Ch – 1
History , Intro. & Language, Basics Classes and Objects

Prepared By :

Ms. Kakadiya Jainam A.

About Java :

- Java was developed by James Gosling, Patrick Naughton and their team at the Sun Microsystems, Inc. in **1991**.

- Java is an Object Oriented Programming Language.

- The term OOP can be understood by the OOP principles.

- The programming language that follows or supports the OOP principles can be termed as Object Oriented Programming Language.

Object Oriented Programming And Principles :

- In OOP your complex program is divided in module like data (member variables) and methods (functions).
- This data and methods are wrapped into a single unit which is known as class.
- There are mainly three principles of OOP.

  Encapsulation

  Inheritance

  Polymorphism

Encapsulation :

- Encapsulation is a mechanism by which we can bind data (member variables) and methods together.

- It is a process of wrapping up of data and methods into a single unit.

- So by encapsulation we can achieve the concept of data hiding.

Inheritance :

- Inheritance is very useful principle of Object Oriented Programming.

- It simply means to inherit something from one class to another.

- **For Ex.** If object of class A gets the properties, data or methods of class B, then B class is called being inherited in the class A. That means the process of obtaining properties of another class to the object of one class is known as Inheritance.

Polymorphism :

- Polymorphism is a Greek term, which means the ability to take more than one forms by a single entity.
- In OOP, polymorphism plays an important role.
- Polymorphism can be achieved by many ways like method overloading, constructor overloading and method overriding etc.
- For Example, A single man can behave differently in different situations. Same as a single method performs differently depending its argument.

Features of Java / Java Buzzwords :

- Java Buzzwords are features of Java. This makes the Java most striking of programming language among other languages.

1) Simple
2) Secure
3) Portable
4) Object-Oriented
5) Robust (Strong)
6) Multithreaded
7) Architecture Neutral (Performance Independent)
8) Interpreted
9) High Performance
10) Distributed
11) Dynamic

Features of Java / Java Buzzwords :

1) Simple

- Learning Java is said to be simple if you have some basic knowledge of C / C++.
- Yes, Java adopts many syntax of C language and if one understand the concept of Object Oriented Programming, then java will be much easier.
- Many syntax like comments, declaring variables, etc are same as C and C++.

Features of Java / Java Buzzwords :

2) Secure

- Security is the most important need for any programming language.
- While downloading a file from Internet there is a big risk of viral infection.
- In case of Java there is no risk of it. Because Java provides a ***Firewall*** between your computer and the application from which the computer downloads files.
- Java does not allow access to other parts of your program.

Features of Java / Java Buzzwords :

3) Portable

- A Java program is fully portable. So to make the program portable.
- Java compiler generates executable code known as Bytecode. Bytecode then interpreted via Java Virtual Machine (JVM).
- This Bytecode is in machine language code which every machine can read. This is main benefit of language portability.

Features of Java / Java Buzzwords :

4) Object Oriented

- Java is a pure Object Oriented Programming language.
- In OOP language the main focus is on the data (member variables) and methods.
- Everything to be processed is put in the methods using data or member variables.

Features of Java / Java Buzzwords :

5) Robust (Strong)

- Java is said to be robust (strong) language.
- There are two main reasons that makes it very robust : Memory Management and Exception Handling.
  - Memory Management is much easier in Java as it automatically deallocates (free) memory through garbage collection.
  - Exception Handling in Java helps programmer to write efficient (કાર્યક્ષમ) programs.
  - Divide by zero error, File not found exceptions etc. are errors that often arise during programming. Java has its own Exception Handler or you can manually handle exception also.

Features of Java / Java Buzzwords :

6) Multithreaded

- A thread is a sub (child) process.
- Using multithreading you can execute more than one thread simultaneously.
- Java has its own Thread class.

Features of Java / Java Buzzwords :

7) Architecture Neutral (Platform Independent)

- In Java the source code is first compiled by java compiler and then the compiler generates Bytecode.
- The Bytecode is in machine readable form so that every machine can understand it. Then the Bytecode interpreted using Java Virtual Machine.
- Therefore the java program written on any platform can be run on any other platform.
- This features makes java Architecture Neutral or Platform Independent.

Features of Java / Java Buzzwords :

8) Interpreted

- The execution of Java program is a two step process.
- First the source code is Compiled using Java compiler and generated Bytecode.
- Then the Bytecode is machine independent which is interpreted using Java Virtual Machine. And give output.

Features of Java / Java Buzzwords :

9) High Performance

- This feature is the result of above two features.
- As Java generates Bytecode which is machine independent and are interpreted by JVM (Java Virtual Machine) there is a great performance arise.

Features of Java / Java Buzzwords :

10) Distributed

- Java is a very good in Networked programming like client / server architecture.
- A method on one PC can be invoked by the object of another PC. This is done by Remote Method Invocation (RMI).

Features of Java / Java Buzzwords :

11) Dynamic

- Java is far more dynamic language than C or C++.
- In java Libraries can add new methods and variables as needed.
- You can get information of variables easily.
- This is useful when you want to add or update code while downloading programs from Internet.

JDK and its components :

- The Java Development Kit (JDK) is the installation package of java.
- It has a collection of components that are used in development of Java program. The JDK components are given below.

JDK Tool	Meaning	Use
javac	Java Compiler	Compiles the source code **(.java file)** and translates it into Bytecode.

JDK and its components :

JDK Tool	Meaning	Use
java	Java Interpreter	Interprets Bytecode and generate output.
javadoc	Java documentation	Used to create documentation for Java source code in HTML format.
javah	Java Header files	It produces header files for the program which uses native methods.

JDK and its components :

JDK Tool	Meaning	Use
javap	Java dissembler	It converts Bytecode file **(.class file)** to program description
jdb	Java Debugger	Used to debug your program.
appletviewer	Applet Viewer	Used to execute Java Applets

Application V/S Applet :

- **Application :** Java Application is a normal program which is run on your machine under the control of operating system.
- **Applet :** Java applet is a program which is run from the web browser or it can be run by appletviewer.

A simple Java Program :

- **(1)** You can type a program using an editor like notepad.

Def 1: Write a java program to print a message on the screen…

```
// This is a testing of message printing
class test
{
    public static void main(String args[])
    {
        System.out.println("Hello");
        System.out.print("Welcome to the world of Java");
    }
}
```

A simple Java Program :

- Save the file by name test.java (it is good to save the file by classname.java).
- Compile the file using javac command.
- If there is no error, that means your program is successfully compiled.
- After successful compilation a test.class file will be generated in you directory **.Class** files are generated for each class in your program.
- Now run the class file using **java** command. Then the output will be displayed.

Definition :

Def [2] Write a program to print your address using println() function.

Def [3] Write a program to print any ten names of religious temple.

Def [4] Write a program to print Phonebook with ten record.

