Unit 2

Tokens, Expression and Control Structure

Tokens:

Tokens are the smallest unit of program that are identified by the compiler i.e., meaningful to the compiler. Tokens include identifiers, keywords, literals, operators, expression and separators. Tokens are used by compiler to detect errors.

Identifiers are the proper name given to variable, constant, class, methods, array etc. Identifier may be sequence of character, symbols like underscore, dollar etc. that are used to give proper name to the class, variable, methods etc. There are some rules that needs to be followed while declaring identifier:

- Identifier should not begin with number or digits but can contains digits beside first position. For example, 2hello, 6var $\longrightarrow$ invalid. Hello211 $\longrightarrow$valid
- Identifier for class name should begin with capital letter. For example: class First, class BCA
- The first letter of an identifier must be letter, underscore or a dollar sign. For example: count, $check, Sum, calcTotal
- As java is case sensitive: identifiers are also case sensitive. For example: hello is different from Hello.
- The white space cannot be included in the identifier

Literals:

Literals are the constant value i.e., values that are not assigned to variables. Literals simply means something that evaluates to itself such as number (44, 55 etc.) or string (“hello”) etc. a literal can be used anywhere if the value of its type is allowed. Some literals are:

100: integer literal

55.5: float literal

‘a’: char literal

“this is OOP in java”: string literal

Separators:

Separators are the symbols that are used to denote specific information in code such as some separator denotes end of statement, some are used to separate other token, some denote start and end of the program etc. Some separators used in java:

Symbol of separator	Name of Separator	Purpose
()	Parenthesis	Used for storing parameter in method, used for defining precedence in expression, used for storing expression in control statement and used for surrounding cast types.
{ }	Braces	Used for defining start and ending of code block in class, method, local scope etc. Values of

		automatically initialized array are also store in braces
[ ]	Square bracket	Used for declaring array.
,	Comma	Used to separate two value, statement and parameter. Also used to combine statement together inside a for statement.
;	Semicolon	Used to denote end of statement or terminate statement
.	Period	Used to separate package name from sub package and classes. Used to separate a variable or method from a reference variable
=	Assignment Operator	Used to assign a variable and constant
::	Colon	Used to create method or constructor reference

Java Keywords:

Keywords are the predefined reserved words by java and each keyword has a special meaning. It is always written in lower case and cannot be use as identifier i.e., cannot be used for naming variable, class, method etc. There are 50 keywords currently defined by java. Keywords form the foundation of java language when combined with syntax of operator and separators.

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

In addition to these keywords, java also reserves these values: **true, false and null.**

Comments:

Comments are used to add some description of code i.e., it provides information about code to make it more readable. Comment's line is not executable statement so, any things inside comment line are ignored by java. Comments explains the operation of program to anyone who is reading a source code. It is mostly used for documenting the code. There are two ways to add comment in java:

Sigle line comments is created by using pair of forward slash (//) which will turn the whole single line into comment i.e., that particular line will be ignored by compiler. It is not application for more than one line. For example

// running the first java program

Multiline comments begin with **/*** symbol and end with ***/** symbol which will turn more than one line into comment. Anything inside /* */ symbol is ignored by java compiler. We can use comment anywhere inside code. For example

/* this is first java program

We are running the first java program in NetBeans

*/

Variable in Java:

Variables are used to store data temporarily i.e. they are like container for storing data. When variable is declared then the memory location is set for storing a value. In other, it can be referred as name of the reserved memory location in which some values can be stored.

Declaring a variable:

Variables are declared with its type. In java variable must be declared before it is used. A variable is defined by combination of identifier (name), type (what kind of value it will store such as integer, float etc.) and an optional initializer (default value).

Syntax:

type identifier = value [optional];

for example;

int a = 50;

type identifier value [optional initializer]

we can declare variable in many ways:

int sum, diff, mul; //declaring three int type variable

int a = 50, b=60; //here comma is used to declare more than one variable of same type

char x = ‘a’; //declaring variable x which contains the value a;

Dynamic Initialization:

Java allows variable to be initialized dynamically by using any expression of valid type with that of variable. Expression is any statement that perform some operation and generated a value. For example: 10 + 5 +6 is an expression which will perform addition operation on that three operand and give some result. Expression consists of variables, operators, literals and method calls. Expression is also used to assign value to variable. Therefore, value assigned to variable by using expression is refer as **dynamic initialization.**

For example

int x = 10, y = 20;

int sum = x + y; //dynamic initialization

Types of variable:

There are three types of variable in java:

i. **Local variable:**
A variable declared inside body of method (can be user defined or main method) is known as local variable. This type of variable is declared by defining its type. This kind of variable can be used only within that method where it is defined. Method in one class will not be aware about the existence of variable that are defined on other method.

ii. **Instance variable:**
A variable declared inside the class but outside the body of method is called instance variable. It is also known as member variable. It is also known as data of class. Only the object of such class can access it.

iii. **Static variable:**
A variable declared using static keyword is called static variable. This type of variable is initialized in class and the value of it will be fixed and cannot be changed explicitly by object. The main use of static variable is that it can be accessed directly by class name. A single copy of it is shared by all instance and the value of it cannot be changed.

Data Type:

Data type specifies type and size of value that can be stored in variable. There are two types of data type in Java: i) Primitive Data type ii) non-primitive data type

i. **Primitive Data Type:**
Primitive data type are basic data types that specifies type of value that can be stored and size of value but doesn’t contains any other addition method that perform some special operation. Java defines eight primitive types of data: byte, short, int, long, char, float, double and Boolean. Primitive data types are further divided into four group

- **Integer type:** which includes **byte, short, int and long** which are for whole valued signed number
- **Floating point type:** this includes **float and double** which represents numbers with fractional precision.
- **Character:** includes **char** which represents symbols in a character set like letter, number etc. Char type value is written inside single quote.
- **Boolean:** includes Boolean which represents true or false and mainly used in decision making

The primitive data type represents only single value (not complex type) and are defined to have explicit range and mathematical behavior. As java is strictly typed language, all the data type has fixed range. For example, int is 32-bit, byte is 8 bits

Types of Primitive data type

Integer type:

Java defines four integer types: byte, short, int and long. All of these are signed, positive and negative. Java does not support unsigned integer. The width of the integer type specifies the amount of storage it consumes. Different integer type has different width as shown in table below:

Name	Width	Range
long	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32 bits	-2147483648 to 2147483647
Short	16 bits	-32768 to 32767
byte	8 bits	-128 to 127

Byte:

The smallest integer type is byte which is a signed 8-bit type and has range from -128 to 127. Byte is useful if we are working with a stream of data from a network or file. Byte variable can be declared using byte keyword. For example, byte a=124, b=0; but byte c = 128; is illegal as it is out of range.

Short:

Short is the signed 16-bit type which ranges from -32768 to 32767. It is least use. For example, short z = 24, y=32000;

Int:

It is the most commonly used integer data type which is 32 bits in size and have range of -2147483648 to 2147483647.

Long:

Long is also a least used type which have size of 64 bit and are only used if int type is not sufficient to hold desired value. The range of long type is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

Floating Point Type:

Floating point are used when we require fractional precision i.e., required when evaluating expression that need decimal precision. Also known as real number. For example, 2.99, 50.005 calculation of square root etc. it has two types: float and double.

Name	Width	Range
Double	64	4.9e-324 to 1.8e+308
Float	32	1.4e-045 to 3.42+038

Float:

Float uses 32 bit of storage and specifies a single precision. In single precision: one bit is used to indicated sign, 8 bits is reserved for exponent and remaining 23 bits are used to represent the digits. Variable of float type are useful when fractional component is needed but don’t require a large precision. Float types are denoted by float keyword. The precision indicates how many digits can a value have after decimal point and float can have only six to seven decimal digits. For example: float low=3.5, calc;

Double:

Double type uses 64-bit storage and specifies a double precision. In single precision: one bit is used to indicated sign, 11 bits is reserved for exponent and remaining 52 bits are used to represent the digits. Double types are denoted by using double keyword. For high-speed mathematical calculation, double type is use and are also fast than float type. Double type can have precision of about 15 digits.

Characters:

Character is denoted by char and used to store single character. The character must be surrounded using single quote such as ‘a’, ‘c’ etc. char is 16 bit and uses Unicode to represent character. Unicode defines a fully international character set that can represent all of the characters found in all human languages. If we assigned some numeric value to variable of char type without using single quote then it will convert to alphabets (ASCII value) represent by that number. For example: char ‘a’, 65;

Char type can also be used as an integer type on which some arithmetic operation can be performed. For example: incrementing the value of character such as char ch = ‘a’; a++;

Booleans:

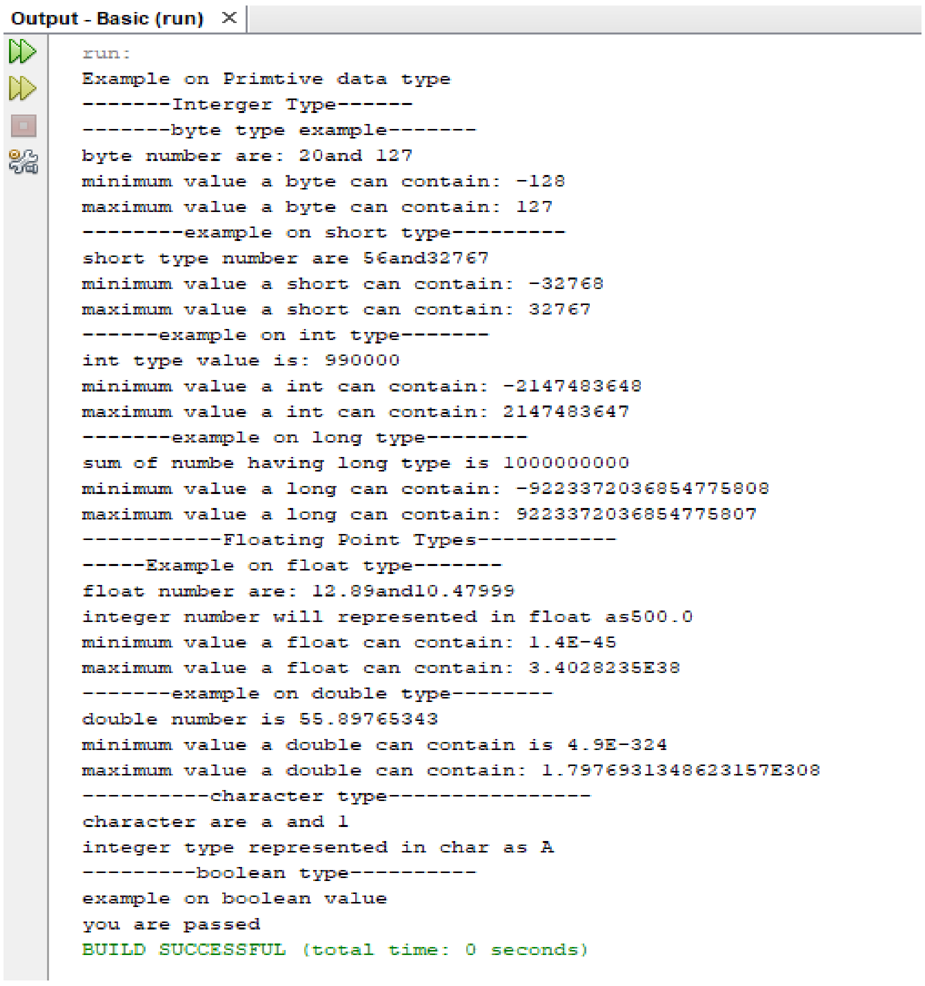
Boolean type is used for storing logical value. It is one bit and stores either true or false value. Boolean type is declared using Boolean keyword. For example: boolean result = true; Boolean type is useful while using conditional expression to check some statement and also used for comparing some values like greater than, small than etc.

Following program shows how primitive data types are declared and used:

```
package com.var;
public class Var {
   public static void main(String[] args) {
      System.out.println("Example on Primtive data type");
      System.out.println("-------Interger Type------");
      System.out.println("-------byte type example-------");
      byte x = 20, y=127;
      System.out.println("byte number are: "+x+"and "+y);
      //finding out range of byte
      System.out.println("minimum value a byte can contain: "+Byte.MIN_VALUE);
      System.out.println("maximum value a byte can contain: "+Byte.MAX_VALUE);
      System.out.println("--------example on short type---------");
      short s=56, b=32767;
      System.out.println("short type number are "+s+"and"+b);
      //finding out the range of short type
      System.out.println("minimum value a short can contain: "+Short.MIN_VALUE);
      System.out.println("maximum value a short can contain: "+Short.MAX_VALUE);
      System.out.println("------example on int type-------");
      int i = 220, j = 4500, total;
      total = i*j;
      System.out.println("int type value is: "+total);
      //finding out range
      System.out.println("minimum value a int can contain: "+Integer.MIN_VALUE);
      System.out.println("maximum value a int can contain: "+Integer.MAX_VALUE);
      System.out.println("-------example on long type--------");
      long l1 = 100000, l2 = 10000;
      long sum = l1*l2;
      System.out.println("sum of numbe having long type is "+sum);
      //finding out range
      System.out.println("minimum value a long can contain: "+Long.MIN_VALUE);
      System.out.println("maximum value a long can contain: "+Long.MAX_VALUE);
      System.out.println("-----------Floating Point Types-----------");
      System.out.println("-----Example on float type-------");
      float f1=12.89f,f2=10.47999f, f3=500;
      System.out.println("float number are: "+f1+"and"+f2);
      System.out.println("integer number will be represented in float as"+ f3);
      System.out.println("minimum value a float can contain: "+Float.MIN_VALUE);
      System.out.println("maximum value a float can contain: "+Float.MAX_VALUE);
      System.out.println("-------example on double type--------");
      double d1 = 55.89765343;
      System.out.println("double number is "+d1);
      System.out.println("minimum value a double can contain is "+Double.MIN_VALUE);
      System.out.println("maximum value a double can contain: "+Double.MAX_VALUE);
      System.out.println("----------character type----------------");
      char c1 = 'a';
      char c2 = 65;
      char c3= '1';
      System.out.println("character are "+c1+" and " +c3);
      System.out.println("integer type represented in char as "+c2);
      System.out.println("---------boolean type----------");
```

```
        boolean res = true;
        if(res) {
            System.out.println("example on boolean value");
            System.out.println("you are passed");
        }

    }
}
```

Output:


Output - Basic (run)

```
run:
Example on Primtive data type
-------Interger Type------
-------byte type example-------
byte number are: 20and 127
minimum value a byte can contain: -128
maximum value a byte can contain: 127
--------example on short type---------
short type number are 56and32767
minimum value a short can contain: -32768
maximum value a short can contain: 32767
------example on int type-------
int type value is: 990000
minimum value a int can contain: -2147483648
maximum value a int can contain: 2147483647
-------example on long type--------
sum of numbe having long type is 1000000000
minimum value a long can contain: -9223372036854775808
maximum value a long can contain: 9223372036854775807
-----------Floating Point Types-----------
-----Example on float type-------
float number are: 12.89and10.47999
integer number will represented in float as500.0
minimum value a float can contain: 1.4E-45
maximum value a float can contain: 3.4028235E38
-------example on double type--------
double number is 55.89765343
minimum value a double can contain is 4.9E-324
maximum value a double can contain: 1.7976931348623157E308
----------character type----------------
character are a and 1
integer type represented in char as A
---------boolean type----------
example on boolean value
you are passed
BUILD SUCCESSFUL (total time: 0 seconds)
```

<u>Non- Primitive Data Type:</u>

Non primitive data types are the reference type which refers to the object and are not predefined in java i.e., it is created by the programmer using the program. Non-primitive includes classes, interfaces and Arrays. Non-primitive type is used for calling method to perform some operations and have same size for all. It can also contain null value.

Literal and its types:

Literals are the constant value i.e., values that are not assigned to variables. Literals simply means something that evaluates to itself such as number (44, 55 etc.) or string (“hello”) etc. a literal can be used anywhere if the value of its type is allowed. Literals can be of any data type like int, float, double, char etc.

Literals Types:

Integer Literals:

Any whole number value are integer literals and are commonly used in most of the case. Normally, it describes the base 10 value like 20,30 ,50 etc. But integer literal can also use other two base: octal (base 8) and hexadecimal (base 16). Octal numbers are denoted in java by leading zero i.e., prefix by zero for example 0122, 0127. If we write 09 then it will result in error as octal only contains up to 7. Hexadecimal number are denoted by leading 0x or 0X. for example 0X2AB, 0x3AC etc. Binary number can also be denoted in java by leading 0b for example 0b1010, 0b1111. If an integer literal contains value more than the range of integer that it should be specified by using ‘L’ on last of the value. For example, 0x7fffffffffffffffffffL

After jdk 7, we can embed one or more underscore in an integer literal. Such underscore will be discarded once program is compiled. For example, int a = 123_45_489;

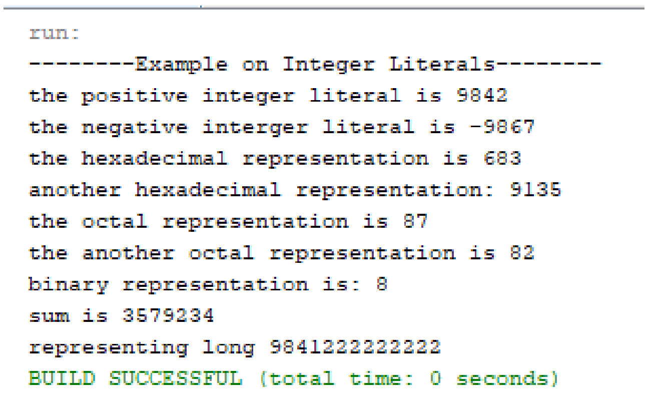
The underscore are used for separating the value like that of comma but it should not come in beginning and at end of value. Such use of underscore will be useful while storing long numeric values like telephone number, customer ID etc.

Int x = 0b1010_111_110;

Following program shows example on declaring integer literals:

```
package com.var;
public class Lit {
    public static void main(String[] args) {
        System.out.println("--------Example on Integer Literals--------");
        System.out.println("the positive integer literal is "+9842);
        System.out.println("the negative interger literal is "+ -9867);
        System.out.println("the hexadecimal representation is "+ 0x2AB);
        //hexadecimal, octal and binary can also be assigned to variable having int type
        int hex = 0x23AF;
        int oct = 0122;
        int bin = 0b1000;
        System.out.println("another hexadecimal representation: "+hex);
        System.out.println("the octal representation is "+0127);
        System.out.println("the another octal representation is "+oct);
        System.out.println("binary representation is: "+bin);
        //uses of underscore
        int z = 123_567, y=345_5667;

        int sum = z + y;

        System.out.println("sum is "+ sum); //here underscore is igmored

        System.out.println("representing long "+ 9841222222222L);

    }

}
```

Output:



```
run:
--------Example on Integer Literals--------
the positive integer literal is 9842
the negative interger literal is -9867
the hexadecimal representation is 683
another hexadecimal representation: 9135
the octal representation is 87
the another octal representation is 82
binary representation is: 8
sum is 3579234
representing long 9841222222222
BUILD SUCCESSFUL (total time: 0 seconds)
```

Floating Points Literals:

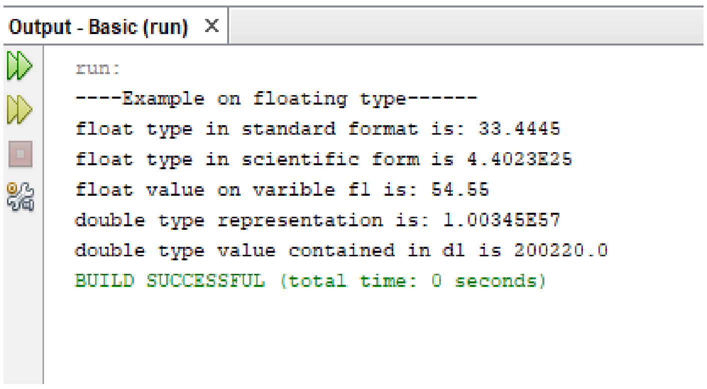
Floating points literals represents decimal value with fractional components. They can be represented in two form: standard form and scientific form. Standard form represents floating point literal as a decimal value followed by fractional points. For example, 2.5, 45.79 etc. Scientific form represents floating point literal as decimal number suffixed or followed by exponential unit (power of 10). For example: 5.33E22, 3.15E-4 etc.

