Chapter 4:

Java Programming Fundamentals

Informatics Practices

Class XI (CBSE Board)

Revised as per CBSE Curriculum 2015

"Open Teaching-Learning Material"

Visit www.ip4you.blogspot.com for more....

Authored By:- Rajesh Kumar Mishra, PGT (Comp.Sc.)
Kendriya Vidyalaya Upper Camp, Dehradun (Uttarakhand)
e-mail : rkmalld@gmail.com

Learning Objectives

In this presentation, you will learn about-

- Java Character Set & Tokens
- Keywords & Identifiers
- Naming Rules & Conventions for Identifiers
- Literals in Java
- Data Types in Java
- Concept of Variables
- Operators in Java
- Java Expressions
- Data type Conversion in Java
- Developing Simple Applications.

JAVA Character Set & Tokens

- Character set is a set of valid characters that a language can recognize. It may be any letter, digit or any symbol or sign. For e.g. English language supports a-z, A-Z, 0-9 . ? ! etc.
- JAVA uses 2-Byte UNICODE character set, which supports almost all characters of all languages like English, Arbic, Chinese and so on. This makes Java a 'Multi-lingual' programming language.
- The smallest individual unit in a program is known as **Token**. It may be any word, symbols or punctuation mark etc.
- Following types of tokens used in Java-
  - Keywords
  - Identifiers
  - Literals
  - Punctuators (; [] etc.)
  - Operators (+,-,/,*, =, == etc.)

Keywords & Identifiers

- **Keywords** are the reserve words for the Java, which have some special meaning to the Java compiler. In general, keywords are Java command, literals or operators.
- Commonly used keywords are-

  char, long, for, case, if, double, int, short, void, main, while , new , true, false etc.
- **Identifiers** are user given name to **variables**, **objects**, **classes**, **swing controls** and **methods** etc. Identifiers as name suggests, identifies some user defined entities. For example name of control like jTextField1 or BtnOK etc.

Java is case sensitive Language, so keywords and identifiers should written as stated or declared.

Naming Rules for Identifiers in Java

- When we declare any identifiers like variable, class and methods etc., the following rules must be followed while naming an identifier.
  - Identifiers may have alphabets, digits and dollar ($), underscore (_) sign.
  - They must not be Java keywords.
  - They must not begin with digit.
  - They can be of any length.
  - They are Case Sensitive ie. Age is different from age.
- Example of **Valid** identifiers-

  MyFile, Date9_7_7, z2t09, A_2_Z, $1_to_100, _chk etc.
- Example of **Invalid** identifiers-

  Date-RAC, 29abc, My.File, break, for

Literals in Java

- Literals or constants are data items that have fixed data value.
- Java allows following types of literals-
  - **Integer Literals**

    Any integer number without decimal.
  - **Floating Literals**

    Any fractional number with decimal.
  - **Boolean Literals**

    Two-state values like True or false
  - **Character Literals**

    Any single character enclosed within single quotes.
  - **String Literals**

    Group of characters enclosed within double quotes.
  - **The null literals**

    Special value indicate ‘nothing’.

Integer Literals

- An integer constant or literals is a whole number.
  - It must have at least one digit.
  - It must not contain decimal point (.) and commas (,).
  - It may contain + or – sign. A number without sign assumed to be positive.
  - **L** or **U** suffix can used to represent **long** and **unsigned** integer literals respectively.
- Java allows three types of integer literals -
  - **Decimal Integer Literals** (Base 10)

    e.g. 1234, 41, +97, -17 etc.
  - **Octal Integer Literals** (Base 8)

    e.g. 010, 014 (Octal must start with 0)
  - **Hexadecimal Integer Literals** (Base 16)

    e.g. 0xC, 0xab (Hexadecimal numbers must starts with 0x)

Floating or Real Literals

- A floating literals are fractional numbers having at least one digit before and after decimal point with + or – sign.

  The following are valid real numbers-

  2.0, 17.5, -13.0, -0.00626

  The following are invalid real numbers-

  7, 7. , +17/2, 17,250.26 etc.

- A real literals may also be represented in Exponent form having Mantissa and exponent with base 10 (E). Mantissa may be a proper real numbers while exponent must be integer.