Understanding main() method :

- The signature of main() method is:

  public static void main(String args[])

- **public :** It is access specifier which is public. So that the main() method can accessed from any class.
- **static :** static keyword specifies that main() method is called without having to create instance (object) of its class.
- **void :** It is the return type of main() method. Which specifies that main() method doesn't return any value.

Understanding main() method :

- **main :** main() method is the entry point of your program. From main() the execution of program starts.
- **String args[] :** main() method can take array of String type values as argument, which can be supplied as command line argument.

System.out.println() :

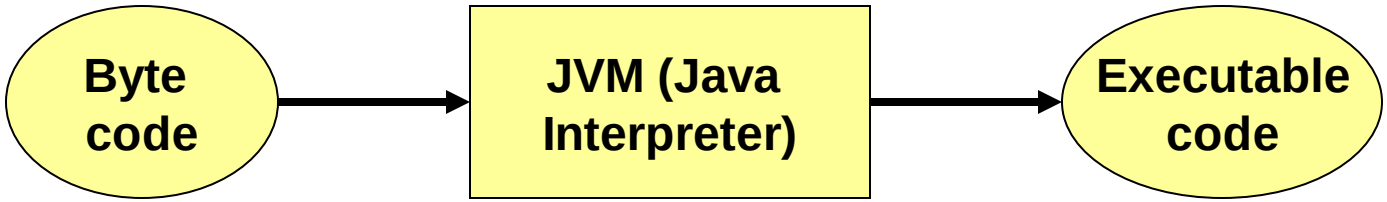
- **System :** System is class that provides access to the system.
- **out :** It is the output stream that is used to send data to the console (output screen).
- **println() :** Prints the string passed as argument and gives a line feed.
- **print() :** Print the string passed as argument.

Java Virtual Machine (JVM) :

- Java Virtual Machine (JVM) is the interpreter for the Bytecode generated from the source code. The execution of java program is two step process.
- **Step 1 :**

[. Java file] **javac compiler** **[.class file]**

- **Step 2 :**



[. class file] **java command** **[output]**

Java Virtual Machine (JVM) :

I. The source file (.java file) is Compiled by javac compiler and Bytecode is generated if no errors are present. Bytecode is .class file.

II. This Bytecode is then interpreted by JVM and the output is generated.

Basic Structure of Java Program :

- There may be more than one class in a java program. But you have to run that class which has the main method.
- Basic structure of a Java Program is as under :

  (1) Documentation Section

  (2) Package Statement

  (3) Import Statement

  (4) Class definition

  (5) Main method

Basic Structure of Java Program :

1) Documentation Section :

- In this section you can write the documentation for your program like, program definition, author information, the logic of your program etc.
- This is optional statement.
- It is written in comment lines (//) or (/*……..*/)

Basic Structure of Java Program :

2) Package statement :

- If you want to save your program in a package (directory) then the package statement is included.
- This is also an optional statement.

3) Import statement :

- If you want to import any library or user defined package, the import statement is used.
- This is an optional section.
- Now from this imported package you can use its classes and methods.
- Example : import java.util.Date;

Basic Structure of Java Program :

4) Class definition

- You can define classes in this section.
- Use class keyword to declare a class.
- There can be member variables and methods in this class.
- This is required and it is compulsory section.

Basic Structure of Java Program :

5) Main method

- In your program you can have more than one class, but there should be one class that contains the main method.
- This class is run to generate output because the main method is the entry point of program execution.
- This is compulsory section.

Java IDE (Integrated Development Environment) (NetBeans and Eclipse) :

- NetBeans and Eclipse are two most popular IDEs (Integrated Development Environment) in which you can develop, debug and deploy your programs easily.
- **NetBeans :** NetBeans is an open source editor with lots of features to develop any kind of complex project.
- It lets you develop java desktop, web and mobile applications, while also providing great tools for PHP and C/C++ developers very quickly and easily.

Java IDE (Integrated Development Environment) (NetBeans and Eclipse) :

- Following are some striking features of NetBeans IDE :

1) Fast and Smart Coding :

- It's not simply a text editor but much more than that. It helps you to code very easily and quickly by providing automatic line indents, it highlights similar keywords, and pair of brackets.
- Also gives coding tips by opening pop-up menu to suggest the related code.

Java IDE (Integrated Development Environment) (NetBeans and Eclipse) :

2) Easy and Efficient (કાર્યક્ષમ) Project Management :

- It helps you to easily manage your project even if it is getting large with number of files with a large number of lines.

3) Rapid User Interface Development :

- It provides drag and drop facility to rapidly develop and designed your form or pages.

Java IDE (Integrated Development Environment) (NetBeans and Eclipse) :

4) Bug free Code :

- The NetBeans provides static analysis tools such as FindBugs tool for identifying and fixing common errors in Java Code.

- **Eclipse :** The Eclipse is another very powerful editor for so many languages such as Java, PHP, C, C++ etc.
- The Eclipse IDE for Java developers contains everything that you will need to built Java applications.

Points of learning :

- Comments
- Identifier
- Literals
- Separator
- Keywords
- Data Types
- Variable
- Type conversation and Type Casting
- Array with its type

Comments :

- There are three types of comments in java.

1) Single line comments : Double slashes are used to indicate single line comment.

   Ex. //This is a single line comment.

2) Multi line comments : The text written between /* and */ considered as comments.

   Ex. /* This line is in comment.
   This is also in comment. */

3) Documentation Comment : Documentation Comment is used to generate as HTML documentation file for your program. The text between /** and */ are documentation comment.

   Ex. /**
   * This is documentation comment
   * @author Mitesh Mandaliya
   */

Identifiers :

- An identifier is a name given to a variable, a method, or a class. To declare an identifier we must follow some rules :
- Identifier may consist of letters (uppercase or lowercase), digits, underscore or a dollar sign.
- It must not start with a number.
- No white space is allowed.
- No keyword can be used as identifier.

Some valid examples of identifiers:

valididentifier, sub1, mark1, first_value, $price

Some invalid examples of identifiers :

In valid, 1mark, first-value.

Literals :

- In Java any constant value is known as literal. The literal can be any one of the following :

- Integer literal :

  Ex. int x=300;

  Here x is a variable and 300 is value which is integer literal.

- Floating point literal :

  Ex. double d=39.86;

  Here 39.86 is floating point literal.

Literals :

- Character literal :

  Ex. char c=‘A’;

  Here A is a character literal.

- Some character literals should be written using escape sequence character, which is back slash (‘\’). For example if you want to print some special character like quotes (“ ”) , back slash will be used.

Literals :

Escape Sequence	Meaning
\n	To give new line
\b	Backspace
\t	To give space as much as tab
\’	To print single quote
\”	To print double quote
\\	To print back slash

Literals :

- String literal :

  Ex. string str=“Java”;

  Here Java is a string literal.

