Introduction to keywords and variables

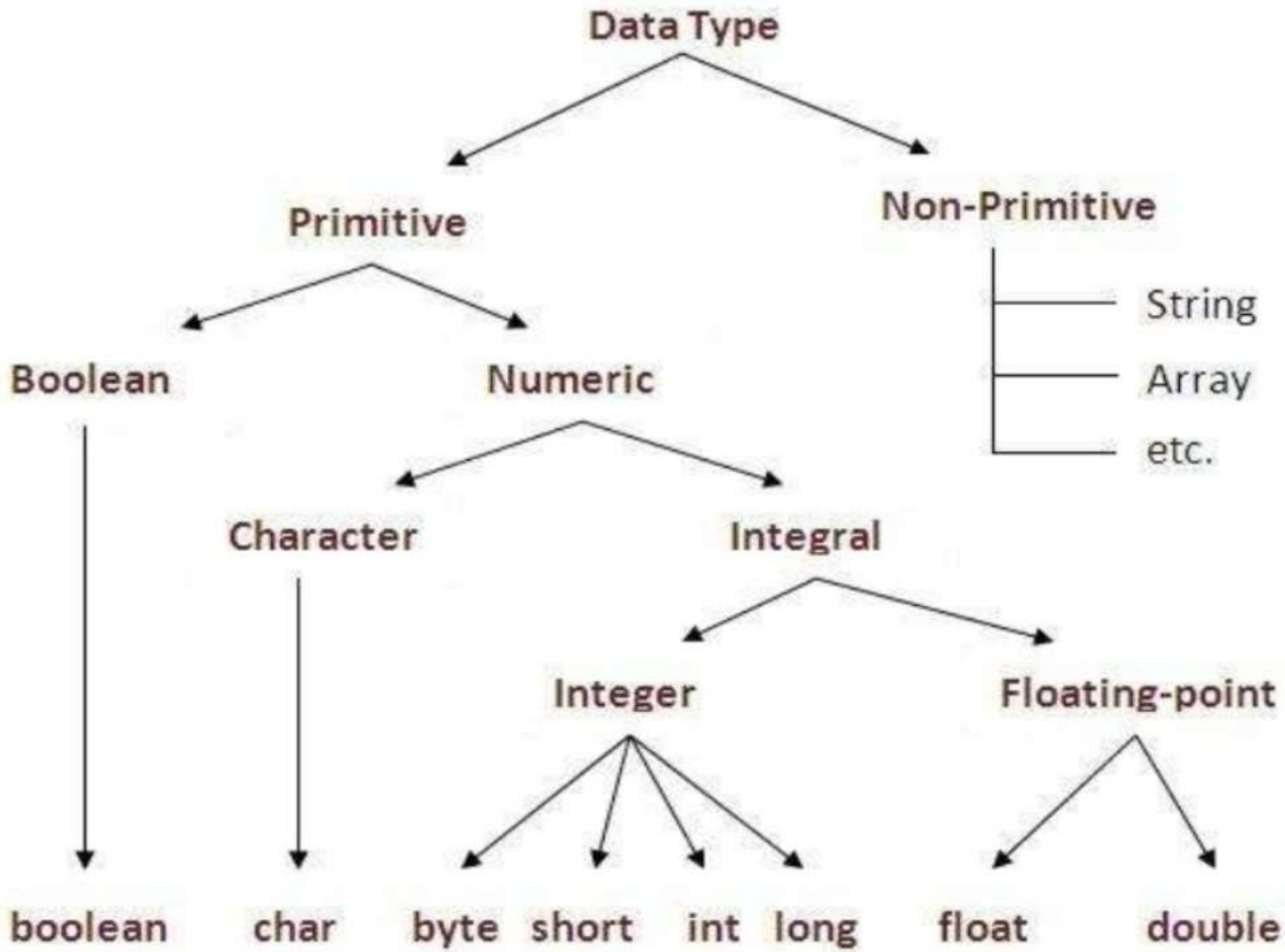
In this Lecture we Learn

- Data Types in Java
- Comments in Java
- Variable in Java
- Types of Variable in Java
- Unicode System in Java
- Type Casting in Java
- Type Casting in Java
- Java Operators
- Arithmetic Operators in Java

Data Types in Java

- The variables are the way to store data in the programming languages. In Java, you have to specify a data type of variables that tells what kind of data to be stored in it.
- Depending on the data types the operating system allocates memory and decides what type of data to be stored e.g. a number, a character etc.
- **There are two kinds of Java data types:**
- Primitive data types
- Object data types also known as the Reference data types

Data Types in Java



Primitive Data Types

There are eight primitive datatypes supported by Java. Primitive datatypes are predefined by the language and named by a keyword. Let us now look into the eight primitive data types in detail.

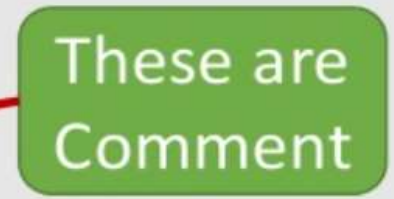
- byte – 8 bit, and number.
- short – is number type and takes two bytes.
- int – is a numeric data type and takes four bytes.
- long – is numeric and takes eight bytes.
- float – is a single precision and takes four bytes.
- double – is a double precision and takes eight bytes.
- char – can store any character and takes two bytes.
- boolean – It takes one byte and can store either of two values : *True or False*.

Primitive Data Types

- **Primitive** data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.
- **Example**
- `int a;` **// valid**
- `a=10;` **// valid**
- `a=10, 20, 30;` **// invalid**

