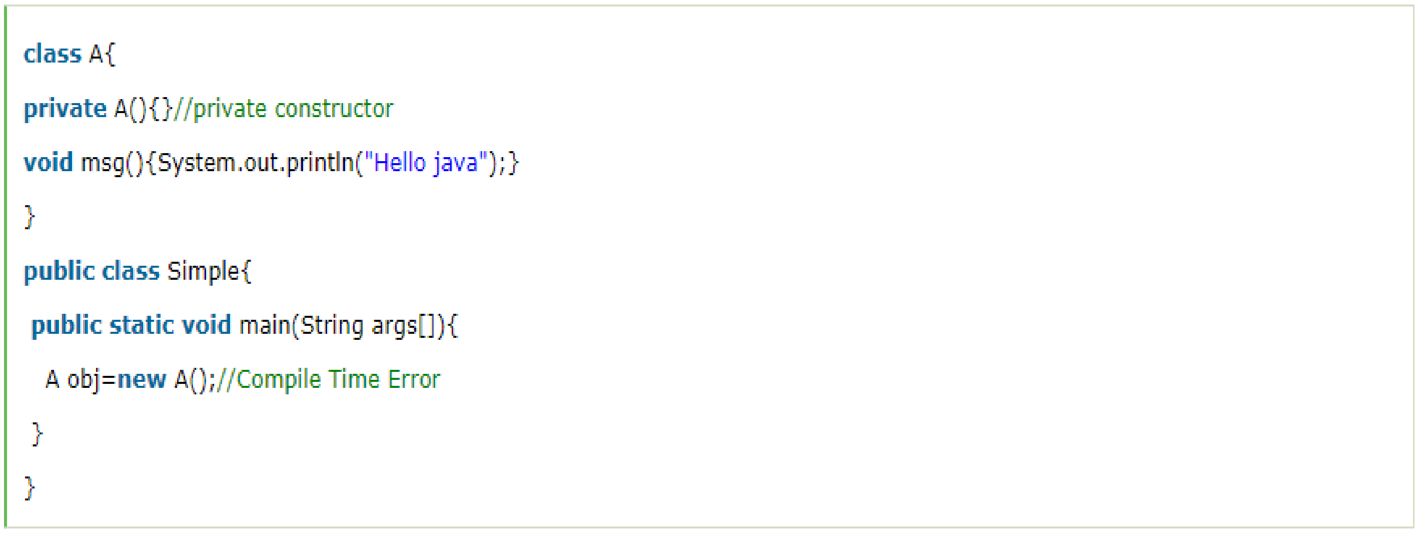
Simple example of private access modifier

- In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Role of Private Constructor

- If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:



```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

2) default access modifier

- If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package.

Example of default access modifier

- In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

3) Protected access modifier

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

- In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

```
Output:Hello
```

4) public access modifier

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

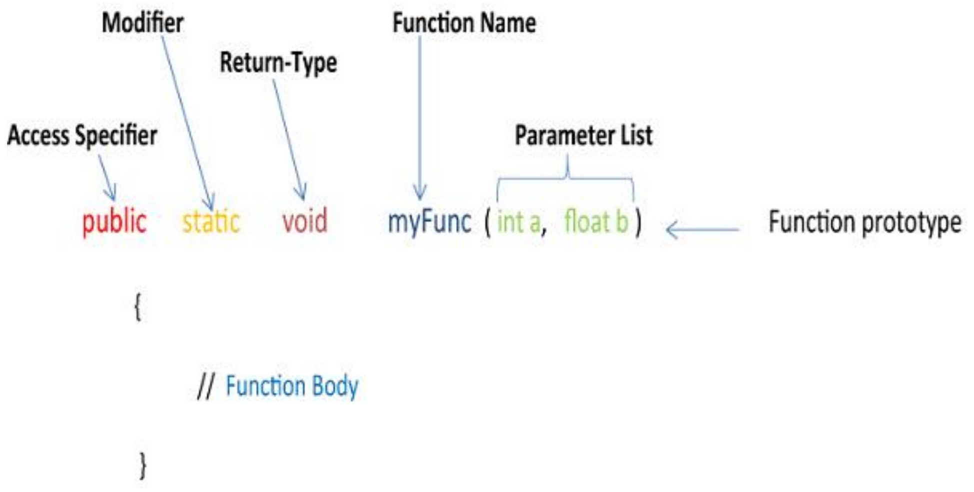
Example of public access modifier

```
//save by A.java

package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

```
Output:Hello
```