- To print “ I like “Java” “.

  System.out.println(“I like \”Java\” “);

- Boolean literal :

  Boolean literal can be true or false.

  Ex. boolean flag=true;

  Here true is Boolean literal.

White-spaces :

- The white-space can be a space, a tab or a new line.
- The extra white space are ignored by Java compiler.
- Therefore if you write your program n single line it makes no difference.

Separators :

- The separators are used to separate the statements for each other.
- For example, the semicolon (;) is used to terminate a statement. The list of various separator is shown below.

Separator	Name	Use
;	Semicolon	Used to terminate a statement

Separators :

Separator	Name	Use
,	Comma	Used to declare more than one variables of same type. Also used in for() loops to specify more than one conditions.
.	Dot or period	Used to access a variable or method of a class through object. Also used to separate package from sub package.
:	Colon	Used to specify a label.
()	Parenthesis	Used in method declaration and method call. Also used in if conditions and loops.

Separators :

Separator	Name	Use
{}	Braces	Used to enclosed statements of methods, class or any block of code. Also used to initialize an array.
[ ]	Brackets	Used to declare array and array elements.

Keywords :

- Keyword is a pre-defined word.
- There are 49 keywords in Java which are reserved and each one has its unique meaning.
- They can not be used as identifier.
- They are listed below:

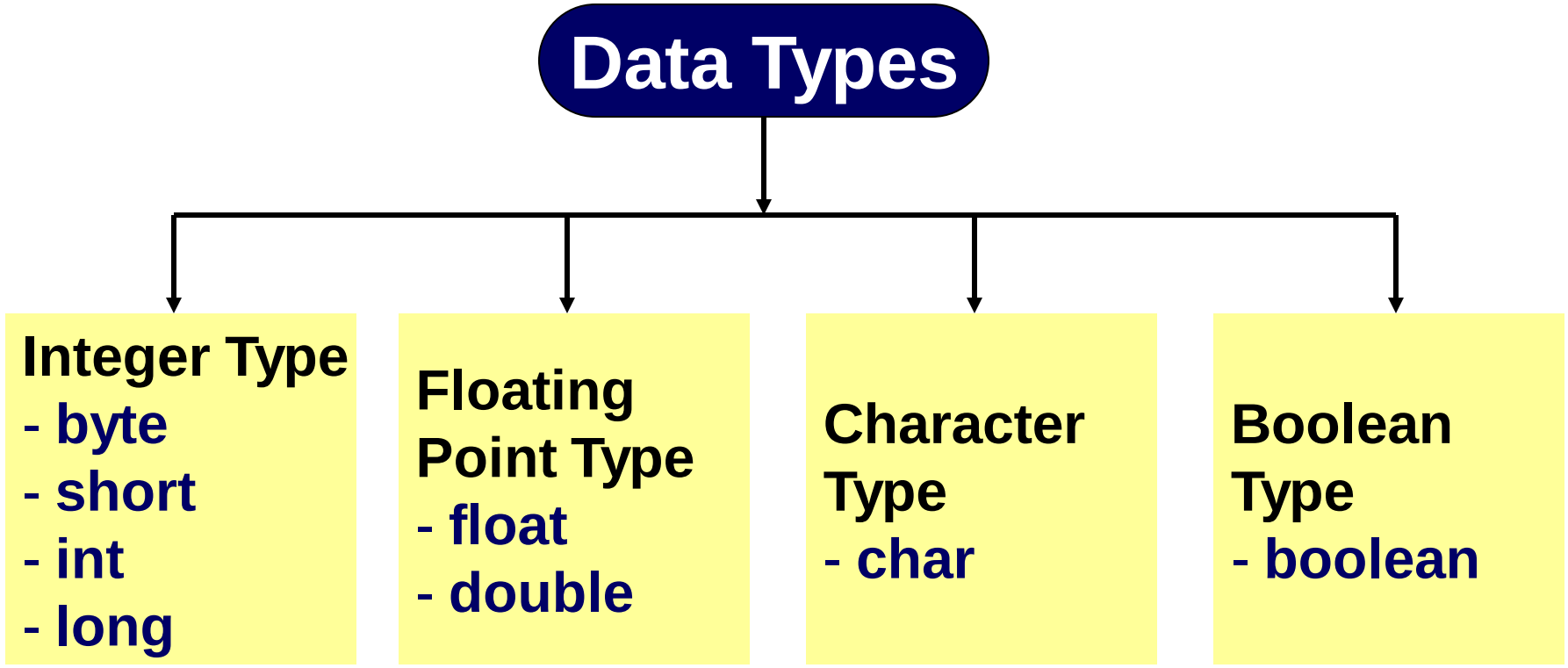
abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
do	double	else	extends	final	finally
float	for	goto	if	implements	Import

Keywords :

instanceof	int	interface	long	native
package	private	protected	public	return
static	strictfp	super	switch	synchronized
throw	throws	transient	try	void
while	new	short	this	volatile

Data types :

- In Java, every variable has a data type. They can be classified as follows:



❑ Integer Data Types :

- The byte, short, int and long are integer data types. They all are signed types. So they can have positive as well as negative values.
- The size and range of these types are as follows.

Type	Size (in bits)	Range
byte	8	-128 **to** 127
short	16	-32,768 **to** 32767
Int	32	-2,14,74,83,648 **to** 2,14,74,83,647
long	64	-9,223,372,036,854,775,808 **to** 9,223,372,036,854,775,807

❑ Integer Data Types :

- ❑ Byte:
- ❖ Byte is a one byte data type. Its size is **8** bits and it can store values between **-2^7 to 2^7-1**.
- ❖ To declare a byte type variable, use byte keyword.
- ❖ **Ex.** byte b;

❑ Integer Data Types :

- ❑ Short:
- ❖ Short is a 16 bit data type. It can store values between **-2^{15} to $2^{15}-1$**.
- ❖ To declare a variable of a short type **short** keyword is used.
- ❖ **Ex.** short s=1000;

❑ Integer Data Types :

- ❑ Int:
- ❖ Size of int type is 32 bits. Its range is **-2^{31} to $2^{31}-1$**.
- ❖ Int keyword is used to declare an int type variable.
- ❖ **Ex.** int i,j;

❑ Integer Data Types :

- ❑ Long
- ❖ Long type is of 64 bit and it can store values between -2^{63} to $2^{63}-1$.
- ❖ To declare a long type variable *long* keyword is used.
- ❖ **Ex.** long val;

❑ Floating point types :

- The float and double are floating point type data types.
- They are used to store floating point values. The size and range of these types are as follows.

