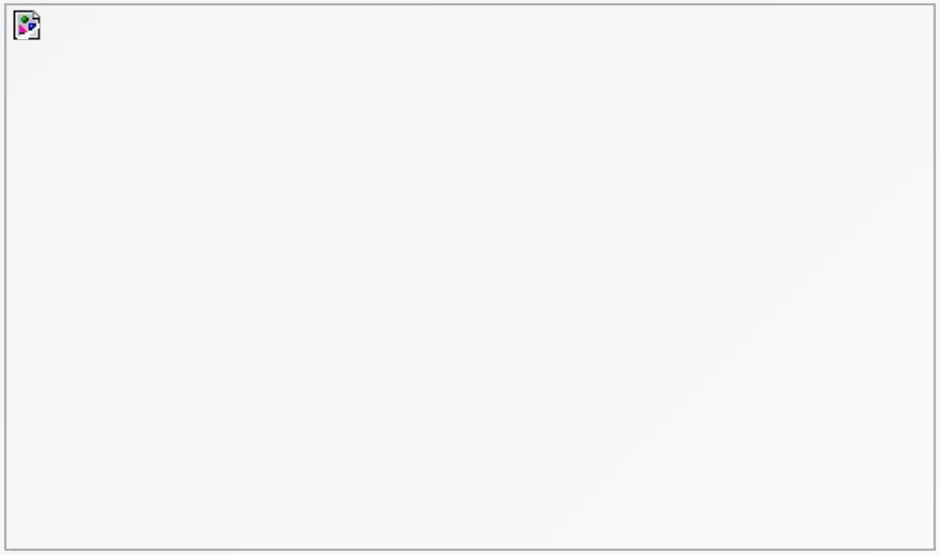
Variable

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.

Java Basic



int data=10;//Here data is variable

Types of Variables:

There are three types of variables in java:

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Java example

Example to understand the types of variables in java :

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

Java example

```
class Simple
{
public static void main(String[] args)
{
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}
}
```

Java example

```
class Simple{
public static void main(String[] args)
{
int a=10;
float f=a;
int b=(int)f;
System.out.println(a);
System.out.println(f);
}
}
```

Java example

```
class Simple
{
public static void main(String[] args){
float f=10.5f;
//int a=f;//Compile time error
int a=(int)f;
System.out.println(f);
System.out.println(a);
}
}
```

Java example

```
class Simple{
public static void main(String[] args){
//Overflow
int a=133;
byte b=(byte)a;
System.out.println(a);
System.out.println(b);
}}
```

output :

133 -123

Java example

```
class Simple
{
public static void main(String[] args){
byte a=10;
byte b=10;
//byte c=a+b;//Compile Time Error: because a+b=20 will be int
byte c=(byte)(a+b);
System.out.println(c);
}
}
```

Data Types in Java

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 Integer, Character, Boolean, and Floating Point.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

Data Types in Java

Java Primitive Data Types :