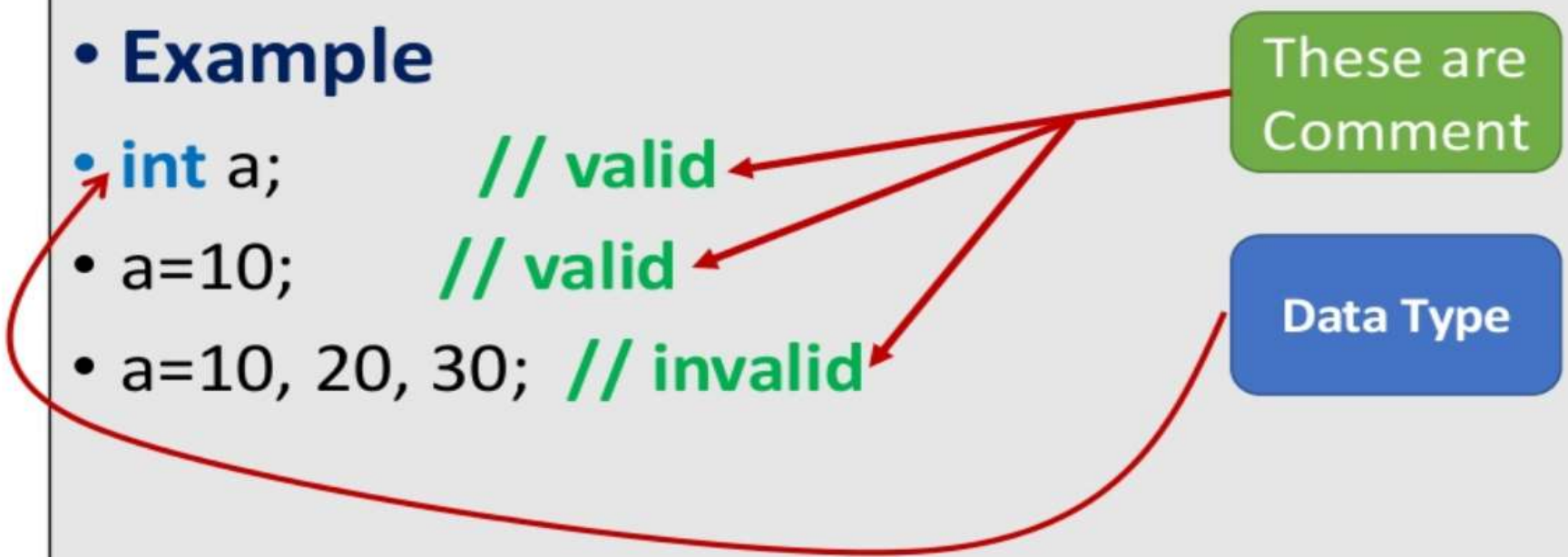
Primitive Data Types

- **Primitive** data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.
- **Example**
- **int** a; **// valid**
- a=10; **// valid**
- a=10, 20, 30; **// invalid**



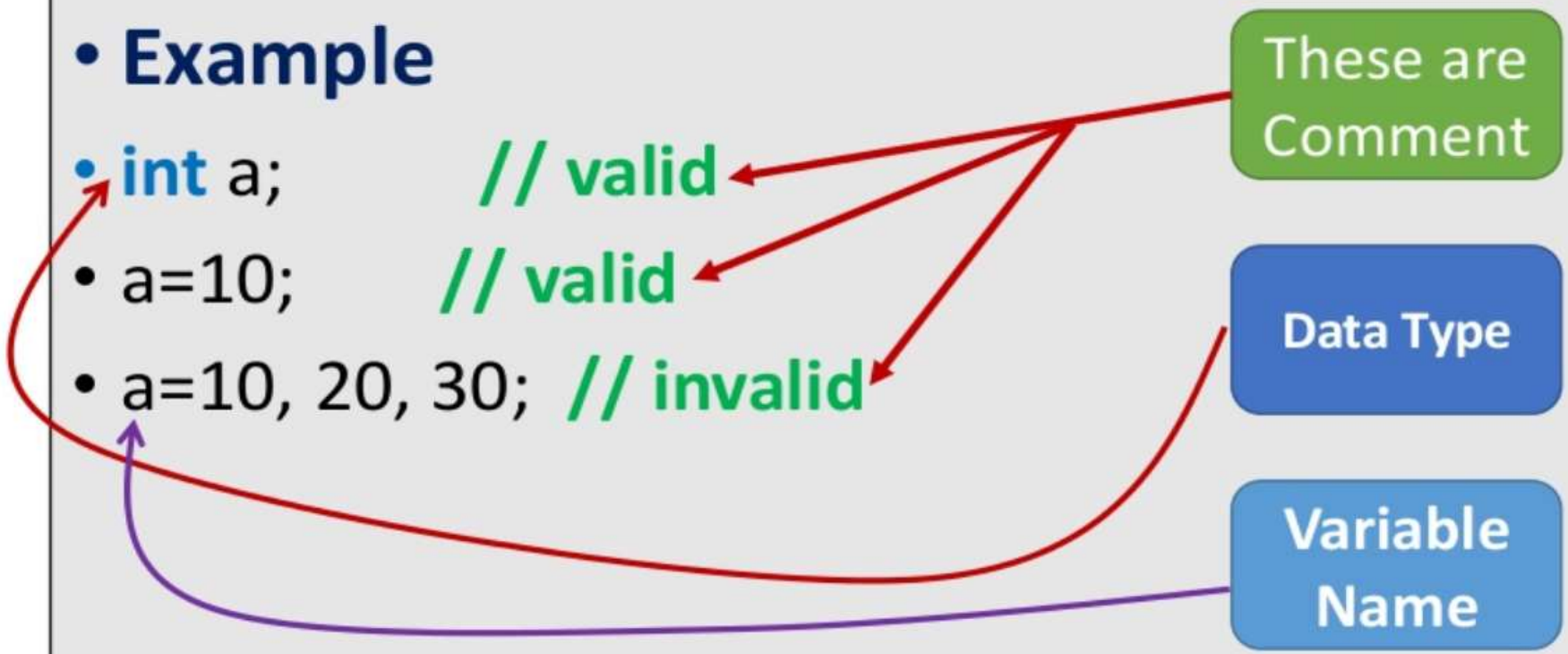
Primitive Data Types

- **Primitive** data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.
- **Example**
- **int** a; **// valid**
- a=10; **// valid**
- a=10, 20, 30; **// invalid**



Primitive Data Types

- **Primitive** data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.
- **Example**
- **int** a; **// valid**
- a=10; **// valid**
- a=10, 20, 30; **// invalid**



Comments in Java

- Commenting in **Java**. **Comments** are an integral part of any program. They help the person reading the code (often you) better understand the intent and functionality of the program.
- The Java language supports three types of comments –
- **Single line comment.**
- **//** symbol is use to write single line comment in java.
- **Multi Line Comment.**
- **/* */** symbol is use to write multi line comment in java.