Type	Size (in bits)	Range
float	32	1.4e - 045 **to** 3.4e + 038
double	64	4.9e - 324 **to** 1.8e + 308

❑ Floating Point Types :

- ❑ Float
- ❖ Float data type is used when no more precision is required. Its size is 32-bits.
- ❖ It is generally used to store rupees, dollars, etc. with some less digit precision.
- ❖ **Ex.** float f;

❑ Floating Point Types :

- ❑ Double
- ❖ Double is a 64 bit data type. It stores data with double precision (keyword).
- ❖ Therefore if we want to deal with the mathematical functions like sine, cosine, or want to find square root of a number then double data type is used.
- ❖ **Ex.** double d;

❑ Character types :

Type	Size (in bits)	Range
char	64	0 to 65,535

- ❖ Char type is used to store a single character.
- ❖ The char type in Java differs from C and C++.
- ❖ In C and C++ char type is 8 bits in size whereas here char is 16-bit wide which uses Unicode 16-bit format to store data which is used worldwide.
- ❖ Ex. char c='A';

  char ch=65;
- ❖ Both the variables c and ch have the same value.

❑ Boolean types :

Type	Size (in bits)	Range
boolean	8	True **or** False

- Boolean data type is 8 bit data type and used to store logical values. i.e. true/false.
- Every relational operator returns a boolean value.
- **Ex.** boolean flag=false;

  boolean isOK=true;
- In Java all variables are of a specific data type and must be initialized before using them.

❑ Boolean types :

- So if we do not initialize them, they have following default values.
- But remember that these default values are just to initialize with default values, i.e. Java does not automatically initialize them.

Data Type	Default Initial Value
byte	0
short	0
int	0
long	0L

Data Type	Default Initial Value
float	0.0f
double	0.0d
char	'\0'
boolean	false

❑ Variable :

- A variable is an identifier for storage in a Java program. Each variable has a data type and it has a scope.
- To declare a variable:

  Synatax: Data_type Identifier [= value];

  Ex. int i=0; or int x;

Here data type is any one data type.

You can declare multiple variables of same types at once.

Ex. int a,b,c; or int a=11,b=220,c;

❑ The scope and Lifetime of Variable :

- When you declare a variable, it has a specific scope up to where it is visible and can be used.
- So when you declare a variable in a block, this block is the scope of this variable.
- So if you declare a variable in a method then it can be used within that method, if you declare it in a class, its scope is the entire class.

❑ An example to demonstrate scope of variable :

```
class scopeEx
{
        public static void main(String args[])
        {
                int a =10;
                if(a>5) //this is a block
                {
                        int b=20;
              System.out.println("A="+a" and B="+b" ");
                a++;
                }       //The scope ends…b's scope ends.
           System.out.println("A is "+a);                    // Ok
// System.out.println("B ="+b); // Error: b's scope is not here…
        }
```

❑ An example to demonstrate scope of variable :

```
    static void scope()
    {
            int c=30;
            System.out.println("c="+c);
      // System.out.println("A is"+a); //Error: A's
      scope is not here…
    }
}
```

❑ Type conversation / Type casting :

- In many expressions we generally assign some values or variables to other variables.
- These variables should be compatible with each other.
- In Java, before assigning value to a variable, if their types are not same, type conversation is necessary.
- There are two types of conversation.

  - Implicit (Automatic) type conversion.

  - Explicit type conversion also known as type casting.

❑ Type conversation / Type casting :

Implicit (Automatic) type conversion.

- The implicit type conversion takes place automatically when the types are compatible and the destination type are larger than the source type.
- For Example, if we assign an int type value to a long type variable, it will be converted automatically.
- All numeric data types are compatible with each other but not with char or boolean type.
- Because char and boolean are not compatible with each other.

❑ Type conversation / Type casting :

Explicit type conversion.

- When in an expression the destination type is not larger than the source type, the conversion will not be done automatically, but we have to do it explicitly by type casting.
- For Example, if we want to assign a long type value to an int type value, this will not done automatically, but you have to use a cast.
- The general form of type casting is:

Var = (target-type) value;

Example: long l=9999999;

int i=(int) l;

❑ Type conversation / Type casting :

❖ The following list shows which type can be converted automatically to which types.

From	To
byte	short, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

❑ Type conversation / Type casting :

- // Simple example of type conversion and type casting.
- **Def.. Write a program to perform implicit and Explicit type casting.**

```
class type_cast
{
        public static void main(String args[])
        {
            int i=1000;
            System.out.println("Value of i="+i);
            long l;
            l=i; // this is implicit type conversion
            System.out.println("Value of L="+l);
```

❑ Type conversation / Type casting :

```
❖  // Simple example
      long ll=999999;
      System.out.println("Value of LL="+l);
      int ii;
      ii=(int)ll; //Explicit type casting.
      System.out.println("Value of ii="+ii);
   }
}
```

❑ Type conversation / Type casting :

1) Write a program for type casting to convert byte type into short.
2) Write a program for type casting to convert short type into double.
3) Write a program for type casting to convert char type into int.
4) Write a program for type casting to convert int type into long.
5) Write a program for type casting to convert long type into float.
6) Write a program for type casting to convert float type into double.

❑ Arrays :

- An array is a group of variables of same data type which have same name.
- Therefore it is used when more than one variables of same data type are to be used in a program.
- There are different types of array can be classified in JAVA.

  (1) One dimensional Array

  (2) Two dimensional Array

  (3) Multi dimensional Array

  (4) Jagged Array

(1) One dimensional Array

- A one dimensional array is a collection of variables with one row and multiple columns or one column and multiple rows.
- Declaration of one dimensional array:
- Syntax:

  Data_Type array_name[ ];

  Ex. int marks[ ];
- Here the mark is said to be array of integers. The array is just created but it is not ready to use. To use it allocate memory space to it using ***new*** keyword.

(1) One dimensional Array

- Syntax:

  Array_name = new Data_Type[ ];

  Ex. marks = new int [10];

- Now marks is allocated size of 10 int values and it can be used now.
- Alternatively you can combine these two statements like:

  int marks[ ] = new int[10];

- Now to initialize array element you can refer to it by its index which starts from 0.

(1) One dimensional Array

- Example:

  int marks[ ] ={60,65,56,76,77};

- Here an array of 5 int is created and initialized with values 60 to first element, 66 to second element and so on.