  The following are valid real in exponent form-

  152E05, 1.52E07, 0.152E08, -0.12E-3, 1.5E+8

  The following are invalid real exponent numbers-

  172.E5, 1.7E, 0.17E2.3, 17,22E05, .25E-7

Other Literals

- The **Boolean Literals** represents either TRUE or FALSE.
- A **null literals** indicates nothing. It always null type.
- **Character Literals** must contain one character enclosed in single quotation mark e.g. ‘a’, ‘%’ , ‘9’ etc.

  Java also allows some control characters which starts with Escape sequence (\). Some Escape Sequence character literals are-

  \a (alert), **\n** (new line), **\t** (Horizontal tab),
  \v (vertical tab), **** (back slash), **\’** (single quote) ,
  \” (double quote) , **\?** (question mark), **\0** (null) etc.
- **String Literals** is a sequence of zero or more characters enclosed in double quotes.

  E.g. “abs”, “amit” , “1234” , “12 A” etc.

In Java, Arithmetic operations (+ - * /) can be applied with characters but are not valid for strings except String addition (concatenation).

‘a’+’b’ ⇨ 195
‘a’-’b’ ⇨ -1
“a”+”b” ⇨ “ab”
“a”-”b” ⇨ invalid

Using Escape Sequence Characters

Suppose we want to display a message in message dialog like-

Raman said “Hello” (Notice that “” is part of message)

If you write-

```
JOptionPane.showMessageDialog(“Raman said “Hello” “);
```

Java will produce an error!! Because of improper termination of string. To notify Java that “ is part of message you must use ‘\”’ escape character. The correct code will be-

```
JOptionPane.showMessageDialog(null,"Raman said \”Hello\” “);
```

Other escape characters also used for specific purpose.

“Hello\n dear”

Or “Hello”+’\n’+”dear”

Hello
dear

\n breaks line

“Jai “+’\t’ + “Hind”

Or “Jai \t Hind”

Jai Hind

\t gives long space (tab)

“Raj\\Kumar “ will give Raj\Kumar

Concept of Data types

- Data types refers the type of data and its associated operations of handling it.
- Java offers two major types of data types.

1. Primitive (Intrinsic) Data Type:

These are in-built data types offered by the Java compiler. Java supports 8 primitive data types.

- Numeric Data types:

  byte, short, int, long, float, double and char.

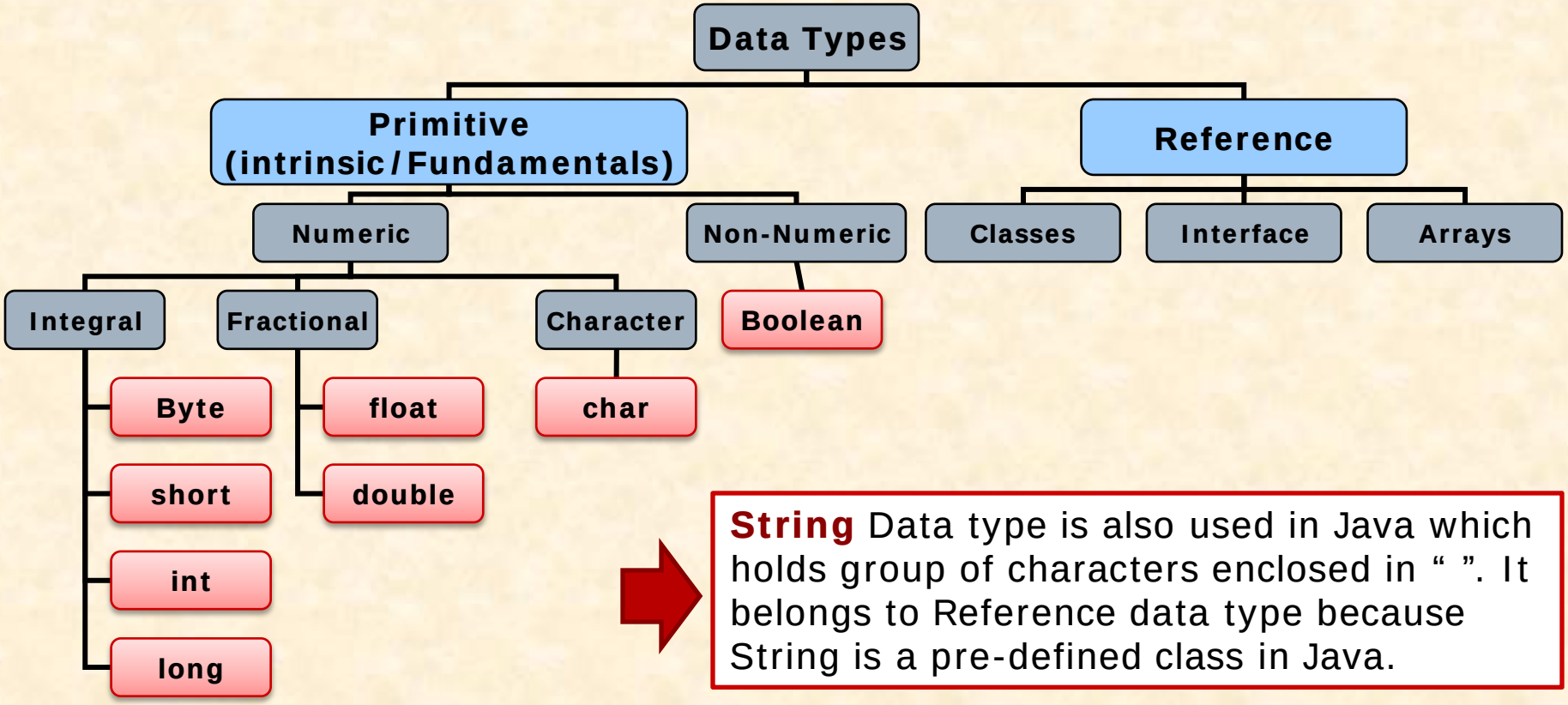
- Non-Numeric Data types:

  boolean

2. **Reference Data Type:**

These are user-defined data types and constructed as per our need by using primitive data types. Reference data types store the memory address of an object. Class, Interface and Array are the example of Reference Data types.

Data Types in Java



String Data type is also used in Java which holds group of characters enclosed in " ". It belongs to Reference data type because String is a pre-defined class in Java.

char data type holds a single character enclosed in ' '. It is assumed as numeric data types because a Character is stored as number (ASCII value) in the memory. So arithmetic operations (+,-,*) are valid for characters. Ex: 'A'+'B' gives 131 (ASCII value of 'A' and 'B' i.e. 65 and 66 are added)

Primitive Data types

Type	Size	Description	Range
byte	1 Byte	Byte integer	-128 to +127
short	2 Byte	Short integer	-32768 to +32767
int	4 Byte	integer	-2^{31} to $2^{31}-1$
long	8 Byte	Long integer	-2^{63} to $2^{63}-1$
float	4 Byte	Single precision floating point (up to 6 digit)	-3.4×10^{-38} to $+3.4\times10^{38}$
double	8 Byte	Double precision floating (up to 15 digit)	-1.8×10^{-308} to $+1.8\times10^{308}$
char	2 Byte	Single character	0 to 65536
Boolean	1 Byte	Logical Boolean values	True or False

- L suffix can used to indicate the value as long.
- By default Java assume fractional value as double, F and D suffix can be used with number to indicate float and double values respectively.

Working with Variables

- A variable is a named memory location, which holds a data value of a particular data type.
- **Declaration and Initialization of variable-**
  <data type> <variable Name>;

```
int age;
double amount, discount;
double price=214.70;
String name="Amitabh"
long x=25L;
byte a=3;
float x= a+b;
```

Variables are Identifiers, so Naming rules for Identifiers are also applied with variables.

- By default all Numeric variables initialized with 0, and character and reference variable with null, boolean with false, if it is not initialized.
- The keyword final can be used with variable declaration to indicate constant.

  E.g. `final double SERVICE_TAX=0.020;`

Using Variables and jTextFields

In GUI application often we require to store the values of text fields to variable or vice-versa. Java offers three method for this purpose-

- **getText():**

  It returns the text stored in the text based GUI components like Text Field, Text Area, Button, Label, Check Box and Radio Button etc. in string type.

  e.g. String str1=jTextField1.getText();

- **parse……()**

  This method convert textual data from GUI component in to numeric type.