Comments in Java

- **Single Line Comment**
- **Syntax**

```
//comment code....
```

- **Multi Line Comment**
- **/* */** symbol is use to write multi line comment in java.

```
/* comment code line 1
comment code line 2
      */
```

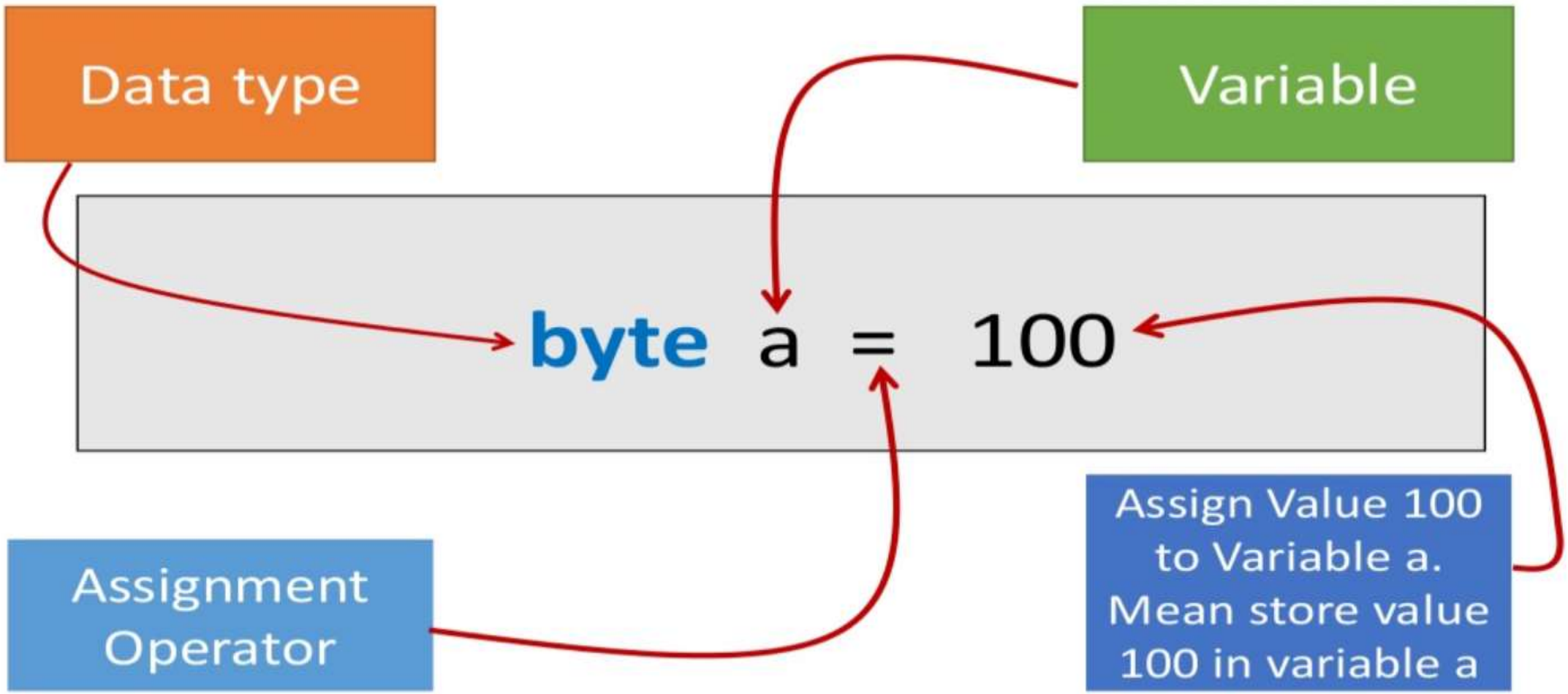
Comments in Java

Sr.No.	Comment & Description
1	**/* text */** The compiler ignores everything from /* to */.
2	**//text** The compiler ignores everything from // to the end of the line
3	**/** documentation */** This is a documentation comment and in general its called **doc comment**. The **JDK javadoc** tool uses *doc comments* when preparing automatically generated documentation.

The byte data type of Java

- Java has eight types of primitive data types to store data in the Java programs. The ***byte*** is one of the primitive data types in Java.
- **A few main points about Java byte data type:**
  - The b*yte* takes eight bits or one byte of signed memory.
  - Byte Is a numeric type.
  - Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
  - The Default value of the byte variable is 0.
  - The Minimum value of the **byte** type can be -128.
  - The Maximum value of the byte java type can be 127.
  - Example: **byte** a = 100, **byte** b = -50

Explanation



The short data type of Java

- Java has eight types of primitive data types to store data in the Java programs. The **Java *short*** is one of the primitive data types in Java.
- **A few main points about the Java short data type:**
  - Takes two bytes OR 16 bits of memory.
  - Is a numeric type.
  - Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer.
  - The default value of the **short** variable is 0.
  - The minimum value of the *short type* can be *–32768.*
  - The maximum value of the **short type** can be *32767.*
  - Example: **short** s = 10000, **short** r = -20000

The integer data type of Java

- Among the eight primitive data types in Java, the Java ***int*** is one of those, to be used in the programs.
- **A few main points about the Java int data type:**
  - The int type takes 32 bits or four bytes of memory.
  - It is a numeric type
  - Integer is generally used as the default data type for integral values unless there is a concern about memory.
  - The default value of the **int** variable is 0.
  - The minimum value of the Java integer type can be – *2,147,483,648.*
  - The maximum value of **int** Java type can be *2,147,483,647*.
  - Example: **int** a = 100000, **int** b = -200000

The long data type in Java

- The ***long*** is one of the primitive data types in Java, among the eight available data types. This is a numeric data type like byte, int etc.
- **A few main points about the Java log data type:**
  - Takes 64 bits or eight bytes memory.
  - The Java *long* is a numeric data type.
  - The default value of a **Long variable** is 0L.
  - The minimum value of the ***long*****,** *Java* data type can be-*9,223,372,036,854,775,808.*
  - This type is used when a wider range than int is needed.
  - Other numeric data types include the **byte**, **short**, **int** while the *float and double* with single and double precision, respectively.
  - Example: **long** a = 100000L, **long** b = -200000L

The float data type of Java

- The ***float*** is one of the primitive data types supported in Java.

- **A few main points about the Java float data type:**
  - Takes 32 bits or four bytes of memory.
  - Float is mainly used to save memory in large arrays of floating point numbers.
  - The **float** is a numeric type with single-precision.
  - The default value of the float variable is 0.0f.
  - Example: **float** f1 = 234.5f

The double data type of Java

- Among the eight primitive data types in Java, the ***double*** is one of those. It is like the float data type but with a double precision.
- **A few main points about the Java double data type are:**
  - A double type variable takes 64 bits or eight bytes memory.
  - This data type is generally used as the default data type for decimal values, generally the default choice.
  - The double is a numeric type with double-precision.
  - The Default value of the double variable is **0.0d.**
  - Example: **double** d1 = 123.4

The char data type in Java

- The ***char*** Java is one of the primitive data types in Java.
- **A few main points about the Java char data type:**
  - Takes 16 bits or two bytes memory.
  - Is used to store any type of character value.
  - The minimum value of char variable is 0.
  - The maximum value is 65,535.
  - Char data type is used to store any character
  - Example: **char** letterA = '**A**'

The boolean data type of Java

- Java has eight types of primitive data types to store data in the Java programs. The ***boolean*** *Java* is one of the primitive data types.
- **A few main points about the Java boolean data type:**
  - A Boolean variable may have two possible values: ***True*** *or* ***False*****.**
  - The default value of the Java Boolean variable is *false.*
  - An example of using the Boolean variable is in the conditional statement like the **if, switch** etc.
  - Example: **boolean** one = true

Primitive Data Types

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Primitive Data Types

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Why char uses 2 byte in java and what is \u0000 ?

because java uses unicode system rather than ASCII code system. \u0000 is the lowest range of unicode system.