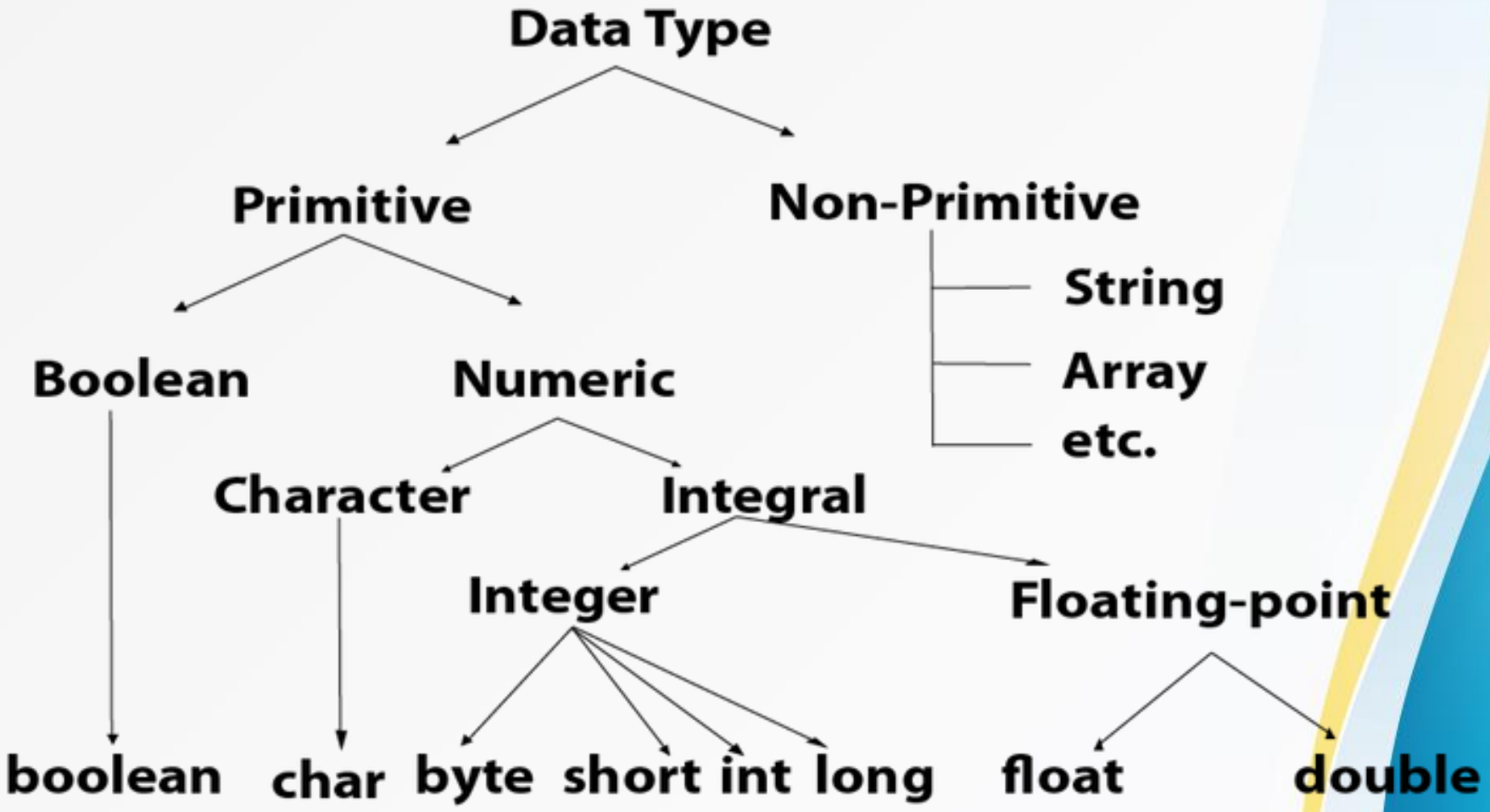
In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

Data Types in Java

There 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

Data Types in Java



Data Types in Java

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

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

This caused two problems:

- A particular code value corresponds to different letters in the various language standards.
- 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.

Unicode System

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

Operators in Java

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

There are many types of operators in java which are given below:

- Unary Operator
- Arithmetic Operator
- Shift Operator
- Relational Operator
- Bitwise Operator
- Logical Operator
- Ternary Operator
- Assignment Operator

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	== , !=

Operators in Java

Operator Type	Category	Precedence
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= ,+=, -=, *= ,/= ,%= ,&=, ^=, |=, <<=, >>=, >>>=

Java Unary Operator Example: ++ and --

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

Java Unary Operator Example : ++ and --

```
class OperatorExample
{
public static void main(String args[]){
int a=10;
int b=10;
System.out.println(a++ + ++a);
System.out.println(b++ + b++);

}
}
```

Java Unary Operator Example: ~ and !

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);
System.out.println(~b);
System.out.println(!c);
System.out.println(!d);
}}
```

Java Unary Operator Example: ~ and !

```
    System.out.println(~a)
```

-11 (minus of total positive value which starts from 0)

```
    System.out.println(~b);
```

9 (positive of total minus, positive starts from 0)

```
    System.out.println(!c);
```

false (opposite of boolean value)

```
    System.out.println(!d);
```

true

Java Left Shift Operator Example

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);
System.out.println(10<<3);
System.out.println(20<<2);
System.out.println(15<<4);
}}
```

Java Right Shift Operator Example

The operator '>>' uses the sign bit (left most bit) to fill the trailing positions after shift. If the number is negative, then 1 is used as a filler and if the number is positive, then 0 is used as a filler.

```
class OperatorExample
{
   public static void main(String args[]){
   System.out.println(10>>2);
   System.out.println(20>>2);
   System.out.println(20>>3);
   }
}
```

Java Shift Operator Example: >> vs >>>

```
class OperatorExample{
public static void main(String args[]){
    //For positive number, >> and >>> works same
    System.out.println(20>>2);
    System.out.println(20>>>2);
    //For negative number, >>> changes parity bit
(MSB) to 0
    System.out.println(-20>>2);
    System.out.println(-20>>>2);
}}
```

output :

```
5     5     -5   1073741819
```

Logical && and Bitwise & Example

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

```
class OperatorExample{
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
}}
```

Logical && and Bitwise & Example

```
class OperatorExample{
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);
System.out.println(a<b&a++<c);//false & true = false
System.out.println(a);
}}
```

output :
false 10 false 11

Logical || and Bitwise | Example

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.