To specify a floating-point literal, a constant f or F should be appended on end of literals. Floating type literals default to double precision. To use double literals, a constant d or D should be appended to the end of literals. Like in integer literal we can use underscore to separate digits on floating point literals too. In floating point literals, underscore can also be used after the decimal value. For example: 2356. 8_89_01

Following program shows example on floating point literals

```
package com.var;

public class FltLit {

    public static void main(String[] args) {

        System.out.println("----Example on floating type------");

        System.out.println("float type in standard format is: "+33.4445f);

        System.out.println("float type in scientific form is "+44.023E24f);

        //declaring on float variable

        float f1 = 54.55f;
        System.out.println("float value on varible f1 is: "+f1);
        System.out.println("double type representation is: "+ 100.345E55d);
        double d1 = 200.22E3d;
        System.out.println("double type value contained in d1 is "+d1);
    }
}
```

Output:



Output - Basic (run) ×

```
run:
----Example on floating type------
float type in standard format is: 33.4445
float type in scientific form is 4.4023E25
float value on varible f1 is: 54.55
double type representation is: 1.00345E57
double type value contained in d1 is 200220.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

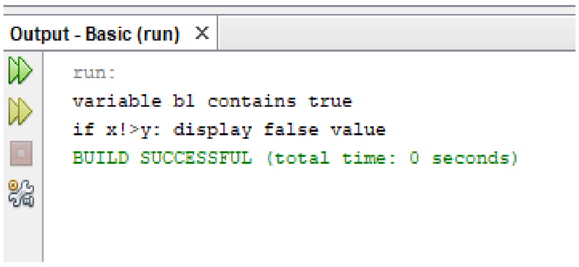
Boolean Literals:

Boolean literals are simple and contains only two logical values true and false. The value of Boolean literal is not convertible to numeric representation i.e. in java true is not equal to 1 and false is not equal to 0. Boolean value can be use only on variable declared as Boolean, while using in some expression with Boolean operator like greater than, less than etc.

Following program shows example on Boolean literal

```
package com.var;
public class BoolLit {
    public static void main (String [] args) {
        boolean b1 = true;
        System.out.println("variable b1 contains "+b1);
        //boolean literal are used in conditional operator
        int x = 27, y=100;
        if(x>y) {
            System.out.println("if x>y: display true value");
        } else {
            System.out.println("if x!>y: display false value");
        }
    }
}
```

Output:



```
run:
variable b1 contains true
if x!>y: display false value
BUILD SUCCESSFUL (total time: 0 seconds)
```

Character Literals:

Character literal use Unicode to represent character. It is 16-bit value which can be manipulated using integer operator such as addition, subtraction, increment etc. a character literal are represented using single quote. All the ASCII value can be directly represented by character literals but there are some character (double code, newline character, tab etc) that cannot be directly represented by character literals. Such character can be represented by using escape sequence. Escape character are the character which convey special meaning such as \t for tab, \n for new line etc. The combination of escape character (\ and a letter) are used to signify that the character after escape character should be treated specially. Some of the escape character and their function are described in table below:

String Literals:

String literals are specified by enclosing sequence of character with double quote. For example: "hello java porgraming", "BCA third semester". String literal should be beginning and end on same line. This literal also contains different escape character which convey special meaning. For example, "Hello\tbca": this will provide space between two words. "I\'ve done assignment": here \'ve will put single quote as string literals.

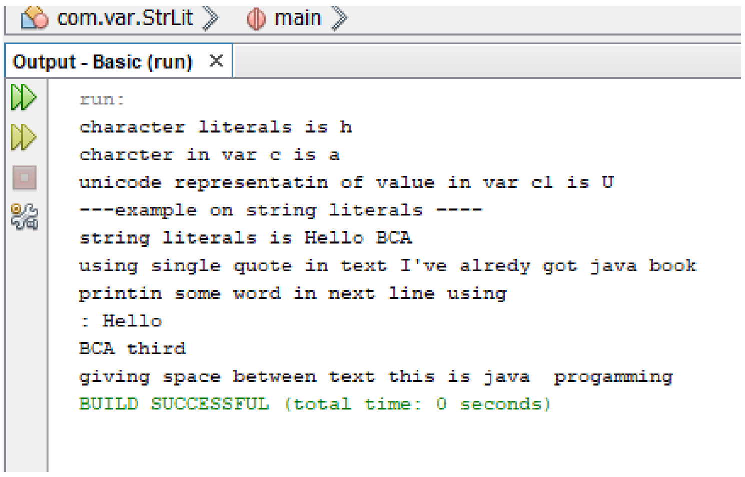
Escape Character	Function
\"	to print next character as double quote, not as a string closer
\'	To print next character as single quote, not as a string closer
\n	Print new line character like print statement
\t	To tab character (put space)
\b	Back space
\\	Print next character as a backslash not a escape character
\r	Print a carriage return (not used much)

Following program demonstrate the use of character and string literals values

```
package com.var;

public class StrLit {

   public static void main (String[] args) {

      char c = 'a', c1=85;

      System.out.println("character literals is "+ 'h');

      System.out.println("charcter in var c is "+c);

      System.out.println("unicode representatin of value in var c1 is "+c1);
```

```
        System.out.println("---example on string literals ----");
        String s1 = "Hello BCA";
        //uses of escape character
        String s2 = "I\'ve alredy got java book"; //embeding single quote in text
        String s3 = "Hello\nBCA third"; // \n means new line
        String s4 = "this is java\tprogamming"; // \t means space
        System.out.println("string literals is "+s1);
        System.out.println("using single quote in text "+s2);
        System.out.println("printin some word in next line using \n: "+s3);
        System.out.println("giving space between text "+s4);

    }
}
```

Output:



```
run:
character literals is h
charcter in var c is a
unicode representatin of value in var c1 is U
---example on string literals ----
string literals is Hello BCA
using single quote in text I've alredy got java book
printin some word in next line using
: Hello
BCA third
giving space between text this is java  progamming
BUILD SUCCESSFUL (total time: 0 seconds)
```

The Scope and Lifetime of Variable:

Till now we have used variable in main method only. But java also allows to declare variable in any block like in inside the conditional statement block, inside other methods' block, inside loop statement etc. any block of statement determines scope.

The scope of the variable determines the region of program upon which a variable is actually available for use or variable is active. In terms of instance, scope determines what objects are visible to other parts of program. In java, lifetime of variable is determined by the class and methods. Scope define by class will be covered on chapter 3.

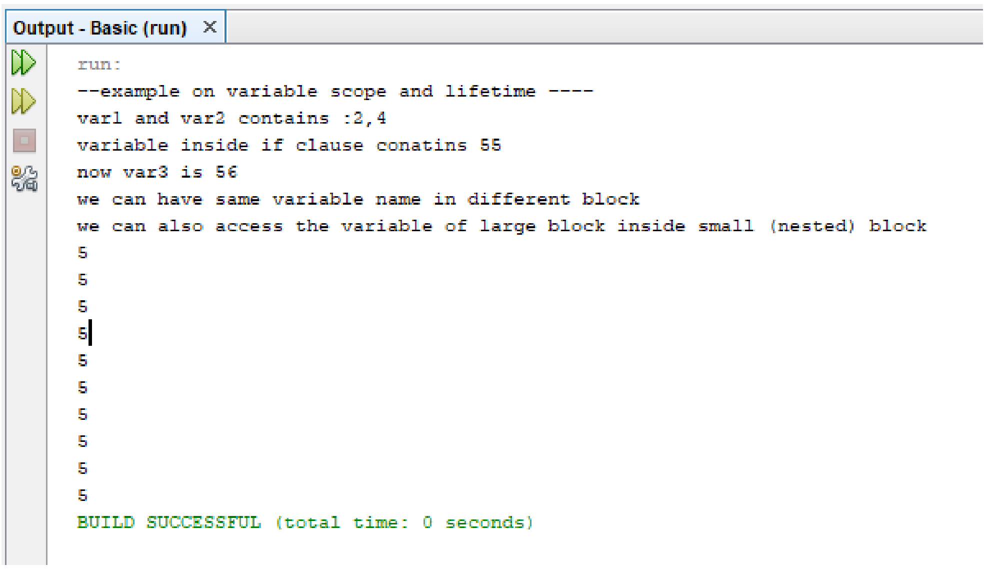
Variable declared inside a scope (block or method) are not visible to the code that are define outside the method. When we declare a variable inside a scope (method) we are protecting that variable from unauthorize access from outside the scope. Scope rule provides foundation for encapsulation. One scope (block) and be nested to another block i.e., one block inside another block. In such case variable declare on outer block is visible to inner block but reverse is not true i.e., the variable declare in inner block cannot be accessible by outer block.

Lifetime of variable refers to up to which point or up to which time a value of variable is valid i.e., value of variable will not be destroyed. The lifetime of variable is within the scope of variable i.e., once the scope of variable is gone, variable will not hold its value.

Following program demonstrate scope and life time of variable:

```
package com.var;

public class VarScope {

   public static void main(String[] args) {

      System.out.println("--example on variable scope and lifetime ----");

      //we cannont use var1 and var2 here as it is dclared on buttom;

      //System.out.println(var1+"and"+var2); error

        int var1=2, var2=4;

        System.out.println("var1 and var2 contains :"+var1+","+var2);


        if(var1 < var2){

           int var3 = 55; //variable deeclared inside another block

           //it is visible up to end of if clause and its value will be persist up to end of if clause

           System.out.println("variable inside if clause conatins "+var3);

           var3++; //increasing value to test lifetime

           System.out.println("now var3 is "+var3);

        }

        //variable var3 cannot be used here as this line is outside if clause

        //System.out.println(var3); //creates error

        System.out.println("we can have same variable name in different block");

        System.out.println("we can also access the variable of large block inside small (nested) block");

        int i, a=5;

        for(i=0;i<10;i++){

           System.out.println(a); //var declare outside block can be access inside

        }

}

}
```

<u>**Output:**</u>



```
run:
--example on variable scope and lifetime ----
var1 and var2 contains :2,4
variable inside if clause conatins 55
now var3 is 56
we can have same variable name in different block
we can also access the variable of large block inside small (nested) block
5
5
5
5
5
5
5
5
5
5
BUILD SUCCESSFUL (total time: 0 seconds)
```

<u>**Type Conversion and Casting:**</u>

Type conversion is the process of converting or assigning the value of one data type into another data type. For example it is possible to assign byte value into int variable. There are two type of type coversion: implicit type casting (automatic) and explicit type casting (forcefully).

If the value of variable of one type is converted automatically to another type whenever required then such conversionn is known as **implicti conversion**. For the automatic conversion the following condition should be met:

- Two types should be compatible
- The destination type is larger than the source type.

For example: value of byte variable can be automatically converted to int type as int is large enough to hold the all values of byte. It is also known as widening conversion: the numeric type like interger and float are compatible to each other. There will not be any automatic conversion from numeric type to char or boolean. Java can also perform implicit conversion when storing literal integer constant into variable of type byte, short, long or char.

If the user force to change the type of variable from one form to another then such casting is known as explicit casting. In java, to perform explicit conversion: desired data type name (data type to which we want to cast to) with parenthesis should be used before variable. Syntax: (target-type) value or variable. From this casting we can cast the value of large data type (int) into small type (byte). For example:

Int x = 150;

Byte b;

b = (byte) x;

here, if int contains the value that is larger than range of byte then then it will reduce modulo by byte range (divide the remainder of integer by byte range).

If floating type value is cast to integer type then truncation will occur i.e. fractional value after decimal value will be removed. For example the value 55.67 will be converted to 55.

For example:

Int x = 2.55f //produces error

Int x = (int) 2.55f; //converts 2.55f to 2;

Int a = 299;

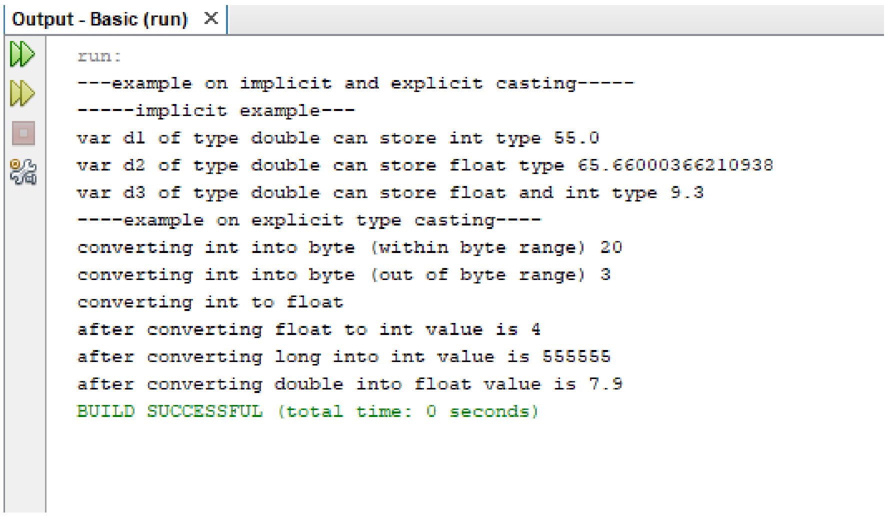
Byte b = (byte) a; // converts to byte. 299 is out of range for byte so the value of x is modulas divided by range of byte (256)

Following program provide example on type casting

```
package com.var;

public class VarCast {

   public static void main(String[] args) {

      System.out.println("---example on implicit and explicit casting-----");

      System.out.println("-----implicit example---");

      double di = 55; //here integer is converted into double type

      double d2 = 65.66f;

      double d3 = 4.3 + 5; //here 5 will convert to 5.0

      System.out.println("var d1 of type double can store int type "+di);

      System.out.println("var d2 of type double can store float type "+d2);

      System.out.println("var d3 of type double can store float and int type "+d3);

      System.out.println("----example on explicit type casting----");

      int i=20, j=259;

      byte b,b1;

      b = (byte)i;

      System.out.println("converting int into byte (within byte range) "+b);

      b1 = (byte) j; //modulo dividw with size of byte 256

      System.out.println("converting int into byte (out of byte range) "+b1);

      System.out.println("converting int to float");

      //int x= 4.5; //produce error

      int x = (int)4.9f; // data loss wil occur i.e. .5 will be removed

      int y = (int) 555555l;

      float f1 = (float) 7.9;

      System.out.println("after converting float to int value is "+x);

      System.out.println("after converting long into int value is "+y);
```

```
        System.out.println("after converting double into float value is "+f1);
    }
}
```

Output:



```
run:
---example on implicit and explicit casting-----
-----implicit example---
var d1 of type double can store int type 55.0
var d2 of type double can store float type 65.66000366210938
var d3 of type double can store float and int type 9.3
----example on explicit type casting----
converting int into byte (within byte range) 20
converting int into byte (out of byte range) 3
converting int to float
after converting float to int value is 4
after converting long into int value is 555555
after converting double into float value is 7.9
BUILD SUCCESSFUL (total time: 0 seconds)
```

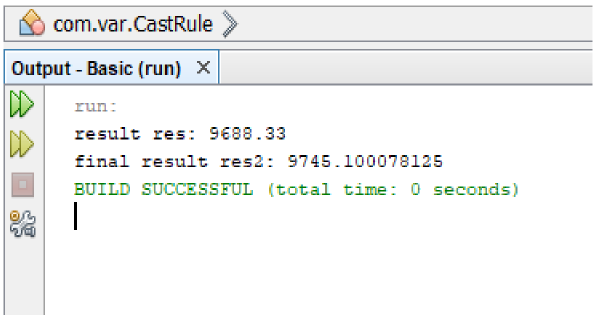
Type Casting Rule:

- First all byte, short and char value are promoted to int.
- If operand is long the all the expression is converted to long.
- If one operand is float then whole expression is converted to float.
- If any operand is double then whole expression is converted to double.

Following program will provide example on casting rule;

```
package com.var;
public class CastRule {
    public static void main (String [] args) {
        byte b = 12;
    short s = 255;
    char ch= 'c';
    int i=67;
    float f1 = 4.67f;
    double d2 = 56.77;
    //int res = b*s + i*ch - f1; //produce error as whole expression will change into float
    float res = b*s + i*ch - f1; // all exp cast into float so float type is used to store result
   // float res2 = res + d2; //produce error as whole expression will change into double
   double res2 = res +d2;
        System.out.println("result res: "+res);
        System.out.println("final result res2: "+res2);
}
}
```

Output:



com.var.CastRule

Output - Basic (run)

```
run:
result res: 9688.33
final result res2: 9745.100078125
BUILD SUCCESSFUL (total time: 0 seconds)
```

Operator:

An operator is symbol that performs some mathematical or logical manipulation. Operator are used in programs to manipulate data and variables and they usually form a part of the mathematical or logical expressions. Each operator takes on different number of operand like: **unary operator** which is used for performing operation on single operand like increment (++), decrement (- -). , **binary operator** which is used for performing operation on two operand like addition, subtraction etc, and final one is **ternary operator** which takes the form expr ? x : y which requires three operands.

Most of Java's operator are divided into four group: arithmetic, bitwise, relational and logical.

Arithmetic Operator:

Arithmetic operator is used to perform some algebraic or arithmetic operation like addition, subtraction, multiplication etc. on operand (data on which operation is going to be performed). The operand of the arithmetic operator should be of numeric type. Following are the different types of arithmetic operators:

Operator	Description	Example
+	Use for addition	x + y
-	Use for subtraction	x - y
*	Use for multiplication	x * y
/	Use for division. Gives quotient	x/y
%	Use for modulo division. Gives remainder	x %y
++	Increment	x++ or ++x
**	use for raising $x to the $yth power	x ** y
+=	Addition assignment	X+=y

-=	Subtraction assignment	x-=y
*=	Multiplication assignment	X*=y
/=	Division assignment	x/=y
%=	Modulo assignment	X%=y
--	Decrement	x- -or --x

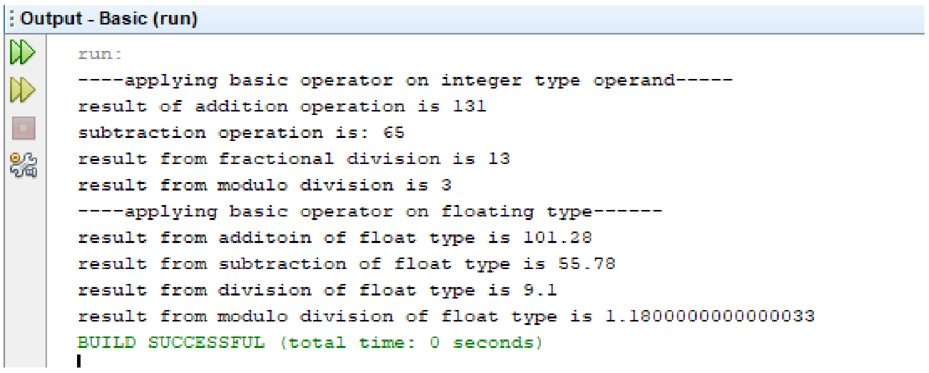
Basic Arithmetic Operator:

The basic arithmetic operator are addition, subtraction, multiplication and division. The addition operator used in single operand will just return the value of operand. The subtraction operator used in single operand will negate the single operator. The modulus operator (%) will return the remainder of a division operation.

Following program provides the example on basic arithmetic operator.