Identifiers in Java

- All Java components require names. Name used for classes, methods, interfaces and variables are called **Identifier**. Identifier must follow some rules. Here are the rules:

✓All identifiers must start with either a letter(a to z or A to Z) or currency character($) or an underscore.

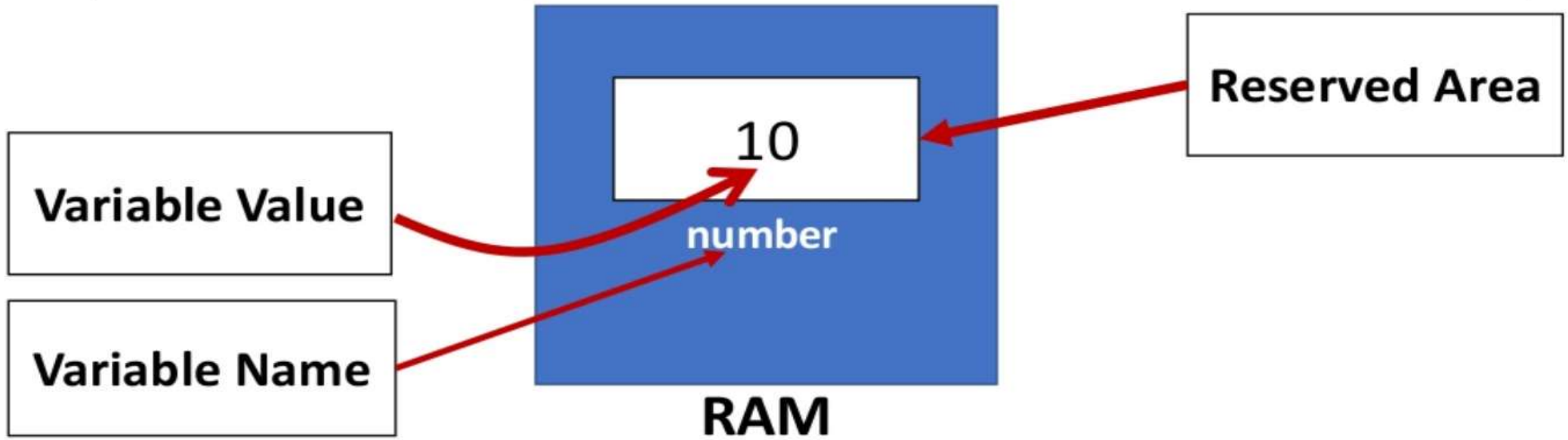
✓After the first character, an identifier can have any combination of characters.

✓A Java **keyword** cannot be used as an identifier.

✓Identifiers in Java are case sensitive, foo and Foo are two different identifiers.

Variable in Java

- **Variable and Data Type**
- Variable is a name of memory location and Data Type specifies size and the type of value that can be stored in a variable (identifier).
- **Variable in Java:**
- **Variable** – is name of reserved area allocated in memory.



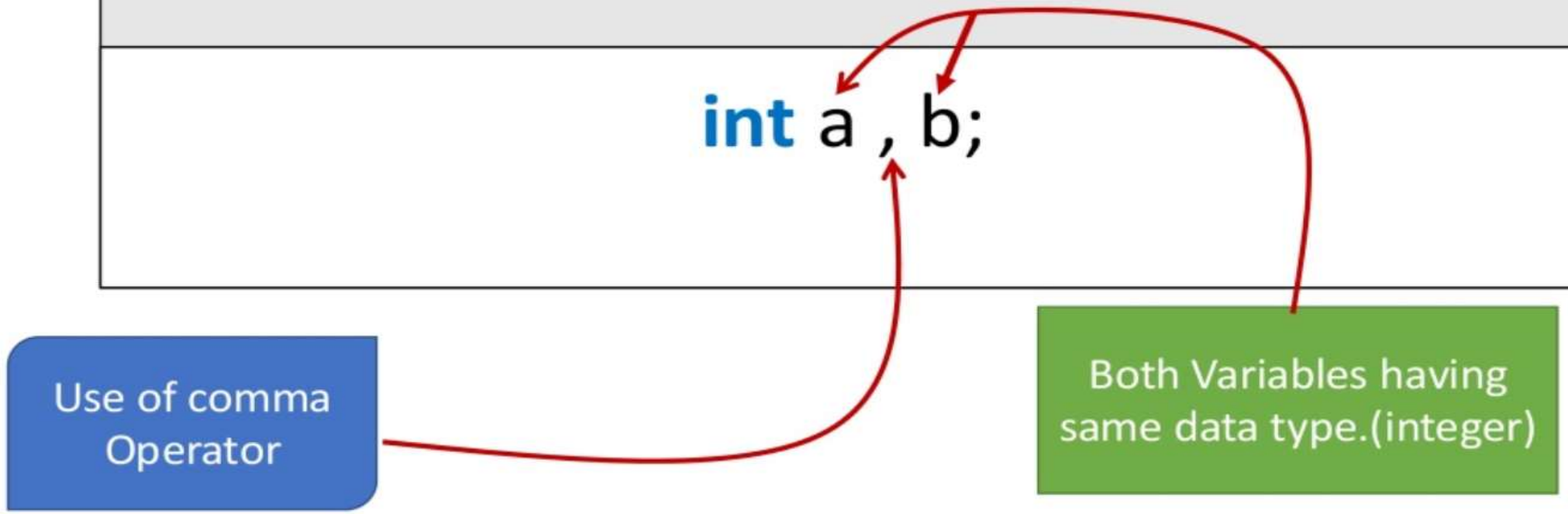
Variable Declaration and initializes in Java

```
data type Variable_name = value;
```

Comma operator (,)

- The comma operator (,) is used to separate two or more expressions that are included where only one expression is expected.
- For example, the following code:

```
int a , b;
```



Example of Variables

```
int a, b, c;                    // Declares three ints, a, b, and c.
int a = 10, b = 10;  // Example of initialization
byte B = 22;                // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';               // the char variable a is initialized with value 'a'
```

Java Naming Conventions

- Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

- But, it is not forced to follow. So, it is known as convention not rule.