Logical || and Bitwise | Example

```
class OperatorExample{
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
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);
}}
```

Java Ternary Operator Example

Java Ternary operator is used as one liner replacement for if-then-else statement and used a lot in java programming. it is the only conditional operator which takes three operands.

```
class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Operator Precedence

When operators have the same precedence, the earlier one binds stronger

highest			
()	[]	.	
++	--	~	!
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|			
&&			
||			
?:			
=	op=		
lowest			

Selection Statements

- if-else statement

```
if (expression) {
                    statement1
                    }
          else
                    {
                    statement2
                    }
```

- if-else-if statement

```
if (expression1) statement1
    else if (expression2) statement2
    else if (expression3) statement3
    ...
    else statement
```

Switch Statement

- switch provides a better alternative than if-else-if when the execution follows several branches depending on the value of an expression

```
switch (expression) {
        case value1: statement1; break;
        case value2: statement2; break;
        ...
        default: statement;
        }
```

Switch Statement

- *Expression must be of type byte, short, int or char*
- *Each of the case values must be a literal of the compatible type*
- *Case values must be unique*
- *Break makes sure that only the matching statement is executed*

Iteration Statement

Java iteration statements enable repeated execution of part of a program until a certain termination condition becomes true

- while statement

```
while (expression)
        statement
```

- How to run an infinte while loop ?
- do-while statement

```
do
        statement
while (expression);
```

Iteration Statement

for statement

```
for (initialization; termination; increment)
    statement
```

- How to run an infinte for loop ?

Java Labeled For Loop

```
public class LabeledForExample {
public static void main(String[] args) {
    aa:
        for(int i=1;i<=3;i++){
            bb:
                for(int j=1;j<=3;j++){
                    if(i==2&&j==2){
                        break aa;
                    }
                    System.out.println(i+" "+j);
                }
        }
}
}
```

Java Labeled For Loop

Output:

```
1 1
1 2
1 3
2 1
```

Java Labeled For Loop

```
public class LabeledForExample2 {
public static void main(String[] args) {
    aa:
        for(int i=1;i<=3;i++){
            bb:
                for(int j=1;j<=3;j++){
                    if(i==2&&j==2){
                        break bb;
                    }
                    System.out.println(i+" "+j);
                }
        }
}
}
```

Java Labeled For Loop

Output:

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Jump Statement

Java jump statements enable transfer of control to other parts of program

- break statement
  - *break;*
  - Java does not have goto statement
  - *break label;*
  - *label: { ... }*
- continue statement
  - The break statement terminates the block of code, in particular it terminates the execution of an iterative statement

Jump Statement

- The continue statement forces the early termination of the current iteration to begin immediately the next iteration

- *continue;*
- *continue label;*

Java Break Statement with Loop

```
public class BreakExample {
public static void main(String[] args) {
    for(int i=1;i<=10;i++){
        if(i==5){
            break;
        }
        System.out.println(i);
    }
 }
}
```

Java Break Statement with Loop

Output:

1
2
3
4

Java Break Statement with Inner Loop

```
public class BreakExample2 {
public static void main(String[] args) {
        for(int i=1;i<=3;i++){
            for(int j=1;j<=3;j++){
                if(i==2&&j==2){
                    break;
                }
                System.out.println(i+" "+j);
            }
        }
}
}
```

Java Break Statement with Inner Loop

Output:

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Java Continue Statement Example

```
public class ContinueExample {
public static void main(String[] args) {
    for(int i=1;i<=10;i++){
        if(i==5){
            continue;
        }
        System.out.println(i);
    }
}
}
```

Java Continue Statement Example

Output:

1
2
3
4
6
7
8
9
10

Java Continue Statement with Inner Loop

```
public class ContinueExample2 {
public static void main(String[] args) {
        for(int i=1;i<=3;i++){
            for(int j=1;j<=3;j++){
                if(i==2&&j==2){
                    continue;
                }
                System.out.println(i+" "+j);
            }
        }
}
}
```

Java Continue Statement with Inner Loop

Output:

```
1 1
1 2
1 3
2 1
2 3
3 1
3 2
3 3
```