```
package com.var;

public class BasicOperator {

  public static void main(String[] args) {

    System.out.println("applying basic operator on integer type operand");

    int a = 65, b=66;

    int addition = a+b;

    int subtraction = addition - b;

    int division = a/5;

    int modDiv = division %5;

    System.out.println("result of addition operation is "+addition);

    System.out.println("subtraction operation is: "+subtraction);

    System.out.println("result from fractional division is "+division);

    System.out.println("result from modulo division is "+modDiv); //return remainder

    System.out.println("applying basic operator on floating type ");

    double d1 = 45.5, d2=55.78;

    double add = d1+d2;

    double sub = add - d1;

    double div = d1/5;

    double moddiv = sub%div;

    System.out.println("result from additoin of float type is "+add);

    System.out.println("result from subtraction of float type is "+sub);

    System.out.println("result from division of float type is "+div);

    System.out.println("result from modulo division of float type is "+moddiv);

  }}
```

Output:


Output - Basic (run)

```
run:
----applying basic operator on integer type operand-----
result of addition operation is 131
subtraction operation is: 65
result from fractional division is 13
result from modulo division is 3
----applying basic operator on floating type------
result from additoin of float type is 101.28
result from subtraction of float type is 55.78
result from division of float type is 9.1
result from modulo division of float type is 1.1800000000000033
BUILD SUCCESSFUL (total time: 0 seconds)
```

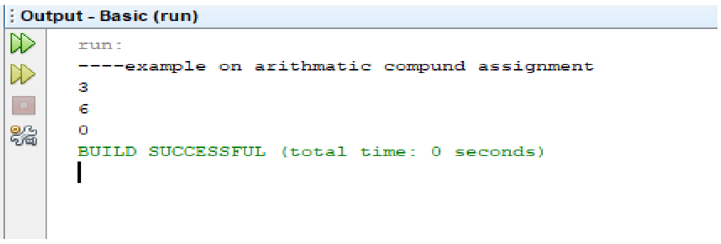
Arithmetic Compound Assignment Operators:

These are the operator used to combined arithmetic operation (addition, subtraction etc) with an assignment. For example, a+=5 (similar to a = a+5) means that the value of variable a and value 5 is added and final result is assigned on variable a. same operation goes for -=, /= and %= operator. Its general syntax is: **variable operator = expression**

Following program gives an example of arithmetic compound assignment operators.

```
package com.var;

public class CompoundOperator {

   public static void main(String[] args) {

      System.out.println("----example on arithmatic compund assignment");

      int var1 = 10,  var2 = 10, var3=5;

      var1 += var2; //additon assignment

      var2 -= 4;    //subtraction assignment

      var3*=var2;  // multiplication assignment

      var1 /=var2; //quotient division assignment

      var3%=2;     //modulo assignment

      System.out.println(var1);

      System.out.println(var2);

      System.out.println(var3);

  }

}
```

Output


Output - Basic (run)

```
run:
----example on arithmatic compund assignment
3
6
0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Increment and decrement operator:

Increment operator is use for incrementing a value of variable by 1 i.e., adds 1 to operand whereas decrement operator is used for decrementing a value of variable by 1 i.e. subtracts 1 from operand. Increment and decrement operator are unary operators and requires single variable as their operand. They are appears on two form: prefix form (operator precedes the operand) and postfix form. Following are the types of increment and decrement operator:

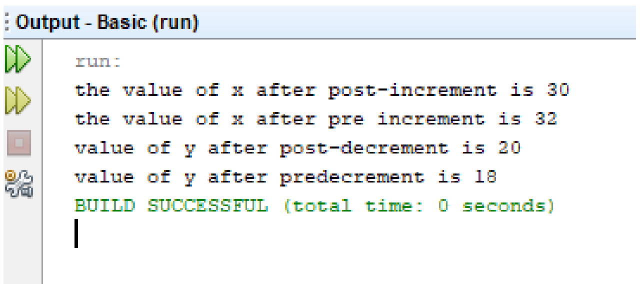
Operator	Description
++x	This operator is known as pre-increment operator which will first increment the value of $x and then expression is evaluated using new value of variable (return new value of x.
x++	This operator is known as post-increment operator which will use original value of $x and then value of variable $x is incremented by one.
--x	This operator is known as pre-decrement operator which will first decrement the value of $x and then expression is evaluated using new value of variable (return new value of x).
x--	This operator is known as post-decrement operator which will use original value of $x and then value of variable $x is decremented by one.

Following program provides an example on use of increment and decrement operator

```
package com.var;

public class IncDecOperator {

   public static void main(String[] args) {

      int x = 30;

      int y = 20;

      double d = 66.7;

      //following code will show value of x to 30 but value is incresed to 31

      System.out.println("the value of x after post-increment is "+ x++);

      /*

      following code will show value of x to 32 because value of

      $x becames 31 from first line of code. Now, pre increment will

      increase value of x from 31 to 32 and assign to x.

      */

      System.out.println("the value of x after pre increment is "+ ++x);

      /*

      follwoing code will show value of y to 20 but value has been

      decrease from 20 to 19;

      */

      System.out.println("value of y after post-decrement is "+ y--);
```

```
        /*
        following code will show value of y to 18 because value of
        y becames 19 from first line of code. Now, pre decrement will
        decrease value of $y from 19 to 18 and assign to $y.
         */
        System.out.println("value of y after predecrement is "+ --y);
    }
}
```

Output:



```
Output - Basic (run)
run:
the value of x after post-increment is 30
the value of x after pre increment is 32
value of y after post-decrement is 20
value of y after predecrement is 18
BUILD SUCCESSFUL (total time: 0 seconds)
```

Comparison or Relational Operator:

Relational operator is used for comparing two values i.e., determines relationship that one operand has to the other. The determine equality (== and! = operator) and ordering (>, <, <=, >=). The result from relational operator is either true or false. For example, 10>5 is true but 5>10 is false. The relational operator is mostly used in conditional statement like if, if…else etc. and various loop statement. Equality operator like == and! = can be applied to integer, floating point, character and Boolean. Only numeric type (integer, floating point and character) can be compared using the ordering operator like >, <, <=, >=.

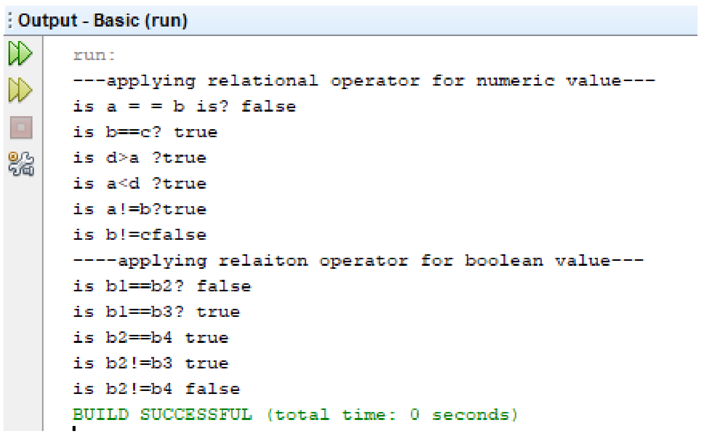
Following are the different relational operators:

Operator	Example	Description
= =	x = = y	This equal operator will return true if value of x is equal to y
>	x > y	This greater than operator will return true if value of x is greater than value of y. Otherwise, false
<	x < y	This less than operator will return true if value of x is less than value of y.
>=	x >= y	This greater or equal to operator will return true if and only if value of x is greater or equal to value of y otherwise false
<=	x <= y	This greater or equal to operator will return true if and only if value of x is less or equal to value of y otherwise false
!=	x! =y	This not equal to operator will return true if the value of x is not equal to value of y.