Byte.parseByte(String s)	– string into byte.
Short.parseShort(String s)	– string into short.
Integer.parseInt(string s)	– string into integer.
Long.parseLong(string s)	– string into long.
Float.parseFloat(string s)	– string into float.
Double.parseDouble(string s)	– string into double.

  e.g. int age=Integer.parseInt(jTextField1.getText());

- **setText()**

  This method stores string into GUI component.

  e.g. jTextField1.setText("Amitabh");

  jLabel1.setText(""+payment);

Alternatively it can written as- jLabel1.setText(Integer.toString (payment));

Operators in Java

- The operators are symbols or words, which perform specified operation on its operands.
- Operators may be Unary, Binary and Turnery as per number of operands it requires.
- Java offers the following types of Operators:-
  1. Arithmetic Operator
  2. Increment/Decrement Operator
  3. Relational or Comparison Operators
  4. Logical Operators
  5. Assignment Operators
  6. Other Operators

1. Arithmetic Operators

+	Unary plus	Represents positive values.	int a=+25;
-	Unary minus	Represents negative values.	int a=-25;
+	Addition	Adds two values	int x= a+b;
-	Subtraction	Subtract second operands from first.	int x=a-b;
*	Multiplication	Multiplies two values	int x= a*b;
/	Division	Divides first operand by second	int x=a/b;
%	Modulus (remainder)	Finds remainder after division.	int x= a%b;
+	Concatenate or String addition	Adds two strings	“ab”+”cd” ⇨ ”abcd” “25”+”12” ⇨ ”2512” “”+5 ⇨ ”5” “”+5+”xyz” ⇨ ”5xyz”

2. Increment & Decrement Operator

- Java supports ++ and -- operator which adds or subtract 1 from its operand. i.e.

  a=a+1 equivalent to ++a or a++

  a=a-1 equivalent to --a or a--

- ++ or -- operator may used in Pre or Post form.

  ++a or --a (increase/decrease before use)

  a++ or a-- (increase/decrease after use)

  Ex. Find value of P? (initially n=8 and p=4)

  p=p* --n; ⇨ 28

  p=p* n--; ⇨ 32

  Ex. Evaluate x=++y + 2y if y=6.

  =7+14 = 21

3.Relational Operator

- Relational operators returns **true** or **false** as per the relation between operands.
- Java offers the following six relational operators.

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Relational operators solved from left to right.

Ex: 3>=3 true

3!=3 false

a==b true if a and b have the same value.

a<b<c => (a<b)<c true if a is smallest.

4.Logical Operator

- Logical operators returns true or false as per the condition of operands. These are used to design more complex conditions.
- Java offers the following six (5 binary and 1 unary) logical operators.

Operator	Name	use	Returns true if
&&	And	x&&y	X and y both true
||	Or	x||y	Either x or y is true
!	Not	!x	X is false
&	Bitwise and	x&y	X and y both true
|	Bitwise or	x|y	Either x or y is true
^	Exclusive or	x^y	If x and y are different

Ex: 5>8 || 5<2 (false)
6<9 && 4>2 (true)
!(5!=0) (false)

1==0||0>1 (false)
6==3 && 4==4 (false)
!(5>9) (true)

5.Assignment Operator

- In Java = operator is known as Assignment operator, it assigns right hand value to left hand variables.

  Ex: int x=5;

  z= x+y;

- Java offers some special shortened Assignment operators, which are used to assign values on a variable.

Operator	use	Equivalent to
+=	X+=y	X=X+Y
-=	X-=y	X=X-Y
*=	X*=y	X=X*Y
/=	x/=y	X=X/Y
%=	X%=y	X=X%Y

Ex: x-=10 ⇨ x=x-10

x%=y ⇨ x=x%y

6. Other Operators

- In Java some other operators are also used for various operations. Some operators are-

Operator	Purpose
? :	Shortcut of If condition (turnery operator) <Condition> ? <true action>:<false action>
[]	Used to declare array or access array element
.	Used to form qualified name (refer)
(type)	Converts values as per given data type (type casting)
new	Creates a new object
instanceof	Determines whether the object is an instance of a class.

Ex. result = marks **>=** 50 **?** 'P' **:** 'F'

6>4 ? 9:7 evaluates 9 because 6>4 is true.

How to use Conditional (turnery) operator

Operator's Precedence

Operator's precedence determines the order in which expressions are evaluated. There is certain rules called Precedence for evaluating a complex expression.

e.g. y=6+4/2 (why 8 not 5 ?)