(1) One dimensional Array (Example)

❖ Example of one dimensional array: **Def ()…Write a program to create an array and print it's sum of values.**

```
class oneDimArray {
    public static void main(String args[])
    {
            int marks[ ];
            marks = new int[3];
            marks[0]=66;
            marks[1]=65;
            marks[2]=70;
            int total=marks[0]+marks[1]+marks[2];
            double per=total/3;
            System.out.println("Total marks is:"+total);
            System.out.println("Percentage is:"+per);
        }
}
```

(1) One dimensional Array (Example)

- **Def.. Write a program to create an array with name of days and print array with for() loop.**
- class oneDimArray {

```
public static void main(String args[])
{
   String days[]={“Monday”,”Tuesday”,
”Wednesday”,”Thursday”,”Friday”,”Saturday”,
”Sunday”};
```

(1) One dimensional Array

```
        for(int i=0;i<7;i++)
        {
                System.out.println("The day ["+(i+1)+"] is: "+days[i]);
        }
    }
}
```

(2) Multi dimensional Array

- The multi dimensional arrays are array of array. It represents a variable which has values in tabular form i.e. in rows and columns.
- The syntax to declare a two dimensional array is:

  data_type array_name[ ][ ] =new data_type[row] [column];

  Ex. int marks[ ] [ ] = new int [3][2];
- Here marks is a two dimensional array which has 3 rows [students] and 2 columns [subjects].

(2) Multi dimensional Array

- To access an element simply specify row and column index. To access marks of 2nd subject of 3rd student use:

  Ex. marks[2][1]=67;

  marks[2] means 0 to 2 = 3 student

  marks[1] means 0 to 1 = 2 subject

(2) Multi dimensional Array(Example)

- **Def.. Write a program to create a multi dimensional array with its values and print array with for() loop.**

```
class multiDim
{
    public static void main(String args[])
    {
        int i,j;
        int marks[][]={{10,20,30},{100,200,300},{1000,2000,3000}};
        for(i=0;i<=2;i++)
        {
                System.out.println(+(i+1));
                for(j=0;j<3;j++)
                {
                        System.out.println("value is:"+marks[i][j]);
                }
        }
    }
}
```

(3) Jagged Array :

- Jagged array is a special type of multi dimensional array in which each row has different number of columns.
- Therefore it is also known as variable length array.
- To declare a jagged array you have to specify only row size and leave the column size blank.

Syntax:

Data_type array_name[ ][ ]=new data_type[Rowsize][ ];

Ex. double d[ ][ ] = new double[5][ ];

int j[ ][ ]=new int[5][ ];

- Here j is a jagged array having 4 rows.

(3) Jagged Array :

- Now you can declare the size of each columns for each row as shown below.

```
j[0]=new int[3];
j[1]=new int[2];
j[2]=new int[4];
j[3]=new int[3];
```

	0	1	2	3
0	J[0][0]	J[0][1]	J[0][2]	
1	J[1][0]	J[1][1]		
2	J[2][0]	J[2][1]	J[2][2]	J[2][3]
3	J[3][0]	J[3][1]	J[3][2]	

(3) Jagged Array :

- First row i.e. j[0] contains 3 columns, second row j[1] has 2 columns, third row j[2] has 4 columns, and so on.
- Following is an example of jagged array in which we have initialized all the rows of array.

Example : Jagged Array.

```
class jagged_arrayEx
{
    public static void main(String args[ ])
    {
      int a[][]={{1,2,3,4},{1,2},{1,2,3},{1,2,3,4,5}};
```

(3) Jagged Array :

```
        System.out.println("Jagged Array…..\n\n");
        for(int i=0;i<a.length;i++)
        {
                for(j=0;j<a[i].length;j++)
                {
                        System.out.println(a[i][j]+" ");
                }
                        System.out.println("\n");
        }
    }
}
```

Operators:

- Meaning : Operator is a symbol that performs some operations on its operands and gives the results.
- Java has a rich set of operators. Operators in java can be divided in main four categories.

  (1) Arithmetic Operators

  (2) Relational Operators

  (3) Logical Operators

  (4) Bitwise Operators

(1) Arithmetic Operators :

- Arithmetic Operators are those which are used in mathematical calculators such as addition, multiplications etc.
- Arithmetic operators can be divided in further sub categories.
    - (1) Basic Arithmetic Operators
    - (2) Increment/Decrement Operators
    - (3) Short-Hand Operators
- In all these arithmetic operators we can use only numeric data type values such as **int** or **float**, but we cannot use **boolean** type values in these operators.

❑ Arithmetic Operators :

- Whereas we can use char type values because in fact **char** is a subtype of int data type.
- **Basic Arithmetic Operators :**

Operator	Description
+	Performs addition of given two operands. Also used as unary plus operator.
-	Performs subtraction of given two operands. Also used unary minus operator.

❑ Arithmetic Operators :

Operator	Description
*	Performs multiplication of given two operands.
/	Performs division of given two operands.
%	Returns the reminder of division (Modules)

❑ Arithmetic Operators :

- All basic arithmetic operators are binary operators and the + and – operators are both binary and unary operators.
- Unary (-) operator simply changes the sign of the operands. [i.e. binary operators take two operands and unary operators take only operands. While ternary operators take three operands.] Note: Ternary operators are discussed later in this chapter.

❑ Arithmetic Operators :

- Example of Arithmetic Operators with unary plus and unary minus.

Def :- Write a program to perform arithmetic calculation using arithmetic operators with Unary plus and Unary minus.

```
class arith_unaryEx
{
   public static void main(String args[])
   {
        int num1=10,num2=20;
        int add,sub,mul,div,unaryPlus,unaryMinus;
```

Arithmetic Operators :

```
    add=num1+num2; //Addition
    sub=num1-num2;  //Subtraction
    mul=num1*num2; //Multiplication
    div=num1/num2;          //Division
    unaryPlus=+num1;//Unary Plus
    unaryMinus=-num2        //Unary Minus
S.o.println("Addition is:"+add);
S.o.println("Subtraction is:"+sub);
S.o.println("Multiplication is:"+mul);
S.o.println("Division is:"+div);
S.o.println("Unary Plus is:"+unaryPlus);
S.o.println("Unary Minus is:"+unaryMinus);        }    }
```

❑ Arithmetic Operators :

Def :- Write a program to perform arithmetic calculation with real number (Floating point) using unary plus/minus operators.

❑ The Modulus Operator :

- The modulus operator in Java is more powerful than other languages.
- Because it can be applied to integer type values as well as floating point values.
- It simply returns the reminder of division operation. Following example shows it:

```
class modulusOp
{
   public static void main(String args[])
   {
```

❑ The Modulus Operator :

```
        int num1=24,imod;
        double num2=39.86,rmod;
        imod=num1%10;
        rmod=num2%10;
        S.O.P("\nInteger reminder is:"+imod);
        S.O.P("\nFloating point reminder is:"+rmod);
    }
}
```

❑ Increment (++) /Decrement(--) Operators:

- Both these operators are unary operators.
- The increment operator increases the value of operand by one and the decrement operator decreases the value of operand by one.

- For Example:

  a=a+1; is equivalent to a++; and

  b=b-1; is equivalent to b--;

Here these operators can be used in one of two forms: Postfix and Prefix.