Following program demonstrate the example of relational operator:

```
package com.var;
public class RelationalOp {
    public static void main(String[] args) {
        System.out.println("---applying relational operator for numeric value---");
      int a = 20, b=30, c=30,d=40;
        System.out.println("is a = = b is? "+ (a==b));
        System.out.println("is b==c? "+(b==c));
        System.out.println("is d>a ?"+ (d>a));
        System.out.println("is a<d ?"+(a<d));
        System.out.println("is a!=b?"+(a!=b));
        System.out.println("is b!=c"+(b!=c));
        System.out.println("----applying relaiton operator for boolean value---");
        // we can only use equality operator (!= and ==)
        boolean b1=true, b2=false, b3=true,b4=false;
        System.out.println("is b1==b2? "+(b1==b2));
        System.out.println("is b1==b3? "+(b1==b3));
        System.out.println("is b2==b4 "+(b2==b4));//as false is equal to false it will give ans true (correct)
        System.out.println("is b2!=b3 "+(b2!=b3));
        System.out.println("is b2!=b4 "+(b2!=b4));


    }
}
```

Output:



```
run:
---applying relational operator for numeric value---
is a = = b is? false
is b==c? true
is d>a ?true
is a<d ?true
is a!=b?true
is b!=cfalse
----applying relaiton operator for boolean value---
is b1==b2? false
is b1==b3? true
is b2==b4 true
is b2!=b3 true
is b2!=b4 false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Boolean Logical Operator:

Logical operator is used for conditional statement which produce true or false based on the condition. Boolean operator operates only on Boolean operand. All of the binary logical operator combines two Boolean values to form a resultant Boolean value.

Operator	Example	Result
&	true & false	Refer to logical AND and return true if both value of variable are true otherwise false. It evaluates both left hand and right-hand operand to produce final result
|	True | false	Refer to logical OR and return true if any one operand is true i.e., return false if both operands are false. It evaluates both left hand and right-hand operand to produce final result
^	True ^false, true^true	Refers to logical XOR operator and return true if the value of both operands is different (either true ^false, false true) and returns false if the value of both operands is same (either true ^true, false false)
||	True || false, true||true	Refer to short circuit OR and its operation is same as logical or (|) but difference is that it produces final result based on left hand (first) operand
&&	True&&false, true&&true	Refers to short circuit AND and its operation is same as logical and (&) but difference is that it produces final result based on left hand (first) operand
!	!true, !x	Refer to logical unary not (applied on single operand) and it inverts the value of variable. i.e. true will be changed to false and false will be changed to true
?:	x>y?x:y	Refers to ternary operator and its operation is same like if then else
&=	A&=b	Refers to AND assignment. This operator first compare two variable and result is append to left variable
|=	A|=b	Refers to AND assignment. This operator first compare two variable and result is append to left variable

The difference between logical operator (&, |) and short-circuit operator (&&, ||) is that logical operator evaluates both the operand always and produce result but short circuit operator produces result based on first operand and it will evaluate second operand when it is necessary.

For example:

A	B	A|B	A&B	A^B	! A	!B
True	True	True	True	False	False	False
True	False	True	False	True	False	True
False	True	False	False	True	True	False
False	False	False	false	false	true	true

<u>Following program shows the example on Boolean logical operator</u>

<u>Bitwise Operator</u>

Bitwise operator is type of operator which perform some logical operation on binary data (in each bit). Bitwise operator can be applied to integer type like int, short, long, char and byte but the value will be changed to binary before performing operation. Java defines following bitwise operator:

Operator	Example	Description
~	~var1	Bitwise unary not or complement operator which will invert the bit value. For example, 1 will convert to 0 and 0 is convert to 1
&	Var1 & var2	Bitwise AND which will perform and operation on each bit. It will produce 1 if the both operands are 1 otherwise 0
|	Var1 | var2	Bitwise OR which will perform OR operation on operand. It will produce 1 if any one operand is on. 0 is produce if both operand is 0
^	Var1 ^ var2	Exclusive OR operator which perform XOR operation on operand. It produces 1 if operands are different (1, 0 or 0,1). 0 is produce if both operands are same (1,1 or 0,0)
>>	Syntax = value >> num Var1>>2 means that the bits contained in var1 should be shift 2 times to the right	Right shift operator shifts the bit value in right direction for specified number of times.
<<	Syntax = value << num Var1<<2 means that the bits contained in var1 should be shift 2 times to the left	Left shift operator shifts all the bit value to the left for specified number of times.
>>>	Var1>>>2	Right shift with zero fill.
&=	Var1&=var2	Bitwise AND assignment.
|=	Var1|=var2	Bitwise OR assignment
^=	Var1^=var2	Bitwise xor assignment
>>=	Var1>>=var2	Bitwise shift right assignment
>>>=	Var1>>>=var2	Shift right zero fill assignment
<<=	Var1<<=var2	Shift left assignment

For bitwise operation: each value is converted to binary number and operation is perform in each binary value.

For byte: value is presented using 8 bits for example

Byte b1 = 8;

It is represented in 8-bit representation: 0000 1000

For short: value is presented using 16 bits. For example:

Short s1 = 10;

It is represented as 16 bits so: 10 = 0000 0000 0000 1010

For int: value is presented using 32 bits

Int i1 = 12;

It is represented as 32 bits so: 12 = 0000 0000 0000 ……… 1100

Negative number are represent using 2's complement:

byte = -4;

first change the value to positive number representation:

4 = 0000 0100

After this find complement of 4

4 = 1111 1011

Then add 1:

```
 1111 1011
         +1
___________
1111   110 0 => -4
```

Now let's consider each bitwise operator in detail

<u>The bitwise AND Operator:</u>

Produce 1 if both the operator are 1 other wise 0 (if one of the operand is 0). Its symbol is &.

Byte b1 = 10, b2 = 15;

Representing 10 and 15 in binary and performing bitwise AND operation

10 =>	0	0	0	0	1	0	1	0
15 =>	0	0	0	0	1	1	1	1
&	0	0	0	0	1	0	1	0

b1 & b2 = 00001010

b1 & b2 = 10

<u>The bitwise OR operator:</u>

Produces 1 if either one of the bits in the operand is 1. It produces 0 if both operands are 0.

Byte b1 = 10, b2 = 15;

Representing 10 and 15 in binary and performing bitwise AND operation

10 =>	0	0	0	0	1	0	1	0
15 =>	0	0	0	0	1	1	1	1
|	0	0	0	0	1	1	1	1

B1 | b2 = 00001111

B1 |b2 = 15

<u>The bitwise XOR:</u>

Its symbol is ^ and produce 1 if each operand is of different types like 1,0 or 0,1. It produce false if both operands are same such as 1,1 or 0,0.

Byte b1 = 10, b2 = 15;

Representing 10 and 15 in binary and performing bitwise AND operation

10 =>	0	0	0	0	1	0	1	0
15 =>	0	0	0	0	1	1	1	1
^	0	0	0	0	0	1	0	1

B1^b2 = 00000101

B1^b2 = 5

<u>The bitwise NOT or complement:</u>

Its symbol is '~' and inverts all the bits of operand such as 1 will be convert to 0 and 0 will be convert to 1.

Byte b1 = 10, b2 = 15;

Representing 10 and 15 in binary and performing bitwise AND operation

10 =>	0	0	0	0	1	0	1	0
~	1	1	1	1	0	1	0	1

<u>Bitwise Left Shift (important for exam)</u>

The left shift operator is denoted by '<<' symbol and shift all the bit to the left according to specified number. its syntax is: value << number. the number specifies how many bit to shift left. If number contains 2 then the bit should be shifted 2 times. The high order (bit on left hand sides) is shifted out and the required number of zero is filled on right side. For example: if 2 bits is shifted out from left side then right side should be filled with two zeros.

<u>For example:</u>

Byte b1 = 5;

Byte res = (byte) (b4 << 2) means that the bits of value of b4 i.e., 5 have to be shift to left 2 times.

Solution:

First convert the value of b4 i.e. 5 into binary we get:

5 = 0000 0101;

As we have to perform left shift so the high order bit from left side will be shifted out and required number of zero is filled on right side:

5<<2 = 0000 0101;

= 00 0101 00 (convert into decimal)

5<<2 = 20

Bitwise left shift for negative number:

Byte b5 = -5;

Byte res1 = (byte) (b5<<2)

For negative number, first find 2’s complement of positive number. in our example we will find 2’s complement of positive 5.

5:	0	0	0	0	0	1	0	1
1’s complement	1	1	1	1	1	0	1	0
Add 1							+	1
-5 =	1	1	1	1	1	0	1	1

Therefore, -5 is represented as 11111011

Now finding -5<<2

-5<<2 = 1111 1011 (two 1 from left side will be removed)

-5<<2 = 11 1011 00 (two zeros are added in right side)

Here, High order bit (bit in left side) is 1 which means negative number. so we have to find 2’s complement for 11 1011 00

-5<<2:	1	1	1	0	1	1	0	0
1st complement	0	0	0	1	0	0	1	1
Add 1							+	1
-5<<2	0	0	0	1	0	1	0	0

Therefore -5<<2 = 00010100 (convert into decimal)

-5<<2 = -20

Bitwise shift right operator:

The right shift operator is denoted by '>>' symbol and shift all the bit to the right according to specified number. its syntax is: **value** >> **number**. the **number** specifies how many bit to shift right. If number contains 2 then the bit should be shifted 2 times. The low order (bit on right hand sides) is shifted out. after shifting out the bit, the required number of 0 is placed on left side if the number is positive and required number of 1 is placed on left side if number is negative. For example: if 2 bits is shifted out from right side then left side should be filled with two zeros if the number is positive and if number is negative then two one should be filled.

<u>Example of bitwise shift right operator for positive number</u>

Byte b1 = 5;

Byte b2 = 2;

Byte res = (byte) (b1>>b2) means that the bits of value of b1 i.e., 5 have to be shift to right 2 times.

Solution:

First convert the value of b1 i.e. 5 into binary we get:

5 = 0000 0101;

As we have to perform right shift so the low order bit from right side will be shifted out and required number of zero is filled on left side because 5 is positive number.

5>>2 = 0000 0101; (two bit from right is shifted i.e. 01 shifted out)

= 00 0000 10 (two zero is placed on left side as 5 is positive number)

5<<2 = 2

<u>Example of bitwise shift right operator for negative number</u>

Byte b5 = -5;

Byte res1 = (byte) (b5>>2)

For negative number, first find 2's complement of positive number. in our example we will find 2's complement of positive 5.

5:	0	0	0	0	0	1	0	1
1's complement	1	1	1	1	1	0	1	0
Add 1							+	1
-5 =	1	1	1	1	1	0	1	1

Therefore, -5 is represented as 11111011

Now finding -5>>2

-5>>2 = 1111 1011 (two 1 from right side will be removed)

-5<<2 = 11 1111 01(two one are added in left side because -5 is negative number)

Here, High order bit (bit in left side) is 1 which means negative number. so we have to find 2's complement for 11 1111 01

-5>>2:	1	1	1	1	1	1	0	1
1st complement	0	0	0	0	0	0	1	0
Add 1							+	1
-5>>2	0	0	0	0	0	0	1	1

Therefore -5<<2 = 00000011 (convert into decimal)

-5<<2 = -3

Control Statement:

Control statement is used to perform a certain action based on certain condition and controls the flow of execution. In java, control statements are categorized into three form: selection, iteration and jump statement.

Selection Statement:

The selection statement allows to control the flow of program's execution based on condition. It includes 'if' and 'switch' statement.

If statement:

It is basically a two-way decision-making statement and used with conjunction with an expression in which some action is performed if one condition is true.

It takes the form: **if (test expression or condition) {code to execute if condition is true};**

Here condition is any statement that return Boolean value: true or false. First the expression inside small bracket () is evaluated and depending on whether the value of expression (relation or condition) is 'true' or 'false', it transfer the control to a particular statement which is placed inside middle bracket { }. If the condition is false then control is transfer outside of curly bracket. If there is only single statement to execute then middle braces can be ignored.

For example:

```
Int marks = 50;

If(marks >=20){

System.out.println("you are passed"); //if condition is matched this will execute

}
```

The if else statement:

The if … else statement is an extension of the simple if statement in which code inside middle braces of if part will be executed if condition is true and if false then code inside else will be executed. It takes the form:

If (test condition) {

True block statement or code to be executed if condition is true;

}

else{

False block statement or code to be executed if condition is false;

}

For example:

```
Int marks = 50;

If(marks >=20){
```

```
System.out.println(“you are passed”);
}
else{
System.out.println(“you are  failed”);
}
```

<u>The if … elseif ladder or if..elseif..else statement:</u>

It is a chain of if in which the statement associated with each else is an if and executes different codes based on the condition. If the condition of if statement is false then the control is passed to else if and if the condition of if and all else if is false then control will be passed to else statement and code of else is executed. It takes the form:

if (test condition) {

Code to be executed if condition is true;

}

elseif (test condition){

Code to be executed if the condition of if statement is false and condition of elseif is true;

}

else{

Code to be executed if the condition of both if and else if statement is false;

}

<u>For example:</u>

```
int marks = 50;
if(marks >60){
System.out.println(“you are passed with high grades”);
}
else if((marks >30 && marks <=60){
      echo "you are passed with lower grades";
   }else{
      echo "you are failed";
   }
```

<u>The nested if statement:</u>

Nested if statement is an if statement which contains another if statement inside its block. When we declare nested if statement then we have to remember that an else statement always refers to the nearest if statement.

```
Int a = 10, b=15, e = 20, d=40;

If ( a == 15){

If(b<20){

a = e;

}else{ //this else refers to if (b<20)

        A = d;

}

}else{ //this else refer to if(a == 15)

a = f;

}
```

<u>The switch statement:</u>

The switch statement is used when one variable or the result of expression can have multiple value and each of which should be trigger a different activity. The switch statement tests the value of a given variable or expression against the list of case values and when a match is found a block of statements associated with that case is executed. It takes the form:

```
switch (expression)

{

        case value-1:

                Block-1 i.e. code to be executed if expression = value-1

                Break;

        case value-2:

                Block-2 i.e., code to be executed if expression = value-2

                Break;

…….

        default:

//Default block i.e. code to be executed if expression is different from all cases

        break;

}
```

Expression must be of type byte, short, int, char or an enumeration. With JDK7, expression can be also of type String. Each value specified in the case should be unique constant expression. The type of value in case must be compatible with the type of expression.

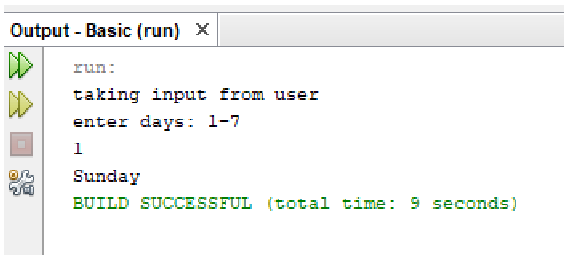
Here, when switch is executed, the value of the expression is compared against the case value-1, value-2 and so on. If a value of expression matched with value of any case, then the block of statement that follows the matched case are executed. The break statement signals the end of particular case and transfer the control out of the switch block. The default statement is executed if the value of expression does not match with any of the case values.

Example on switch:

Reading a number from user and pointing a day. (1 = Sunday, 2 = Monday … 7= Saturday)

```
package basic;

import java.util.Scanner;

public class SwitchDemo {

    public static void main(String[] args) {

        System.out.println("taking input from user");

        Scanner sc = new Scanner(System.in); //taking input from user

        System.out.println("enter days: 1-7");

        int a = sc.nextInt();

        switch(a){

            case 1:

                System.out.println("Sunday");

                break;

            case 2:

                System.out.println("Monday");

                break;

            case 3:

                System.out.println("Tuesday");

                break;

            case 4:

                System.out.println("Wednesday");

                break;

            case 5:
                System.out.println("Thursday");
                break;
            case 6:
                System.out.println("Friday");
                break;
            case 7:
                System.out.println("Saturday");
                break;
            default:
                System.out.println("Invalid input");
                break;
        }
    }
}
```

Output:



```
run:
taking input from user
enter days: 1-7
1
Sunday
BUILD SUCCESSFUL (total time: 9 seconds)
```

Nested Switch Statement:

Switch can be used as a part of statement for outer switch i.e switch statement inside another switch statement. This is called nested switch statement:

Syntax:

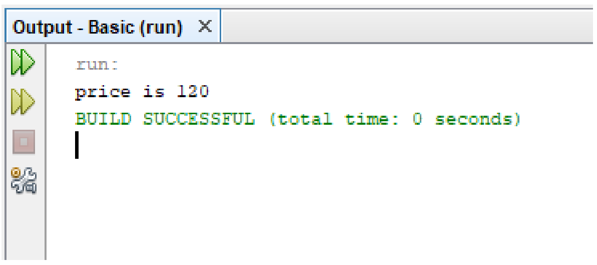
```
switch (expression) {//outer switch
case value-1:
            Block-1 i.e. code to be executed if expression = value-1
            Break;
case value-2:
            Switch (expression) {//inner switch
                case value-1:
                        Block-1 i.e. code to be executed if expression = value-1
                        Break;
                  case value-2:
                        Block-2 i.e. code to be executed if expression = value-2
                        Break;
                  Default:
                        block i.e. code to be executed if expression is different from all
                        cases
                        break;
                        }
break;
case value-3:
Block-3 i.e. code to be executed if expression = value-3
break;
Default:
block i.e. code to be executed if expression is different from all cases
break;
}
```

Following program demonstrate use of nested switch: in this program first menu is inserted whether it is pizza or momo. If menu is pizza then price for particular flavor is displayed and if menu is momo then price for particular subitem of momo is displayed

```
package basic;
public class NesSwichDemo {
    public static void main(String[] args) {
        String menu = "momo";
        String type = "buff";
        switch (menu){
            case "pizza":
                System.out.println("choose flavour for pizza");
                switch (type){
                    case "cheese":
```

```
            System.out.println("price is 100");
            break;
          case "double-cheese":
            System.out.println("price is 200 ");
            break;
          case "chicken":
            System.out.println("pice is 400");
            break;
          default:
            System.out.println("not available");
            break;


        }
        break;
      case "momo":
        switch (type){
          case "buff":
            System.out.println("price is 120");
            break;
          case "chicken":
            System.out.println("price is 340");
            break;
          default:
            System.out.println("not available");
            break;
        }
       break;
      default:
        System.out.println("main item not available");
        break;
        }
    }
}
```

Output:


Output - Basic (run)

```
run:
price is 120
BUILD SUCCESSFUL (total time: 0 seconds)
```

Iteration Statement:

Iteration statement are used to reputedly executes the same set of instruction until a termination condition is met. Iteration statement are also referred as loops and are used if we need to execute same block of code for specified number of times. Iteration consists of two segment, one tis the body of the loop and other one is the control statement that test the condition and directs the repeated execution of statement contained in the body of loop. In looping process, following steps are include:

- Setting and initialization of a condition variable
- Execution of the statement in the loop
- Test for a specified value of the condition variable for execution of the loop.
- Incrementing or updating the condition variable.

Java consist of following loop:

While loop:

While loop is used to execute a block of code as long as test condition is true. It takes the form:

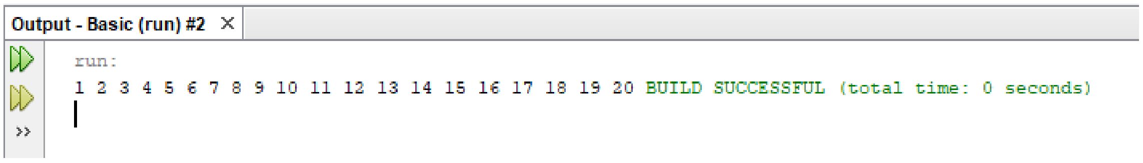
```
while (test-condtion){
    ///body of loop
}
```

First test-condition is evaluated (test-condition can be any Boolean expression) and if the condition is true then body of the loop is executed. After execution, once again test condition is evaluated and if it is true again the body of loop is executed. The same process is repeated until the test-condition becomes false and control is transfer out of the loop.

Example of while loop: printing 1 to 20 using while loop