Operators	Remark	Associativity
. [] ()	() used to make a group, [] used for array and . Is used to access member of object	L to R
++ -- !	Increment/Decrement Operator, Not Operator Returns true or false based on operands	R to L
New, (type)	New is used to create object and (type) is used to convert data into other types.	R to L
* / %	Multiplication, division and modulus	L to R
+ -	Addition and Subtraction	R to L
< <= > >=	Relational Operators	L to R
== !=	Equality and not equality	L to R
& , ^, |	Bitwise And, Bitwise Exclusive Or , Bitwise or	L to R
&&	Logical And	L to R
||	Logical or	L to R
? :	Shortcut of IF (conditional Operator)	R to L
=, +=, -=, *=, /=, %=	Various Assignment operators	R to L

Expression in Java

- An expression is a valid combination of operators, constants and variable and keywords i.e. combination of Java tokens.
- In java, three types of expressions are used.
- **Arithmetic Expression**

  Arithmetic expression may contain one or more numeric variables, literals and operators. Two operands or operators should not occur in continuation.

  e.g. x+*y and q(a+b-z/4) are invalid expressions.

  Pure expression: when all operands are of same type.

  Mixed expressions: when operands are of different data types.
- **Logical Expression**

  Logical or Boolean expression may have two or more simple expressions joined with relational or logical operators.

  e.g. ◆ x>y ◆ (y+z)>=(x/z) ◆ x||y && z
- **Compound Expression**

  It is combination of two or more simple expressions.

  e.g. ◆ (a+b)/(c+d) ◆ (a>b)||(b<c)

Data Type Conversion in JAVA

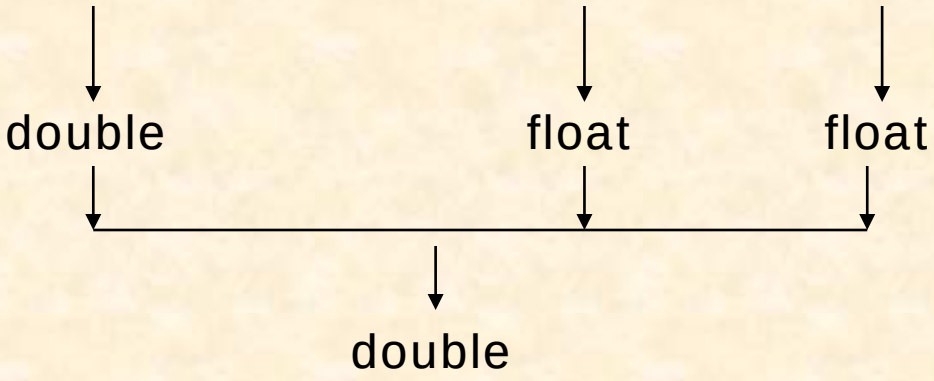
- The process of converting one predefined type into another is called type conversion.
- In mixed expression, various types of constant and variables are converted into same type before evaluation.
- Java facilitates two types of conversion.
  - ✓ Implicit type conversion
  - ✓ Explicit type conversion

Implicit Type Conversion

- It is performed by the compiler, when different data types are intermixed in an expression. It is also called Coercian.
- In Implicit conversion, all operands are promoted (Coercion) up to the largest data type in the expression.

Example: Consider the given expression, where f is float, d is double and I is integer data type.

Result= (f * d) - (f + i) + (f / i)



Explicit Conversion in JAVA

- An explicit conversion is user defined data type conversion that forces to convert an operand to a specific data type by (type) cast.

  e.g. (float) (x/2) suppose x is integer.

  The result of x/2 is converted in float otherwise it will give integer result.

- In pure expression the resultant is given as expression's data type i.e two integers yields integer only.

  e.g. 100/11 will give 9 not 9.999 (since both are integer)

- In mixed expression the implicit conversion is applied (largest type promotion)

  e.g. int a, m=2, k=4 then evaluate

  a=m*3/4+k/4+8-m+5/8

  = 2 * 3 / 4 + 4 / 4 + 8 – 2 + 5 / 8

  = 6 / 4 + 1 + 8 – 2 + 5 / 8

  = 1 + 1 + 8 – 2 + 0 (6/4 will give 1)

  = 8

JAVA Statements

- A statement in Java is a complete unit of execution. It may consists of Expression, Declaration, Control flow statements and must be ended with semicolon (;)
- Statements forms a block enclosed within { }. Even a block may have no statement (empty).