❑ Increment (++) /Decrement(--) Operators:

- Postfix operator is one which the operator is put after (post) the operand and in Prefix operator the operator is put before (pre) the operand.
- ***For Example:***

x++ is same as ++x and y-- is same as --y

```
int a=10,b=20;
int c=++a;
```

Here value of C will 11.

But, if we use postfix operator, like

```
int d=b++;
```

Then the value of d will be 20.

❑ Increment (++) /Decrement(--) Operators:

- Because in case of postfix operator the increment or decrement the happens after (post) the assignment, and in prefix the increment or decrement happens before assignment takes place. i.e.

❑ c=++a ++a a=a+1 a=10+1

a=11 → c=a → c=11 and a=11 → and

→ →

❑ d=b++ d=b b=20 d=20

b++ → b=b+1 → b=20+1 → b=21 and

d=20 → → → →

❑ Increment (++) /Decrement(--) Operators:

➢ Def : Write a program to print value of variables using increment / decrement operator.

```
class incr_decrEx
{
   public static void main(String args[])
   {
        int a,b,c,d;
        a=b=c=d=10;          OutPut
        S.O.P("a="+a);       a=10
        S.O.P("b="+b);       b=10
        S.O.P("c="+c);       c=10
        S.O.P("d="+d);       d=10

        c=a++;
        d=--b;               OutPut
        S.O.P("a="+a); a=11
        S.O.P("b="+b); b=9
        S.O.P("c="+c); c=11
        S.O.P("d="+d); d=9 }}
```

❑ Short hand Operators:

- Short hand operators are the combination of an arithmetic operator and assignment operator.
- These operators are used because they are shorter to write and faster (efficient) to use.
- Syntax :

  variableName = variableName ope expression;
- Here variableName is any valid variable, ope is any arithmetic operator and expression is any expression.
- **For Example :** x=x+3; is same as x+=3;
- Its advantage can be seen when complex operation is involved.

❑ Short hand Operators:

X=x+(y*z*2.5);

- This statement can be rewrite using short hand operator as follows:

x+=y*z*2.5;

Example:

```
class shorthandOpe
{
   public static void main(String args[])
   {
```

❑ Short hand Operators:

```
        double a,b,c;
        a=24.10;
        b=88;
        c=39.86;
        a+=3.2;
        b%=10;
        c/=2;
        S.O.P("a="+a);
        S.O.P("b="+b);
        S.O.P("c="+c);
    }
}
```

(2) Relational Operators:

- The relational operators in Java are used to check the relation between two operands.
- They return a boolean value i.e. true or false.
- The relational operators are listed below:

Operator	Description
= =	Is equal to
! =	Is not equal to
<	Is less than
>	Is greater than
< =	Is less than or equal to
> =	Is greater than or equal to

(2) Relational Operators:

- These operators are mostly used in if statements and loops to check the condition.

Def : WAP to perform a relational operator's calculations.

```
class relations_opeEx
{
    public static void main(String args[])
    {
        int a,b,c,d;
        a=10,b=20,c=10,d=5;
```

(2) Relational Operators:

```
S.o.p("a="+a+"b="+b+"c="+c+"d="+d+"\n");
S.o.p("a==b is :"+(a==b));
S.o.p("a==c is :"+(a==c));
S.o.p("a>b is :"+(a>b));
S.o.p("a<=b is :"+(a<=b));
S.o.p("a>d is :"+(a>d));      } }
```

Output :

a==b is : false

a==c is : true

a>b is : false

a<=b is : true

a>d is : true

(3) Logical Operators:

- Logical operators in Java are used to combine two or more boolean conditions together.
- Here the operands must be of boolean type variables or conditions and the results returned is also a boolean value.

Operator	Description
&	Logical AND
&&	Short-Circuit AND
|	Logical OR
||	Short-Circuit OR
^	Exclusive OR
!	Logical NOT

(3) Logical Operators:

X	Y	X&Y	X|Y	X^Y	!X	!Y
True	True	True	True	False	False	False
True	False	False	True	True	False	True
False	True	False	True	True	True	False
False	False	False	False	False	True	True

- **The logical AND (&) operators returns true if both the operands are true else returns false.**
- **Whereas the logical OR (|) operator returns false if both the operands are false else returns true. Means it returns true if any one operand (condition) is true.**
- **The Ex-OR (^) operator returns true if from both operands exactly one operand is true (i.e. not zero**

(3) Logical Operators:

- **Def : WAP to demonstrate of simple logical operators.**

```
class logicalOpEx
{
    public static void main(String args[])
    {
        boolean a=(10<5);        // false
        boolean b=('1'=='1');    // true
        boolean c=(5==5);        //true
        S.o.p("a="+a);
        S.o.p("b="+b);
        S.o.p("c="+c);
        S.o.p("a&b="+(a&b));
        S.o.p("a | b="+(a|b));
        S.o.p("a^b="+(a^b));
        S.o.p("b^c="+(b^c));
        S.o.p("!a="+(!a));
    }    }
```

Difference between simple and short-circuit operators:

- The && and || operators are known as short-circuit operators which are not available in other programming languages. There is a minor difference between short-circuit and simple operators.
- From the operation logical AND and logical OR operators we can find that the AND operator evaluates to true if both the operands are true and otherwise false. Means if anyone operand is false then the result is false.

Difference between simple and short-circuit operators:

- And the OR operator evaluates to true if anyone operand is true no matter what the other is.
- The short-circuit operators uses this concept to perform AND / OR operation.
- Short-circuit AND (&&) operator checks the first condition and if it is false then it doesn’t check the next condition, as this will evaluates to false.
- Same as in short-circuit OR (||) operator, on encounter of a true expression it will never check the next condition and will evaluate to true.

Difference between simple and short-circuit operators:

- For example : if(b!=0 **&&** a | b > 5)

```
{
        //statements….
}
```

- Here if ***b*** is **0** then the first condition becomes false so the next expression will never be performed.
- So there is no change of divide by exception. But if this statement have written using & operator then both conditions would have checked.