- All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

Java Naming Conventions

- **Major naming convention in java:**
- package in java should be in small letters as **java.lang**.
- Each word of interfaces should start with capital letter as **Serializable**
- Each word of classes should start with capital letter as **Sting**
- Method name should be in small letters as **println()**.
- Constants should be in all capital letters like **MAX_VALUE**
- Variable name should be start with small letter and change as per word change.

Java Naming Conventions

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

Keywords in Java

- In Java, a **keyword** is a word with a **predefined** meaning in Java programming language syntax. **Reserved** for **Java**, **keywords** may **not** be **used** as identifiers for **naming** variables, classes, methods or other identifier.
- **Examples:**
  - **final**
  - **class**
  - **this**
  - **synchronized**

A complete List of Java Keywords

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

Variable Assignment

- Value is assigned to a variable if that is already declared or initialized.

- a + 2 = b + 10 **Wrong** Statement
- z = x + 4 **Ok** Statement
- a + 14 = m **Wrong** Statement
- m = x + y **Ok** Statement

Quadratic Equation

- **In algebra**
  - Quadratic = $ax^2+bx + c$

- **In Java**
  - Quadratic = a*x*x + b*x + c

Quadratic Formula

- **In algebra**
  - $X=-b\pm\sqrt{b^2-4ac}/2a$
- **In Java**
  - x = (-b +- sqrt((b^2)-4ac))/2a

Variable Program Example

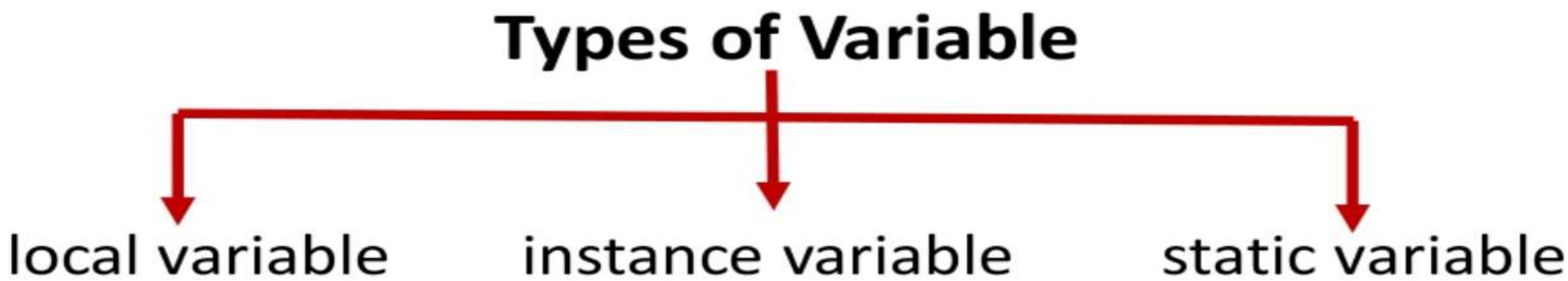
```
public class JavaApplication1
{

        public static void main(String[] args) {
        int number=10;
       System.out.println("Value of number is "+number);
    }

}
```

Declare and initialized a variable

+ Operator is use to Concatenate Strings

Types of Variable

- There are three types of variables in java
- local variable
- instance variable
- static variable



Local Variable

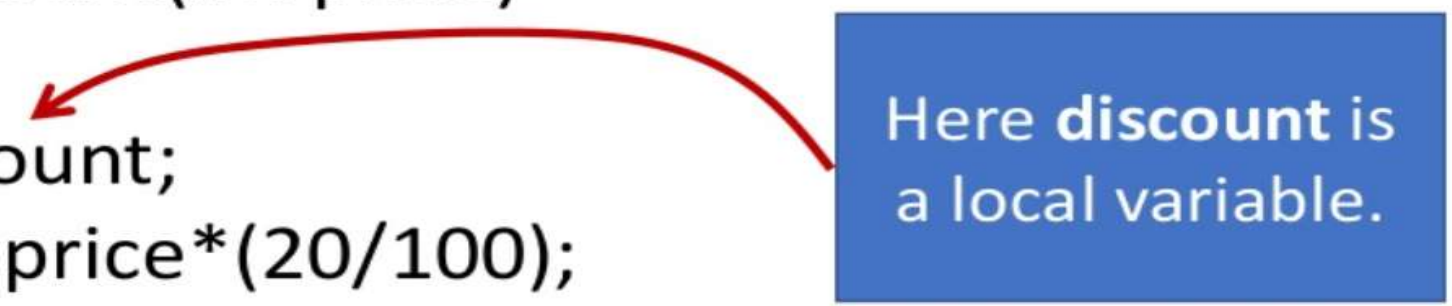
- A variable that is declared inside the method is called local variable.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Local Variable

- A variable that is declared inside the method is called local variable.

Example :

```
float getDiscount(int price)
{
    float discount;
    discount=price*(20/100);
 return discount;
}
```



Instance Variable

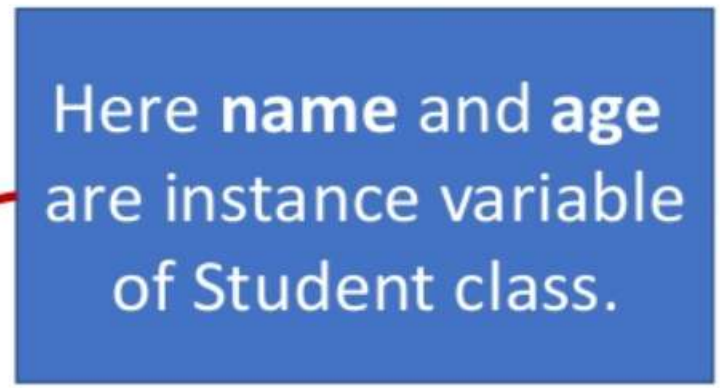
- A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.
- **Instance variable** in java is automatically initialized with default value
- **Instance variable** will be accessible any where within class.

Instance Variable

- A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.

Example :

```
class Student
{
    String name;
     int age;
}
```



Static variable/Class Variable

- A variable that is declared as static is called static variable. It cannot be local.
- Class variable has only one copy for each class, regardless of how many objects are created from it.
- Class variable is created when program starts and ends with the program.
- It has same default values as instance variable.
- Static can only be accessed by using it's fully qualified name as : ClassName.VariableName

Static variable/Class Variable

- A variable that is declared as static is called static variable. It cannot be local.