E.g.
```
If(a>b)
  {…….
    …….
  }
```

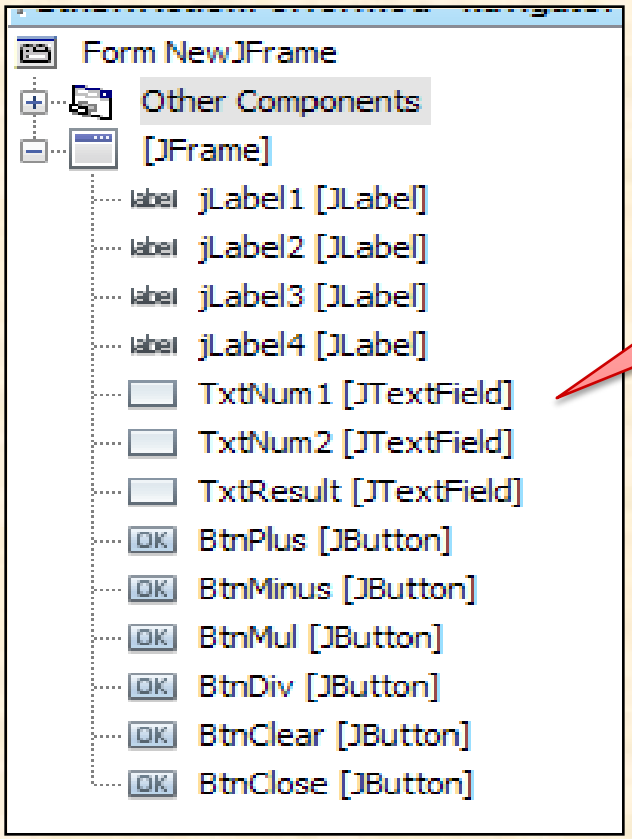
System.out.print() is a Java non-GUI command which works as TextField but gives its result in Output Window

- **Just Think….. Why?**

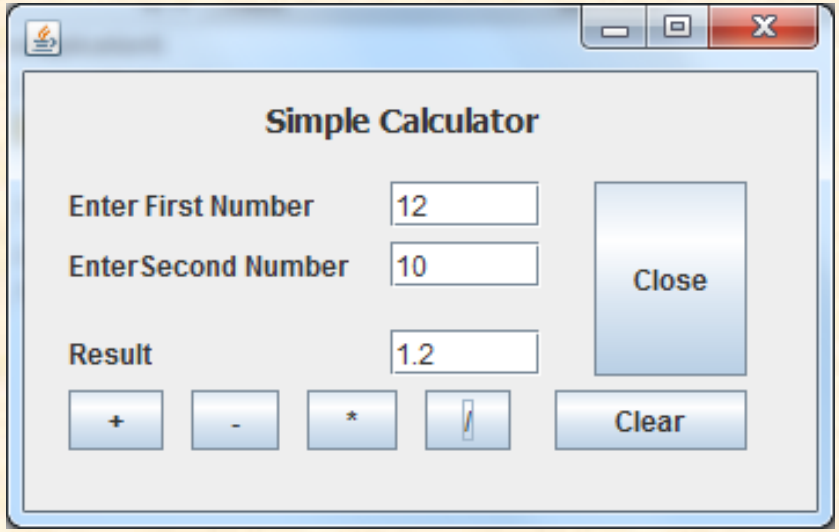
```
System.out.print('h'+'a') will give 169
System.out.print(" "+'h'+'a') will give ha
System.out.print("2+2="+2+2) will give 2+2=22
System.out.print("2+2="+(2+2)) will give 2+2=4
```

Demo Application- Using Variables

Consider the following Java Application “Simple Calculator” which performs +,-,* and / operation on given number.
Design Application as per design and controls given.



Default names of controls are renamed for better understanding of the program..