Function/Methods in Java

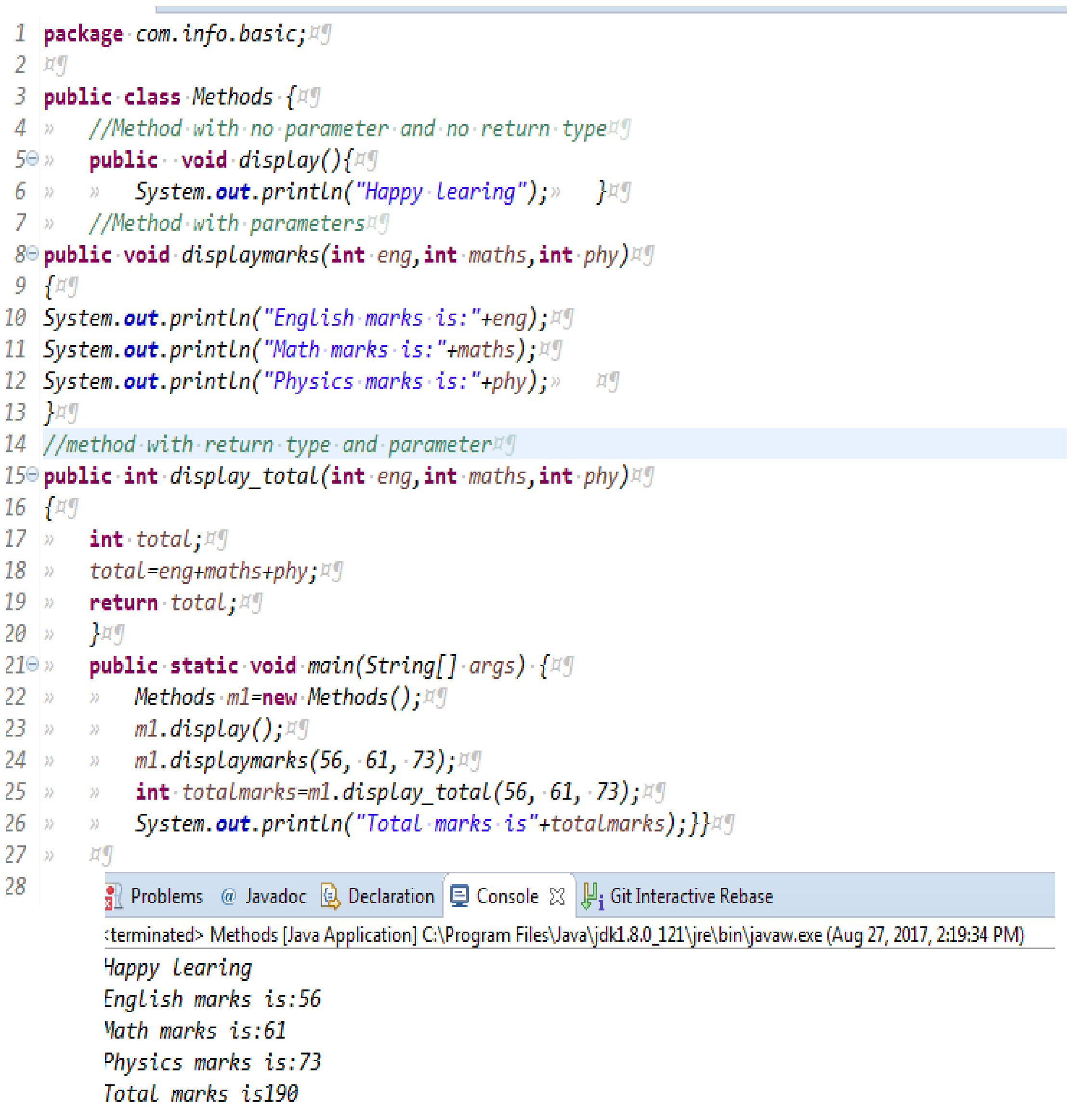


❖ What is method in Java?

- Methods are collections of statements for performing actions.
- Method consists of name, return type, access modifier and parameters.
- Method can return value Or can return nothing in which case type is void

❖ Way to write method in Java

- Method with no return type no parameter
- Method with parameters
- Method with return type and parameters.



```java
package com.info.basic;

public class Methods {
	//Method with no parameter and no return type
	public  void display(){
		System.out.println("Happy Learing");	}
	//Method with parameters
public void displaymarks(int eng,int maths,int phy)
{
System.out.println("English marks is:"+eng);
System.out.println("Math marks is:"+maths);
System.out.println("Physics marks is:"+phy);	
}
//method with return type and parameter
public int display_total(int eng,int maths,int phy)
{
	int total;
	total=eng+maths+phy;
	return total;
	}
	public static void main(String[] args) {
		Methods m1=new Methods();
		m1.display();
		m1.displaymarks(56, 61, 73);
		int totalmarks=m1.display_total(56, 61, 73);
		System.out.println("Total marks is"+totalmarks);}}
	
```

```
<terminated> Methods [Java Application] C:\Program Files\Java\jdk1.8.0_121\jre\bin\javaw.exe (Aug 27, 2017, 2:19:34 PM)
Happy Learing
English marks is:56
Math marks is:61
Physics marks is:73
Total marks is190
```

Static methods vs Instance methods in Java

Instance Method

Instance method are methods which require an object of its class to be created before it can be called. To invoke a instance method, we have to create an Object of the class in within which it defined.

```
public void geek(String name)
{
 // code to be executed....
}
// Return type can be int, float String or user defined data type.
```

Important Points:

- Instance method(s) belong to the Object of the class not to the class i.e. they can be called after creating the Object of the class.
- Every individual Object created from the class has its own copy of the instance method(s) of that class.
- They can be overridden since they are resolved using **dynamic binding** at run time.

Static Method

Static methods are the methods in Java that can be called without creating an object of class. They are referenced by the **class name itself** or reference to the Object of that class.

```
public static void geek(String name)
{
 // code to be executed....
}

// Must have static modifier in their declaration.
// Return type can be int, float, String or user defined data type.
```

Important Points:

- Static method(s) are associated to the class in which they reside i.e. they can be called even without creating an instance of the class i.e ClassName.methodName(args).
- They are designed with aim to be shared among all Objects created from the same class.
- Static methods can not be overridden. But can be overloaded since they are resolved using **static binding** by compiler at compile time.

Instance method vs Static method

- Instance method can access the instance methods and instance variables directly.
- Instance method can access static variables and static methods directly.
- Static methods can access the static variables and static methods directly.
- Static methods can't access instance methods and instance variables directly. They must use reference to object. And static method can't use this keyword as there is no instance for 'this' to refer to.