Example :

```
class Student
{
    String name;
    int age;
    static int DepartCode=1101;
}
```

Example to Understand the types of Variables

```
class Test{


        int id;    //instance variable
        static float salary;    //static  variable
   public static void main(String args[]){
     int a=10;      //local variable
   }
}
```

Static variable Example

```
class A
{
   static int number=10;  //static Variable
}
 class Test
{
   public static void main(String args[])
  {
     System.out.println(A.number);   // static variable called
  }
}
```

Local variable Example

```
class Test
{
  public static void main(String args[])
  {
      int number=10;   //Local Variable
     System.out.println(number);
  }
}
```

Output is:
10

Static Variable can not be Local

In Java applications, static variables are always class level variables, they never be local variables.

```
class Test
{
  public static void main(String args[])
 {
    static int number=10;  //Compiler Error
    System.out.println(number);
 }
}
```

Unicode System

- Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.
- **Why java uses Unicode System?**

Before Unicode, there were many language standards:

- ASCII (American Standard Code for Information Interchange) for the United States.
- ISO 8859-1 for Western European Language.
- KOI-8 for Russian.
- GB18030 and BIG-5 for Chinese, and so on.

Unicode System

Problem

- **This caused two problems:**
  1. A particular code value corresponds to different letters in the various language standards.
  2. The encodings for languages with large character sets have variable length. Some common characters are encoded as single bytes, other require two or more byte.
- **Solution**
  - To solve these problems, a new language standard was developed i.e. Unicode System.
  - In unicode, character holds 2 byte, so java also uses 2 byte for characters.
  - lowest value:\u0000
  - highest value:\uFFFF

Type Casting in Java

- The process of converting one data type to another is called **casting**.
- Casting is often necessary when a function returns a data of type in different form then we need to perform an operation.
- Under above certain circumstances Type conversion can be carried out automatically, in other cases it must be "forced" manually (explicitly).

Type Casting in Java

- In Java type casting is classified into two types:
  - Widening Casting(Implicit)
  - Narrowing Casting(Explicitly)

CASTING

Converting a data type to another type

- Implicit casting: Happens automatically when converting from a narrower range data type to a wider range data type
  → converting an int to a double/float/long
  → converting a float to a double

- Explicit casting: Does not happen automatically. Should be done by the programmer when converting from a wider to a narrower data type
  → converting a double/float/long to an int

IMPLICIT CASTING

```
double d1 = 4; // int → double
double d2 = 5.7f; // float → double
long l1 = 100; // int → long
```

Implicit casting happens because:

- The range of a `double` is wider than an `int`
- The range of a `double` is wider than a `float`
- The range of a `long` is wider than an `int`

EXPLICIT CASTING

```
int i1 = 4.5; // ERROR
int i2 = 8L; // ERROR
float f1 = 4.5; // ERROR
```

Implicit casting can not happen because:

- The range of an `int` is narrower than a `double`
- The range of an `int` is narrower than a `long`
- The range of a `float` is narrower than a `double`

→ Explicit casting is used

EXPLICIT CASTING

(new data type) expression

```
int i1 = (int) 4.5;
int i2 = (int) 8L;
float f1 = (float) 4.5;
```

The programmer tells Java to do the casting:

- (int) 4.5 → 4 (data loss)
- (int) 8L → 8
- (float) 4.5 → 4.5F

Widening Casting(Implicit)

Java's widening conversions are:

From a **byte** to a **short**, an **int**, a **long**, a **float**, or a **double**

From a **short** to an **int**, a **long**, a **float**, or a **double**

From a **char** to an **int**, a **long**, a **float**, or a **double**

From an **int** to a **long**, a **float**, or a **double**

From a **long** to a **float** or a **double**

From a **float** to a **double**

Widening Casting(Implicit)

byte can be converted to **short**, **int**, **long**, **float**, or **double**

Short can be converted to **int**, **long**, **float**, or **double**

char can be converted to **int**, **long**, **float**, or **double**

int can be converted to **long**, **float**, or **double**

long can be converted to **float** or **double**

float can be converted to **double**

Narrowing Casting(Explicitly)

From a **byte** to a **char**

From a **short** to a **byte** or a **char**

From a **char** to a **byte** or a **short**

From an **int** to a **byte**, a **short**, or a **char**

From a **long** to a **byte**, a **short**, a **char**, or an **int**

From a **float** to a **byte**, a **short**, a **char**, an **int**, or a **long**

From a **double** to a **byte**, a **short**, a **char**, an **int**, a **long**, or a **float**

Narrowing Casting(Explicitly)

short can be converted to **byte** or **char**

char can be converted to **byte** or **short**

int can be converted to **byte**, **short**, or **char**

long can be converted to **byte**, **short**, **char**, or **int**

float can be converted to **byte**, **short**, **char**, **int**, or **long**

double can be converted to **byte**, **short**, **char**, **int**, **long**, or **float**

Widening or Automatic type Conversion

- **A data type of lower size (occupying less memory) is assigned to a data type of higher size.**
- This is done implicitly by the JVM. The lower size is widened to higher size.
- The target type is larger than the source type

Examples:

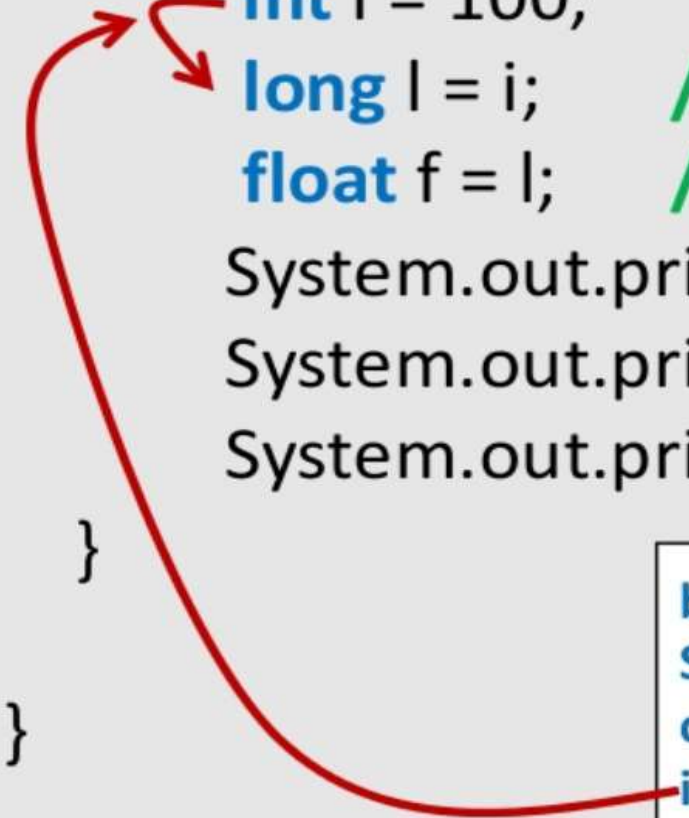
```
int x = 10;                  // occupies 4 bytes
double y = x;                // occupies 8 bytes
System.out.println(y);       // prints 10.0
```