```
package basic;
public class WhileDemo {
   public static void main(String[] args) {
        int i = 1;
        while(i<=20){
           System.out.print(i+" ");
           i++;
        }
    }
}
```

Output:



Output - Basic (run) #2

```
run:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 BUILD SUCCESSFUL (total time: 0 seconds)
```

<u>Do-While loop:</u>

Do while loop will execute a block of code or body of loop first then only test condition is evaluated i.e. it will execute a block of code at least one time. It takes the form:

```
do {
//Body of loop
}
while (test-condition);
```

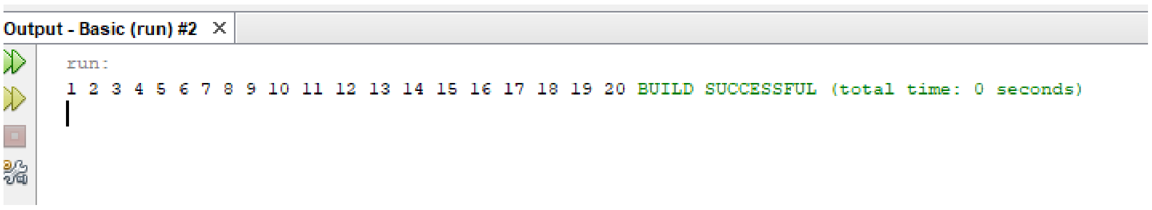
Here, on reaching the do statement, first the body of loop is evaluated. Then, the test condition is evaluated. If the condition is true, the body of loop will be evaluated once again. Same process will be continued until the condition becomes false. After condition is false, the control goes out of the loop.

For example: printing 1 to 20 using do while loop:

```
package basic;
public class DowhileDemo {
    public static void main(String[] args) {
        int i = 1;
        do{
           System.out.print(i+" ");
           i++;
        }while(i<=20);
    }
```

}

Output:



```
run:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 BUILD SUCCESSFUL (total time: 0 seconds)
```

For loop:

for loop combines the abilities to set up variables as loop is enter, test for condition while iterating loops and modify variables after each iteration. For loop is used if we know how many times the script should run. It takes the form:

for (initialization; test-condition; increment counter) {

body of the loop;

}

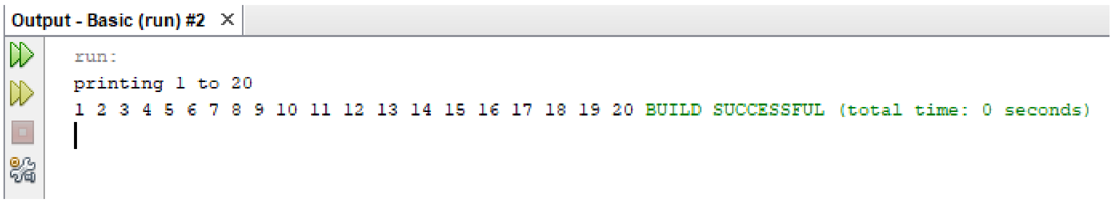
First initialization of the control variable is done using assignment operator like i = 1. After this, value of control variable is checked with test condition. If the condition is true, the body of the loop will be executed otherwise control will be transfer out of the loop. After evaluating last statement of the body of loop the control is transfer back to for statement where the value of control variable will be increment or decrement by using assignment or increment, decrement operator. For example, i=i+1 or i++.

When the initialization variable is declared in for loop then the scope of that variable ends when the for-statement end. If we need to use loop control (initialization variable) elsewhere in program then it should be declared outside of loop.

For example: program to print 1 to 20 using for loop

```
package basic;

public class ForDemo {

    public static void main(String[] args) {

        System.out.println("printing 1 to 20");

        for(int i = 1;i<=20;i++){

            System.out.print(i+" ");

        }

    }

}
```

Output:



```
run:
printing 1 to 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 BUILD SUCCESSFUL (total time: 0 seconds)
```

Jump Statement:

Jumps statement are used to skip a part of the body of the loop under certain condition. Java supports three jump statement: break, continue and return. These statement transfer control to another part of program if particular condition is met.

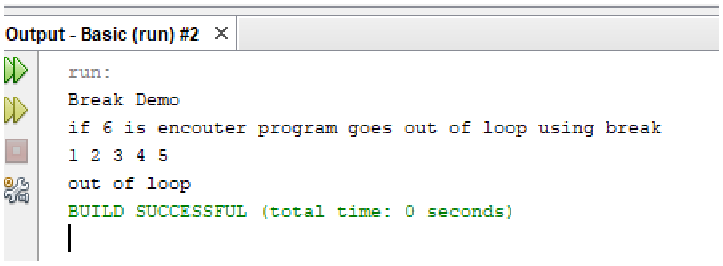
Break statement is used to jump out of loope i.e it terminates whole iteration of loop. Continue statement causes loop to be continued with next iteration after skipping any statement in between i.e. it skips rest of the body of loop and control goes to loop statement. The continue statement break only one iteration if condition occurs whereas break skips whole iteration.

Example of break:

Program that terminates loop if var I ==6;

```
package basic;

public class BreakDemo {

    public static void main(String[] args) {

        System.out.println("Break Demo");

        System.out.println("if 6 is encouter program goes out of loop using break");

        for (int i = 1; i<=10; i++) {

            if(i==6){

                break;

            }

            System.out.print(i+" ");

        }

        System.out.println("\n out of loop");

    }

}
```

Output:



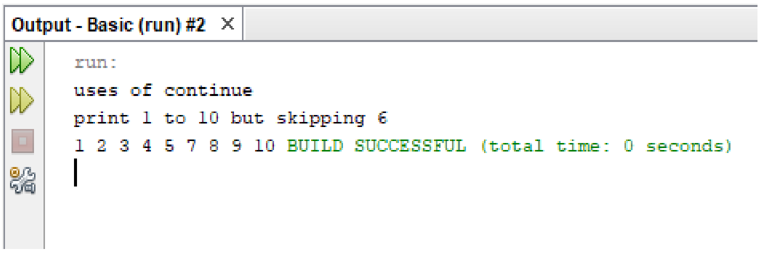
```
run:
Break Demo
if 6 is encouter program goes out of loop using break
1 2 3 4 5
out of loop
BUILD SUCCESSFUL (total time: 0 seconds)
```

Example of Continue:

Program to print 1 to 10 excluding 6

```
package basic;
public class ContinueDemo {
    public static void main(String[] args) {
        System.out.println("uses of continue");
        System.out.println("print 1 to 10 but skipping 6");
        for(int i = 1; i<=10; i++){
            if(i==6){
                continue;
            }
            System.out.println(i+" ");
        }
    }
}
```

<u>Output:</u>



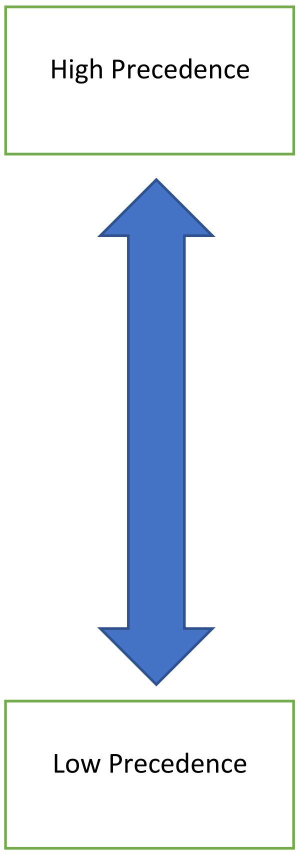
```
run:
uses of continue
print 1 to 10 but skipping 6
1 2 3 4 5 7 8 9 10 BUILD SUCCESSFUL (total time: 0 seconds)
```

<u>Operator Precedence:</u>

Precedence of operator specify which expression or which operator to execute first while evaluating any expression i.e., which operator to give highest priority than other. Associativity determines in which direction the specific operator need to be evaluated if the operator have same precedence. Operator with equal precedence and are non-associative cannot be used next to each other. For example, 1<3<4.

Java have following precedence and associativity

Operator	Associativity
++ (postfix), -- (postfix)	Left to right
++(prefix), -- (prefix). +(unary), -(unary) , ~, !	Right to left
*(multiply), / (division), % (remainder division)	Left to right
+ (addition), - (subtraction)	Left to right
<<(left shift). >>(right shift), >>>	Left to right
< (less than), > (greater than), <= (small or equal), >= (greater or equal), instance of	Left to right
= = (equality), = (non equality)	Left to right
& (bitwise)	Left to right
^ (bitwise xor)	Left to right
| (bitwise or)	Left to right
&& (logical AND)	Left to right
|| (logical OR)	Left to right
?: (ternary)	Right to left
= (assignment)	Right to left



<u>Example on Operator Precedence:</u>

Consider following expression:

Int calc = 5+4*6/3+8-3;

First, +, *, / and – have more precedence over = so first expression will be evaluated. On expression, * and / has more precedence over + and – so either * or / will be evaluated first than + and -. As the precedence of / and * are same its associativity should be considered. Associativity of / and * is from left so in left side there is * in our expression so first 4*6 will be evaluated then the result from it will be divide from 3 then added with 5 and 8 and finally the result will be subtracted from 3 and assigned to variable cal.

int cal = 5+ 24/3+8-3; //multiply is evaluated first as associative is left to right

int cal = 5+8+8-3 // divide is evaluated as it has more precedence than + and –

int cal = 13+8-3 // 5+8 is evaluated as associative is from left to right

int cal = 21-3; //13 + 8 is evaluates as associative is from left to right

int cal = 18 // subtract is evaluated at last.

Note:

Further examples are done in class, so refer your class note

For any query: email at sujeshmanandhar1996@gmail.com