Java language basics

phaneendra

Abstract

Java is one of the most popular and widely used programming language and platform. A platform is an environment that helps to develop and run programs written in any programming language.

Java is fast, reliable and secure. From desktop to web applications, scientific supercomputers to gaming consoles, cell phones to the Internet, Java is used in every nook and corner.

About Java

- **Java is a simple language:** Java is easy to learn and its syntax is simple and easy to understand. It is based on C++ (so easier for programmers who know C++). Java has removed many confusing and rarely-used features e.g. explicit pointers, operator overloading etc. Java also takes care about memory management and for that, it provides an automatic garbage collector. This collects the unused objects automatically.
- **Java is a platform-independent language:** The programs written in Java language, after compilation, are converted into an intermediate level language called the bytecode which is irrespective of the machine on which the programs runs. This makes java highly portable as its bytecodes can be run on any machine by an interpreter called the Java Virtual Machine(JVM) and thus java provides 'reusability of code'.
- **Java is an object-oriented programming language:** OOP makes the complete program simpler by dividing it into a number of objects. The objects can be used as a bridge to have data flow from one function to another. We can easily modify data and function as per the requirement.
- **Java is a robust language:** Java programs must be reliable because they are used in both consumer and mission-critical applications, ranging from Blu-ray players to navigation systems.
- **Java is a multithreaded language:** Java can perform many tasks at once by defining multiple threads. For example, a program that manages a Graphical User Interface (GUI) while waiting for input from a network connection uses another thread to perform the wait instead of using the default GUI thread for both tasks. This keeps the GUI responsive.
- **Java programs can create applets:** Applets are the programs which run on web browsers.
- **Java does not require any preprocessor:** It does not require inclusion of header files for creating a Java application.

1 Identifiers

A name in java programme is by default considered as identifier which can be used for identification purpose.it can be variables name, methods name, classes name, packages name and interfaces name and label name.
Identifiers must be composed of letters, numbers, the underscore _ and the dollar sign $. Identifiers may only begin with a letter, the underscore or a dollar sign.
Each variable has a name by which it is identified in the program. It's a good idea to give your variables mnemonic names that are closely related to the values they hold.
Variable names can include any alphabetic character or digit and the underscore_. The main restriction on the names you can give your variables is that they cannot contain any white space. You cannot begin a variable name with a number. It is important to note that as in C but not as in Fortran or Basic, all variable names are case-sensitive. MyVariable is not the same as myVariable. There is no limit to the length of a Java variable name.

```
// Test.java
class Test
{
  public static void main(String args[])
    {
      int x=0;
      System.out.println("Hello World");
    }
}
```

- Test -class name
- main - method name
- String - predefined class name
- args - array name
- x - variable name

total_number: ✓
total123: ✓
total#: ✗
test_123: ✓
123toal: ✗
all@phani : ✗

```
int number=10; ✓
int Number=10; ✓
int nUMBER=10; ✓
int if=10 ; ✗
int Integer=10; ✓
int Int=10; ✓
int int=10 ; ✗
```

1.1 Java Naming conventions

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc. But, it is not forced to follow. So, it is known as convention not rule.

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

Advantage of naming conventions in java By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

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

1.1.1 CamelCase in java naming conventions

Java follows camelcase syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

2 Keywords in Java

You can't use keyword as identifier in your Java programs, its reserved words in Java library and used to perform an internal operation.

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

3 Data types in Java

There are majorly two types of languages. First one is **Statically typed language** where each variable and expression type is already known at compile time.Once a variable is declared to be of a certain data type, it cannot hold values of other data types.Example: C,C++, Java. Other, **Dynamically typed languages:** These languages can receive different data types over the time. Ruby, Python
Primitives are the most basic kinds of data types and they driectly contain values
There are eight primitive types in total:

- byte
- short
- int
- long
- char
- boolean
- float
- double

Non primitive data types are called reference types in Java and they refer to an object. They are created by the programmer and are not defined by Java like primitives are. A reference type references a memory location where the data is stored rather than directly containing a value.

3.1 byte

The byte data type is an 8-bit signed two's complement integer.The byte data type is useful for saving memory in large arrays.

- **Size:** 8-bit
- **Value:** -128 to 127

```
// Testbyte.java
public class Testbyte
{
    public static void main(String args[])
    {
        byte a = 127;
         System.out.println(a);
        a++;
        System.out.println(a);
    }
}
```

```
//output

D:\>javac  Testbyte.java
D:\>java Testbyte
 127
 -128
```

```
// Testbyte2.java
public class Testbyte2
{
    public static void main(String args[])
    {
        byte a = 500;
         System.out.println(a);

    }
}
```

```
//output

D:\>javac Testbyte2.java
Testbyte2.java:5: error: incompatible types:
 possible lossy conversion from int to byte
        byte a = 500;
                 ^
1 error
```

3.2 short

The short data type is a 16-bit signed two's complement integer. Similar to byte, use a short to save memory in large arrays, in situations where the memory savings actually matters.