Widening Conversion Program Example

```
public class JavaApplication1 {

    public static void main(String[] args) {
            int i = 100;
            long l = i;       //no explicit type casting required
            float f = l;      //no explicit type casting required
           System.out.println("Int value "+i);
           System.out.println("Long value "+l);
           System.out.println("Float value "+f);
    }

}
```

Widening Conversion Program Example

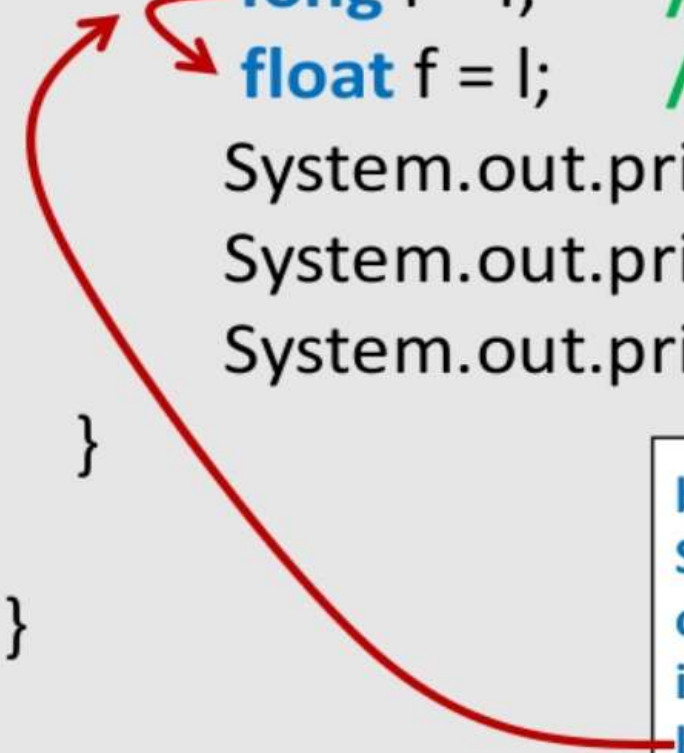
```
public class JavaApplication1 {

    public static void main(String[] args) {
        int i = 100;
        long l = i;       //no explicit type casting required
        float f = l;      //no explicit type casting required
        System.out.println("Int value "+i);
        System.out.println("Long value "+l);
        System.out.println("Float value "+f);
    }

}
```



byte can be converted to short, int, long, float, or double
Short can be converted to int, long, float, or double
char can be converted to int, long, float, or double
int can be converted to long, float, or double
long can be converted to float or double

Widening Conversion Program Example

```
public class JavaApplication1 {

    public static void main(String[] args) {
        int i = 100;
        long l = i;       //no explicit type casting required
        float f = l;      //no explicit type casting required
        System.out.println("Int value "+i);
        System.out.println("Long value "+l);
        System.out.println("Float value "+f);
    }

}
```



byte can be converted to short, int, long, float, or double
Short can be converted to int, long, float, or double
char can be converted to int, long, float, or double
int can be converted to long, float, or double
long can be converted to float or double

Widening Conversion Program Example

```
public class JavaApplication1 {


    public static void main(String[] args) {
            int i = 100;
            long l = i;       //no explicit type casting required
            float f = l;      //no explicit type casting required
           System.out.println("Int value "+i);
           System.out.println("Long value "+l);
           System.out.println("Float value "+f);
    }

}
```

Output is :

Int value 100

Long value 100

Float value 100.0

Narrowing or Explicit type Conversion

- **A data type of higher size (occupying more memory) cannot be assigned to a data type of lower size.**
- This is not done implicitly by the JVM and requires **explicit casting**; a casting operation to be performed by the programmer.
- The higher size is narrowed to lower size.

Example of Explicit type Conversion

- **Examples:**

```
double x = 10.5;     // 8 bytes
int y = x;     // 4 bytes ;  Raies compilation Error
```

In the above code, 8 bytes double value is narrowed to 4 bytes int value. It raises error. Let us explicitly type cast it.

- **Examples:**

```
double x = 10.5;
int y = (int) x;
```

The double **x** is explicitly converted to int **y**. The thumb rule is, on both sides, the same data type should exist.

Narrowing Conversion Program Example

```
14    public class JavaApplication1 {
15
16        public static void main(String[] args) {
17            double d = 100.04;
18            long l = (long)d;  //explicit type casting required
19            int i = (int)l;    //explicit type casting required
20
21            System.out.println("Double value "+d);
22            System.out.println("Long value "+l);
23            System.out.println("Int value "+i);
24
25        }
26
27    }
```

javaapplication1.JavaApplication1 > main > d >

Output - JavaApplication1 (run)

```
run:
Double value 100.04
Long value 100
Int value 100
BUILD SUCCESSFUL (total time: 0 seconds)
```

Narrowing Conversion Program Example

```
public class JavaApplication1 {

    public static void main(String[] args) {
        double d = 100.04;
        long l = (long)d;  //explicit type casting required
        int i = (int)l;    //explicit type casting required
        System.out.println("Double value "+d);
        System.out.println("Long value "+l);
        System.out.println("Int value "+i);

    }

}
```

Narrowing Conversion Program Example

```
public class JavaApplication1 {

    public static void main(String[] args) {
        double d = 100.04;
        long l = (long)d;  //explicit type casting required
        int i = (int)l;    //explicit type casting required
        System.out.println("Double value "+d);
        System.out.println("Long value "+l);
        System.out.println("Int value "+i);

    }

}
```

short can be converted to byte or char
char can be converted to byte or short
int can be converted to byte, short, or char
long can be converted to byte, short, char, or int
float can be converted to byte, short, char, int, or long
double can be converted to byte, short, char, int, long, or

Narrowing Conversion Program Example

```
public class JavaApplication1 {


    public static void main(String[] args) {
            double d = 100.04;
            long l = (long)d;  //explicit type casting required
            int i = (int)l;    //explicit type casting required
            System.out.println("Double value "+d);
            System.out.println("Long value "+l);
            System.out.println("Int value "+i);


    }


}
```

Output is :

Double value 100.04

Long value 100

Int value 100

Boolean Casting

- A boolean value cannot be assigned to any other data type. Except boolean, all the remaining 7 data types can be assigned to one another either implicitly or explicitly; but boolean cannot. We say, boolean is **incompatible** for conversion. Maximum we can assign a boolean value to another boolean.
- Following raises error.

```
boolean x = true;
int y = x;          // Error
```

```
boolean x = true;
int y = (int) x;          // Error
```

Java Operators

- Java provides a rich set of operators environment. Java operators can be divided into following categories
- Arithmetic operators
- Relation operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Misc operators

Arithmetic Operators

- Arithmetic operators are used to make mathematical expressions like addition multiplication.
- Arithmetic operators operator can operate on built in data type of java.
- List of arithmetic operator
- **+(Addition operator)**
- **-(subtraction operator)**
- ***(multiplication operator)**
- **/(division operator)**
- **%(Modulus operator)**

Arithmetic Operators

- Arithmetic operators are used in mathematical expression in the same way that are used in algebra.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denumerator
%	remainder of division
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one

Arithmetic Operators

- Given table shows all the Arithmetic operator supported by Java Language. Lets suppose variable **A** hold 8 and **B** hold 3.