(4) Bitwise Operators:

- **Bitwise operators can be used with integer type values.**
- **As the name suggest they perform operations on operands bit by bit.**
- **Java has following bitwise operators:**

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR (XOR)
~	Bitwise unary NOT
<<	Bitwise shift left
>>	Bitwise shift right

(4) Bitwise Operators:

- **These operators operate on bits. So all the operands are converted into binary number system first and then the operation is perform on bits.**
- **Bitwise AND, OR, XOR and NOT uses the same logic as the logical operators.**
- **Consider 1 for true and 0 for false.**

X	Y	X&Y	X|Y	X^Y	!X	!Y
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	0	0

(4) Bitwise Operators: (AND &)

- **For example :**

int a=12; // 1100 in binary

int b=10; // 1010 in binary

	= 1 1 0 0	**=12**
	= 1 0 1 0	**=10**
a & b	**= 1 0 0 0**	**= 8**

AND

1 & 1 = 1

1 & 0 = 0

0 & 1 = 0

0 & 0 = 0

(4) Bitwise Operators: (OR |)

- **For example :**

	= 1 1 0 0	= 12
	= 1 0 1 0	= 10
a | b	= 1 1 1 0	= 14

OR

1 | 1	= 1
1 | 0	= 1
0 | 1	= 1
0 | 0	= 0

(4) Bitwise Operators: (XOR ^)

- **Bitwise exclusive OR (^) operator gives 1 if from both operands exactly one operand is one(1) i.e. not both should be 1 or 0.**
- **Or for simplicity one can say that if both operands are same [i.e. both 0 or both 1] then it returns 0 else returns 1.**
- **So if,**

a ^ b = 6.

a =	**1 1 0 1**	**=13**
b =	**1 0 1 1**	**=11**
a ^ b=	**0 1 1 0**	**=6**

(4) Bitwise Operators: (NOT ~)

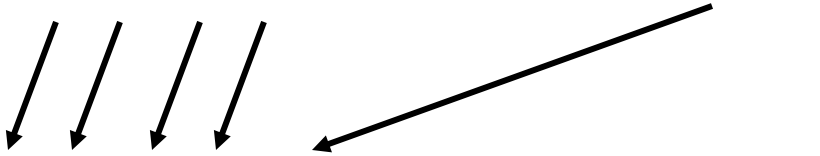
- **Bitwise unary NOT (~) operator simply complements each bit.**
- **Here NOT operator complements each bit, but there is a point which should be kept in mind that in Java int data type is of 32-bits. So ~ operator performs on all 32-bits .**
- **For example byte data type is of 8 bits.**

a	**= 1 0 1 0**	**=10**
a	**= 0 0 0 0 1 0 1 0**	**=10**
~a	**= 1 1 1 1 0 1 0 1**	**!=10**

(4) Bitwise Operators: (Shift Operators [left])

- **Shift left (<<) operator shifts each bit to the left side and the Shift right (>>) operator shifts each bit to the right side.**
- **If a=10 then a<<2 is shifting each bit to left side two times.**

a=10 = 1 0 1 0 extra bit is 0



shift left one time 1 0 1 0 0 =20

Shift left second time 1 0 1 0 0 0 =40

So the a<<2 = 101000=40

(4) Bitwise Operators: (Shift Operators [right])

- **If the a=8;**

a>>1	**0 1 0 0 0 =8**
a>>2	**0 0 1 0 0 =4**
a>>3	**0 0 0 1 0 =2**
a>>4	**0 0 0 0 1 =1**

- **For example:** **SOLVE ERROR**

```
class bitwiseOpe
{
   public static void main(String args[])
   {
```

(4) Bitwise Operators: (Shift Operators [right])

```
int a=12;
int b=10;
S.o.p("a="+a);
S.o.p("b="+b);
S.o.p("a&b="+(a&b);
S.o.p("a|b"+(a|b));
S.o.p("a^b"+(a^b));
S.o.p("~a="+(~a));
S.o.p("a<<2"+(a<<2));
S.o.p("a>>2"+(a>>2));} }
```

Output

```
a=12
b=10
a&b=8
a|b=14
a^b=6
~a=3
a<<2=48
a>>2=3
```

Other Operators :

The Assignment Operator

- **This operator is used to assign some value or expression to a variable. It assigns value from the right side to the left of expression.**
- **For Example :**

```
int i;
i=10;          // 10 is assigned to I
String str;
str="hello";
```

In addition this operator can be used to assign same value to multiple variables.

- **For Example :**

```
int a,b,c,d;
a=b=c=d=0;          // all four variables are initialized to 0.
```

Other Operators :

The Conditional Operator (? :)

- **This ? Is also known as ternary operator.**
- **Syntax:**

  var= (condition) ? (expression1) : (Expression2);
- **Here the first condition is checked. If the condition is true then the expression1 is assigned to the variable.**
- **If the condition if false then the value of expression2 is assigned to variable.**
- **For Example:**

```
class conditionalOpe
{
    public static void main(String args[])
    {
        int a,b,max;
        double per;
```

Other Operators :

The Conditional Operator (? :)

```
        String result;
        a=10;
        b=100;
        per=45.6;
        max=(a>b)?”Pass”:”Fail”;
        System.out.println(“Maximum number is:”+max);
        System.out.println(“Result is:”+result);
    }
}
```

Output

Maximum number is: 100

Result is: Pass

Other Operators :

The instanceof Operator

- **The instanceof operator returns a boolean value, i.e. true or false.**
- **The instanceof is an object reference operator and returns true if the object on the left hand side is an instance of the class given on the right hand side.**
- **This operator allows us to determine whether the object belongs to a particular class of not.**
- **For Example.**

  person instanceof Student is true if the object person belongs to the class Student; otherwise it is false.

Other Operators :

The DOT (.) Operator

- **The dot (.) operator is used to access the instance variables and methods of class objects.**
- **For Example**

  person1.age // reference to the variable age

  person1.salary() // reference to the method salary().
- **It is used to access classes and sub packages from a package.**

Introduction :

1) Selection Statement
   - If
   - Nested If
   - If-Else-If Ladder
   - Switch
2) Iteration Statement
   - While
   - Do While
   - For
3) Jump Statement
   - Break
   - Continue
   - Return

1) Selection Statements :

- Java supports basic two selection statement. If and Switch, while if again divided into three parts.
- These statements allow you to control the flow of your program's execution based upon condition known only during runtime.

1) Selection Statements :

1) IF statement

- The if statement can be use to route program execution through two different paths.

Syntax :

```
if (condition)
{
        Statement1;
}
else
{
        Statement2;
}
```

1) Selection Statements :

1) IF statement (WAP to enter a value and check that a value is positive, negative or zero)

Example :

```
class ifdemo
{
	public static void main(String args[])
	{
		int marks=50;
		String result;
		if(marks>40)
		{
			result="Pass";
		}
		else
		{
			result="Fail";
		}
		System.out.println("Result is :"+result);
	}
}
```

Output :

Result is : Pass

1) Selection Statements :

2) Nested IF statement

- A nested IF is an IF statement that is the target of another IF-ELSE statement.
- Example : **(WAP to enter an age and gender to chech that if age is greater than or equal to 21 and gender is male than print "You Can Marry" otherwise print "You Cannot Marry".**

```
class Nestedifdemo
{
    public static void main(String args[])
    {
            String nm="M;
            int age=21;
```

1) Selection Statements :

2) Nested IF statement

```
if(nm="M")
{
        if(age>=21)
        {
            System.out.println("You can marry");
        }
        else
        {
          System.out.println("You can't marry");
        }
}
    }
}
```

Output : You can marry

1) Selection Statements :

3) IF-ELSE-IF Ladder

- A common programming construct that is based upon a sequence of nested ifs is the IF-ELSE-IF ladder. The IF statements are executed from top to down.
- As soon as on of the conditions controlling the IF is true, the statement associated with the next IF will execute.
- If the none of the condition is true then the final ELSE statement is executed.