- **Size:** 16-bit
- **Value:** -32,768 to 32,767 (inclusive)

```
// Testshort.java
public class Testshort
{
    public static void main(String args[])
    {
        short s=100;
         System.out.println(s);

        s--;
        System.out.println(s);


    }
}
```

```
//output

D:\>javac  Testshort.java
D:\>java Testshort
100
99
```

```
// Testshort2.java
 class Testshort2
{
    public static void main(String args[])
    {
        short s=32900;
         System.out.println(s);

        s--;
        System.out.println(s);


    }
}
```

```
//output

D:\>javac Testshort2.java
Testshort2.java:5: error: incompatible types:
 possible lossy conversion from int to short
        short s=32900;
                ^
1 error
```

3.3 int

It is a 32-bit signed two's complement integer.

- **Size:** 32-bit
- **Value:** -2^{31} to$2^{31} - 1$.

3.4 long

The long data type is a 64-bit two's complement integer.

- **Size:** 64-bit
- **Value:** -2^{63} to$2^{63} - 1$.

3.5 float

The float data type is a single-precision 32-bit IEEE 754 floating point. Use a float (instead of double) if you need to save memory in large arrays of floating point numbers.

- **Size:** 32-bit
- **sufix:** -F/f example 9.8f

```
//Testfloat.java
class Testfloat
{
    public static void main(String args[])
    {
        float f1=12345.678910f;
         System.out.println(f1);

    }
}
```

```
    //output

D:\>javac Testfloat.java

D:\>java Testfloat
12345.679
```

```
//Testfloat.java
class Testfloat
{
    public static void main(String args[])
    {
        float f2=123456789;
         System.out.println(f2);

    }
}
```

```
    //output

D:\>javac Testfloat.java

D:\>java Testfloat
1.23456792E8
```

```
//Testfloat.java
 class Testfloat
{
    public static void main(String args[])
    {
        float f3=12345678910;
         System.out.println(f3);

    }
}
```

```
    //output

D:\>javac Testfloat.java
Testfloat.java:5: error:
 integer number too large: 12345678910
        float f3=12345678910;
                 ^
1 error
```

3.6 double

The double data type is a double-precision 64-bit IEEE 754 floating point. For decimal values, this data type is generally the default choice.

- **Size:** 8-bytes
- approximately $\pm 1.7976931348623157 E + 308$ (15 significant decimal digits)

3.7 char

The char data type is a single 16-bit Unicode character. A char is a single character.

- **Size:** 2-bytes
- 0 to 65,536 (unsigned)

```
//Testchar.java
class Testchar
{
    public static void main(String args[])
    {

       int i=65;
char c=i;

       System.out.println("char: " + c);
        System.out.println("char: " + a);
    }
}
```

//output

```
D:\>javac Testchar.java
Testchar.java:7: error: incompatible types:
 possible lossy conversion from int to char
        char c=i;
               ^
1 error
```

```
//Testchar.java
class Testchar
{
    public static void main(String args[])
    {
        // declaring character
        char a = 'B';
       char i=65;
       System.out.println("char: " + a);
        System.out.println("char: " + i);
    }
}
```

//output

```
D:\>javac Testchar.java

D:\>java Testchar
char: B
char: A
```

3.8 boolean

A Boolean data type can only have a value of either true of false

```
//Testboolean.java
class Testboolean
{
    public static void main(String args[])
    {
      boolean b1=true;
      boolean b2=false;
      System.out.println("boolean1: " + b1);
      System.out.println("boolean2: " + b2);
    }
}
```

```
//output

D:\>javac TestBoolean.java

D:\>java Testboolean
boolean1: true
boolen2: false
```

```
//Testboolean.java
class Testboolean
{
    public static void main(String args[])
    {

      boolean b3=True;
  boolean b4=FALSE;

       System.out.println("boolean1: " + b3);
        System.out.println("boolen2: " + b4);
    }
}
```

```
D:\>javac TestBoolean.java
TestBoolean.java:6: error: cannot find symbol
      boolean b3=True;
                 ^
  symbol:   variable True
  location: class Testboolean
TestBoolean.java:7: error: cannot find symbol
  boolean b4=FALSE;
             ^
  symbol:   variable FALSE
  location: class Testboolean
2 errors
```

3.8.1 List of Java's primitive data types

datatype	size	range
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 to 2,147,483, 647
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	approximately $\pm 3.40282347E+38F$ (6-7 significant decimal digits)
double	8 bytes	approximately $\pm 1.79769313486231570E+308$ (15 significant decimal digits)
char	2 byte	0 to 65,536 (unsigned)
boolean	not precisely defined* true or false	

3.8.2 Default values of primitive data types in Java

byte	0
short	0
int	0
long	0
float	0.0f
double	0.0d
char	'\u0000'
boolean	false
String or other object	null

4 variables

There are three kinds of variables in Java

- instance
- static
- local

4.1 Instance Variables

- nstance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class. Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.

```
//Testiv.java
class Testiv
 {

   int rollno=1230;
   public static void main(String args[])
 {
      Testiv instance=new Testiv();
      System.out.println(instance.rollno);

 }
}
```

```
//output

D:\>javac Testiv.java

D:\>java Testiv
1230
```

```
//Testiv.java
class Testiv
 {
   int rollno=1230;
   public static void main(String args[])
 {
      System.out.println(rollno);
      System.out.println(testiv.rollno);
 }
}
```

```
//output

D:\>javac Instancevar.java
Instancevar.java:9: error:
 non-static variable rollno cannot be referenced from
a static context
      System.out.println(rollno);
                         ^
Instancevar.java:10: error: cannot find symbol
      System.out.println(testiv.rollno);
                         ^
  symbol:   variable testiv
  location: class Testiv
2 errors
```