Demo Application- Using Variables

```
private void BtnPlusActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Float x,y,z;
    x=Float.parseFloat(TxtNum1.getText());
    y=Float.parseFloat(TxtNum2.getText());
    z=x+y;
    TxtResult.setText(""+z);
}
```

```
private void BtnMinusActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Float x,y,z;
    x=Float.parseFloat(TxtNum1.getText());
    y=Float.parseFloat(TxtNum2.getText());
    z=x-y;
    TxtResult.setText(""+z);
}
```

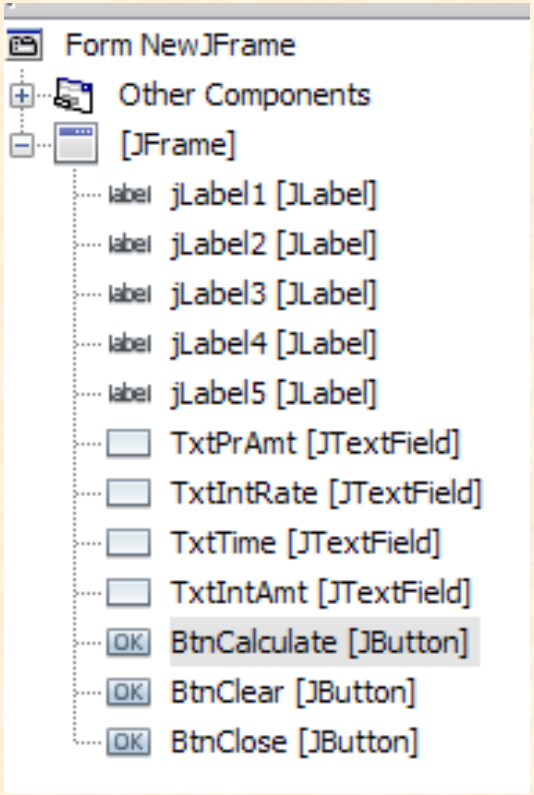
The same code can be written for **BtnMul** and **BtnDiv** also with change in operator.

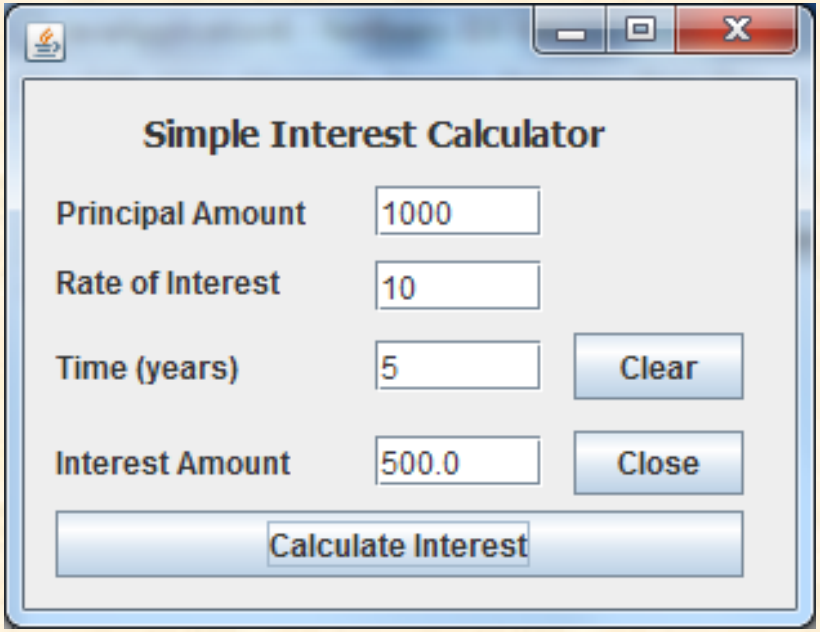
```
private void BtnClearActionPerformed(java.awt.eve
    // TODO add your handling code here:
    TxtNum1.setText("");
    TxtNum2.setText("");
    TxtResult.setText("");
}
```

```
private void BtnCloseActionPerformed(jav
    // TODO add your handling code here:
    System.exit(0);
}
```

Demo Application- Using Variables

Consider the following Java Application “Simple Interest Calculator” which calculate simple interest for given Principal Amount, rate of interest and time (years) using P*R*T/100 formula.
Design Application as per design and controls given.





Demo Application- Using Variables

```
private void BtnCalculateActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Float p,r,t, i;
    p=Float.parseFloat(TxtPrAmt.getText());
    r=Float.parseFloat(TxtIntRate.getText());
    t=Float.parseFloat(TxtTime.getText());
    i= (p*r*t)/100;
    TxtIntAmt.setText(""+i);
}
```

```
private void BtnClearActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    TxtPrAmt.setText("");
    TxtIntRate.setText("");
    TxtTime.setText("");
    TxtIntAmt.setText("");
}
```

```
private void BtnCloseActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    System.exit(0);
}
```