JAVA - QUICK GUIDE

What is Java?

Java is:

- Object Oriented

- Platform independent:

- Simple

- Secure

- Architectural- neutral

- Portable

- Robust

- Multi-threaded

- Interpreted

- High Performance

- Distributed

- Dynamic

Java Environment Setup:

Java SE is freely available from the link Download Java. So you download a version based on your operating system.

You can refer to installation guide for a complete detail.

Java Basic Syntax:

- **Object -** Objects have states and behaviors. Example: A dog has states-color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.

- **Class -** A class can be defined as a template/ blue print that describe the behaviors/states that object of its type support.

- **Methods -** A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

- **Instant Variables -** Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.

First Java Program:

Let us look at a simple code that would print the words *Hello World*.

```
public class MyFirstJavaProgram{

   /* This is my first java program.
    * This will print 'Hello World' as the output
    */

   public static void main(String []args){
      System.out.println("Hello World"); // prints Hello World
   }
}
```

1

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity -** Java is case sensitive which means identifier **Hello** and **hello** would have different meaning in Java.

- **Class Names -** For all class names the first letter should be in Upper Case.

  If several words are used to form a name of the class each inner words first letter should be in Upper Case.

  Example *class MyFirstJavaClass*

- **Method Names -** All method names should start with a Lower Case letter.

  If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

  Example *public void myMethodName*

- **Program File Name -** Name of the program file should exactly match the class name.

  When saving the file you should save it using the class name *Rememberjavaiscasesensitive* and append '.java' to the end of the name. *ifthefilenameandtheclassnamedonotmatchyourprogramwillnotcompile*.

  Example : Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as '*MyFirstJavaProgram.java*'

- **public static void main***Stringargs***[] -** java program processing starts from the main method which is a mandatory part of every java program..

Java Identifiers:

All Java components require names. Names used for classes, variables and methods are called identifiers.

In java there are several points to remember about identifiers. They are as follows:

- All identifiers should begin with a letter *AtoZoratoz*, currency character $ or an underscore _.

- After the first character identifiers can have any combination of characters.

- A key word cannot be used as an identifier.

- Most importantly identifiers are case sensitive.

- Examples of legal identifiers:age, $salary, _value, __1_value

- Examples of illegal identifiers : 123abc, -salary

Java Modifiers:

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers.

- **Access Modifiers :** default, public , protected, private

- **Non-access Modifiers :** final, abstract, strictfp

We will be looking into more details about modifiers in the next section.

Java Variables:

We would see following type of variables in Java:

- Local Variables

- Class Variables *StaticVariables*
- Instance Variables *Nonstaticvariables*

Java Arrays:

Arrays are objects that store multiple variables of the same type. However an Array itself is an object on the heap. We will look into how to declare, construct and initialize in the upcoming chapters.

Java Enums:

Enums were introduced in Java 5.0. Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums.

With the use of enums it is possible to reduce the number of bugs in your code.

For example if we consider an application for a fresh juice shop it would be possible to restrict the glass size to small, medium and Large. This would make sure that it would not allow anyone to order any size other than the small, medium or large.

Example:

```java
class FreshJuice{

   enum FreshJuiceSize{ SMALL, MEDIUM, LARGE }
   FreshJuiceSize size;
}

public class FreshJuiceTest{

   public static void main(String args[]){
      FreshJuice juice = new FreshJuice();
      juice.size = FreshJuice. FreshJuiceSize.MEDIUM ;
      System.out.println("Size :" + juice.size);
   }
}
```

Note: enums can be declared as their own or inside a class. Methods, variables, constructors can be defined inside enums as well.

Java Keywords:

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

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		

Comments in Java

Java supports single line and multi-line comments very similar to c and c++. All characters available inside any comment are ignored by Java compiler.

```java
public class MyFirstJavaProgram{

   /* This is my first java program.
    * This will print 'Hello World' as the output
    * This is an example of multi-line comments.
    */

   public static void main(String []args){
      // This is an example of single line comment
      /* This is also an example of single line comment. */
      System.out.println("Hello World");
   }
}
```

Data Types in Java

There are two data types available in Java:

- Primitive Data Types
- Reference/Object Data Types

Primitive Data Types:

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word. Let us now look into detail about the eight primitive data types.

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

Reference Data Types:

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy etc.
- Class objects, and various type of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.

- Example : Animal animal = new Animal"giraffe";

Java Literals:

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Literals can be assigned to any primitive type variable. For example:

```java
byte a = 68;
char a = 'A'
```

String literals in Java are specified like they are in most other languages by enclosing a sequence of characters between a pair of double quotes. Examples of string literals are:

```java
"Hello World"
"two\nlines"
"\"This is in quotes\""
```

Java language supports few special escape sequences for String and char literals as well. They are:

Notation	Character represented
\n	Newline 0x0a
\r	Carriage return 0x0d
\f	Formfeed 0x0c
\b	Backspace 0x08
\s	Space 0x20
\t	tab
\"	Double quote
\'	Single quote
\\	backslash
\ddd	Octal character ddd
\uxxxx	Hexadecimal UNICODE character xxxx

Java Access Modifiers:

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:

- Visible to the package. the default. No modifiers are needed.
- Visible to the class only private.
- Visible to the world public.
- Visible to the package and all subclasses protected.