Operator	Example (int A=8, B=3)	Result
+	A+B	11
-	A-B	5
*	A*B	24
/	A/B	2
%	A%4	0

Java Program add two Numbers

```
public class JavaApplication1 {

    public static void main(String[] args) {
      int firstNumber=10;
      int secondNumber=5;
      int result;
      result=firstNumber+secondNumber;
      System.out.println("Sum of Two Number is "+result);

    }

}
```

Accepting Input from User

- One of the strengths of Java is the huge libraries of code available to you. This is code that has been written to do specific jobs. All you need to do is to reference which library you want to use, and then call a method into action. One really useful class that handles input from a user is called the **Scanner** class.
- The Scanner class can be found in **java.util** library. To use the Scanner class, you need to reference it in your code. This is done with the keyword **import**.

```
import java.util.Scanner;
```

Accepting Input from User

- The import statement needs to go just above the Class statement:

```
import java.util.Scanner;
  public class JavaApplication1 {


}
```

- This tells java that you want to use a particular class in a particular library - the Scanner class, which is located in the java.util library.
- The next thing you need to do is to create an object from the Scanner class. (A class is just a bunch of code. It doesn't do anything until you create a new object from it.)

Accepting Input from User

- To create a new Scanner object the code is this:

```
Scanner user_input = new Scanner( System.in );
```

- So instead of setting up an **int** variable or a **String** variable, we're setting up a **Scanner** variable. We've called ours **user_input**.
- After an equals sign, we have the keyword **new**. This is used to create new objects from a class. The object we're creating is from the Scanner class. In between round brackets we have to tell java that this will be System Input (System.in).

Accepting Input from User

- To get the user input, you can call into action one of the many methods available to your new Scanner object. One of these methods is called **next**. This gets the next string of text that a user types on the keyboard:

```
int first_Number;
first_Number = user_input.next( );
```

Accepting Input from User

- So after our user_input object we type a dot. You'll then see a popup list of available methods. Double click **next** and then type a semicolon to end the line. We can also print some text to prompt the user:

```
int firstNumber;
System.out.print("Enter first Number: ");
firstNumber = user_input.nextInt( );
```

- Notice that we've used **print** rather than **println**. The difference between the two is that println will move the cursor to a new line after the output, but print stays on the same line.

Accepting Input from User

- We'll add a prompt for a Second Number, as well:

```
int secondNumber;
System.out.print("Enter Second Number: ");
secondNumber = user_input.next( );
```

- This is the same code, except that java will now store whatever the user types into our secondNumber variable instead of our firstNumber variable.

Talking Input from user and Add two Number

```
import java.util.Scanner;
public class JavaApplication1 {
    public static void main(String[] args) {
      int firstNumber, secondNumber, Result;
      Scanner user_input = new Scanner(System.in);
      System.out.println("Enter First Number: ");
      firstNumber=user_input.nextInt();
      System.out.println("Enter Second Number: ");
      secondNumber=user_input.nextInt();
      Result=firstNumber+secondNumber;
      System.out.println("Sum of Two Number is : "+Result);

    }
}
```

```
Scanner user_input = new Scanner(System.in);
```

- **Scanner**: The Scanner class is a class in java.util, which allows the user to read values of various types.
- **User_input:** An object name defined by the user, you can use any object name like year, num, input Device and so on.
- **System.in**: An Input Stream which is typically connected to keyboard input of console programs.
- **new Scanner (System.in):** Creates a Scanner object that is connected to the System.in object. In other words, the created Scanner object is connected to the default input device. The keyword "new" is required by Java; you will use it whenever you create objects that are more complex than the simple datatypes.
- The **assignment operator** in the Scanner declaration statement assigns the value of the new object--that is, its memory address--to the "console" object in the program

Scanner Class

- The Scanner class has several methods which are used to take different types of inputs. They are listed in the table below.

Method	Description
nextByte()	Accept a byte
nextShort()	Accept a short
nextInt()	Accept an int
nextLong()	Accept a long
next()	Accept a single word
nextLine()	Accept a line of String
nextBoolean()	Accept a boolean
nextFloat()	Accept a float
nextDouble()	Accept a double

JOptionPane in Java

The JOptionPane is a class that is used to provide standard dialog boxes. It is a part of Java Swing which is used for creating window-based applications.

JOptionPane is a component from Java Swing and it deals with dialog boxes especially.

The dialog boxes can be of any type such as confirm dialog box, message dialog box or input dialog box.

These dialog boxes can be used to display information to the user or to get input from the user.

Add Two Number using Java Option Pane

```
13  import javax.swing.JOptionPane;
14  public class JavaApplication1 {
15
16      public static void main(String[] args) {
17          int firstNumber,secondNumber,Result;
18          firstNumber=Integer.parseInt(JOptionPane.showInputDialog("Enter First Number"));
19          secondNumber=Integer.parseInt(JOptionPane.showInputDialog("Enter second Number"));
20          Result=firstNumber+secondNumber;
21          JOptionPane.showMessageDialog(null, "Result is: "+Result);
22
23      }
24
25  }
```

```
import javax.swing.JOptionPane;
public class JavaApplication1 {

    public static void main(String[] args) {
      int firstNumber, secondNumber, Result;
      firstNumber=Integer.parseInt(JOptionPane.showInputDialog
     ("Enter First Number"));
      secondNumber=Integer.parseInt(JOptionPane.showInputDialog
     ("Enter second Number"));
      Result=firstNumber+secondNumber;
      JOptionPane.showMessageDialog(null, "Result is: "+Result);

    }
}
```

Integer.parseInt used to convert String into integer because JOptionPane always take the Input Values in the form of String.

Getting String from JOptionPane

```
13  import javax.swing.JOptionPane;
14  public class JavaApplication1 {
15
16      public static void main(String[] args) {
17          String name;
18          name=JOptionPane.showInputDialog("Enter First Number");
19          JOptionPane.showMessageDialog(null,"My Name is: "+name);
20
21
22      }
23
24  }
```