Java Basic Operators:

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

The Arithmetic Operators:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increase the value of operand by 1	B++ gives 21
--	Decrement - Decrease the value of operand by 1	B-- gives 19

The Relational Operators:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	A == B is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	A != B is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	A > B is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	A < B is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	A >= B is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	A <= B is true.

The Bitwise Operators:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	A & B will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	A | B will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	A ^ B will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the efect of 'flipping' bits.	~A will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.

<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	A && B is false.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	A || B is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!A && B is true.

The Assignment Operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C

		= C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Misc Operators

There are few other operators supported by Java Language.

Conditional Operator ? : :

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as :

```
variable x = (expression) ? value if true : value if false
```

instanceOf Operator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular typeclass type or interface type. instanceOf operator is wriiten as:

```
( Object reference variable ) instanceOf  (class/interface type)
```

Precedence of Java Operators:

Category	Operator	Associativity
Postfix	[] . dot operator	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

The while Loop:

A while loop is a control structure that allows you to repeat a task a certain number of times.

Syntax:

The syntax of a while loop is:

```
while(Boolean_expression)
{
    //Statements
}
```

The do...while Loop:

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

The syntax of a do...while loop is:

```
do
{
    //Statements
}while(Boolean_expression);
```

The for Loop:

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

Syntax:

The syntax of a for loop is:

```
for(initialization; Boolean_expression; update)
{
    //Statements
}
```

Enhanced for loop in Java:

As of java 5 the enhanced for loop was introduced. This is mainly used for Arrays.

Syntax:

The syntax of enhanced for loop is:

```
for(declaration : expression)
{
    //Statements
}
```

The break Keyword:

The *break* keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

The continue Keyword:

The *continue* keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

Syntax:

The syntax of a continue is a single statement inside any loop:

```
continue;
```

The if Statement:

An if statement consists of a Boolean expression followed by one or more statements.

Syntax:

The syntax of an if statement is:

```
if(Boolean_expression)
{
    //Statements will execute if the Boolean expression is true
}
```

The if...else Statement:

An if statement can be followed by an optional *else* statement, which executes when the Boolean expression is false.

Syntax:

The syntax of a if...else is:

```
if(Boolean_expression){
    //Executes when the Boolean expression is true
}else{
    //Executes when the Boolean expression is false
}
```

The if...else if...else Statement:

An if statement can be followed by an optional *else if...else* statement, which is very usefull to test various conditions using single if...else if statement.

Syntax:

The syntax of a if...else is:

```
if(Boolean_expression 1){
    //Executes when the Boolean expression 1 is true
}else if(Boolean_expression 2){
    //Executes when the Boolean expression 2 is true
}else if(Boolean_expression 3){
    //Executes when the Boolean expression 3 is true
}else {
    //Executes when the one of the above condition is true.
}
```

Nested if...else Statement:

It is always legal to nest if-else statements. When using if , else if , else statements there are few points to keep in mind.

- An if can have zero or one else's and it must come after any else if's.

- An if can have zero to many else if's and they must come before the else.

- Once an else if succeeds, none of he remaining else if's or else's will be tested.

Syntax:

The syntax for a nested if...else is as follows:

```
if(Boolean_expression 1){
   //Executes when the Boolean expression 1 is true
   if(Boolean_expression 2){
       //Executes when the Boolean expression 2 is true
   }
}
```

The switch Statement:

A *switch* statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

The syntax of enhanced for loop is:

```
switch(expression){
    case value :
        //Statements
        break; //optional
    case value :
        //Statements
        break; //optional
    //You can have any number of case statements.
    default : //Optional
        //Statements
}
```

Java Methods:

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println method, for example, the system actually executes several statements in order to display a message on the console.

In general, a method has the following syntax:

```
modifier returnValueType methodName(list of parameters) {
  // Method body;
}
```

A method definition consists of a method header and a method body. Here are all the parts of a method:

- **Modifiers:** The modifier, which is optional, tells the compiler how to call the method. This defines the access type of the method.

- **Return Type:** A method may return a value. The returnValueType is the data type of the value the method returns. Some methods perform the desired operations without returning a value. In this case, the returnValueType is the keyword **void**.

- **Method Name:** This is the actual name of the method. The method name and the parameter list together constitute the method signature.

- **Parameters:** A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter

list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.

- **Method Body:** The method body contains a collection of statements that define what the method does.

Java Classes & Objects:

- **Object -** Objects have states and behaviors. Example: A dog has states-color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.

- **Class -** A class can be defined as a template/ blue print that describe the behaviors/states that object of its type support.

A sample of a class is given below:

```java
public class Dog{
    String breed;
    int age;
    String color;

    void barking(){
    }

    void hungry(){
    }

    void sleeping(){
    }
}
```

A class can contain any of the following variable types.

- **Local variables .** variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

- **Instance variables .** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

- **Class variables .** Class variables are variables declared with in a class, outside any method, with the static keyword.

Exceptions Handling:

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```java
try
{
    //Protected code
}catch(ExceptionName e1)
{
    //Catch block
}
```

Multiple catch Blocks:

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```java
try
{
    //Protected code
```

```
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}
```

The throws/throw Keywords:

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword. Try to understand the different in throws and throw keywords.

The finally Keyword

The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax:

```
try
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}finally
{
   //The finally block always executes.
}
```

For a complete detail of the Java Programming language, it is recommended to go through our simple Java Tutorial.

Processing math: 26%