1) Selection Statements :

3) IF-ELSE-IF Ladder

- **Syntax :**

```
if(condition1)
{
        Statement1;
}
else if(condition2)
{
        Statement2;
}
else if(condition3)
{
        Statement3;
}
……
……
……
else
Statement N;
```

1) Selection Statements :

3) IF-ELSE-IF Ladder

- **Example : (WAP to create a student result with five subjects and print it's total & percentage with appropriate message)**

```
class if_else_ifdemo
{
    public static void main(String args[])
    {
            String result;
            int marks=65;
            if(marks>=75)
            {
                    result="Distiction";
            }
```

1) Selection Statements :

3) IF-ELSE-IF Ladder

- **Example :**

```
else if(marks>=60)
{
        result="First Class";
}
else if(marks>=50)
{
        result="Second Class";
}
else
{
        result="Third class";
}
```

(WAP to find out maximum number out of given 3 numbers).

1) Selection Statements :

- **Switch**
- The SWITCH statement in Java's multi way branch statement.
- It provide easy way to dispatch execution to different parts of your code based on the value of an expression.
- Syntax :

```
switch(expression)
{   case value1:
        Statement1;
        break;
```

1) Selection Statements :

- **Switch**

```
        case value2:
                statement2;
                break;
        case valueN:
                statementN
                break;
        default:
                Default statement;
}
```

1) Selection Statements :

- **Switch Example: (WAP to enter a number of day and print name of the day using switch statement).**

```
class switch_demo
{
    public static void main(String args[])
    {
        int day=2;
        switch(day)
        {
            case1:
                S.O.P("Monday");
                break;
```

1) Selection Statements :

- **Switch Example:**

```
        case2:
                S.O.P("Tuesday");
                break;
        case3:
                S.O.P("Wednesday");
                break;
        case4:
                S.O.P("Thursday");
                break;
        case5:
                S.O.P("Friday");
                break;
        case6:
                S.O.P("Saturday");
                break;
```

1) Selection Statements :

- **Switch Example:**

```
            case7:
                S.O.P("Sunday");
                break;
            default:
                S.O.P("No match");
                break;
            }
        }
}
```

Output : Tuesday

2) Iteration Statements :

- Java supports basic THREE iteration statement, WHILE, DO-WHILE and FOR.
- These statements create commonly called LOOPS.

1) While:

- While loop will check the condition first.
- If the condition will be true then it will execute the statements given in the loop, after once complete the loop, it will again and again check the condition if condition will false then it will auto exit for the loop.

- **Syntax :**

```
while(test condition)
{
    statements;
}
```

- **Example :**

```
int a=10;
while(a>=1)
{
    Syatem.out.println(“A is:”+a);
    a=a+1;  }
```

2) Iteration Statements :

2) For loop:

- To provide a looping while condition will be true we can use for loop.

❖ **Syntax :**

```
for(initialization; condition; increment or
    decrement)
{
    statements;
}
```

initialization : Initialize the value of variable.

test Condition : Test condition at here.

Increment/decrement: To increase or decrease the value.

❖ **Example :**

```
int a=1;
for($a=1;$a<=10;$a++)
System.out.println("A is"+a);
```

2) Iteration Statements :

3) Do-while loop:

- Do…while loop will check the condition after execution of statements provided in to loop.
- If the condition will be true then it will again execute the statements given in the loop, if condition will false then it will auto exit for the loop.

❖ **Syntax :**

```
do
{
        statements;
} while(test condition);
```

❖ **Example :**

```
int a=10;
do
{
    System.out.println("A is:"+a);
    a=a+1;
}
while(a<10);
```

3) Jump Statements :

1) Break:

- The break command will break the loop.

❖ **Syntax :**

```
break;
```

❖ **Example :**

```
for(int i=1;i<=10;i++)
{       if(i==5)
                break;
        System.out.println("Num. is "+i);
}
```

3) Jump Statements :

2) Continue:

- The continue command will break the current loop and continue with the next loop.

❖ **Syntax :**

```
continue;
```

❖ **Example :**

```
for(int i=1;i<=10;i++)
{       if(i==5)
                continue;
        System.out.println("Num. is "+i);   }
```

3) Jump Statements :

3) Return:

- The return statement cause the control to be transferred back from a method to the caller of the method.

❖**Syntax :**

```
return;
```

❖**Example :**

```
int i=1
While(i<10)
{       if(i==3)
                return;
```

Points of learning :

- Classes
- Objects
- New Keyword
- Adding Methods to class
- Constructors
- Use of this keyword
- Access Protection

Points of learning :

- Nested class and inner class
- Garbage collection and finalize() method
- Command-line arguments

The General Form of a class

- A class is a template from which objects are created. That is objects are instance of a class.

- When you create a class, you are creating a new data-type. You can use this type to declare objects of that type.

- Class defines structure and behavior (data & code) that will be shared by a set of objects

- A class is declared by use of the class keyword. Classes may contain data and code both.

Syntax of Class

```
class ClassName
   {
        type variable2;
        type variable2;
        type methodname1 (parameter list)
        {
                body of method;
        }
        type methodname2 (parameter list)
        {
                body of method;
        }       }
```

Simple Example of Class

```
class Box
{
   double width;
   double height;
   double depth;
}
class BoxDemo
{
   public static void main (String args[])
   {
        Box mybox = new Box ();
```

Simple Example of Class

```
        double vol;
        mybox.width =10;
        mybox.height = 20;
        mybox.depth = 30;
        vol = mybox.width * mybox.height *
        mybox.depth;
        System.out.println ("Volume is: - "+vol);
    }
}
```

Creating Objects

- Here, the class Box has three instance variable. So after creating its object we can access these variables. The objects of class is declared by the statement :

- **Declaring Object** :-
- Box b1=new Box(); // this is a combination of two statements
- **Box mybox;** // declare ref. to object which contains null value.

- **mybox = new Box ();** // allocate a Box object.

New Keyword

- Here, the new keyword is used to allocate memory equal to the size of instance variables of class Banking. Before using new keyword the object can not be used.

- The new keyword allocates memory dynamically-at run-time to objects. So you can create as many objects as you want.
- You can assign a reference to object :
  - Banking b1=new Banking();
  - Banking ref=b1;

Adding Methods to class

- So far we added instance variables to class, now we will see how to add methods to a class. Main strength of a class is in its methods.

- Following example adds a method of banking class (adding method to Banking class) :

```
Class Banking {
    double p,r;
    int r;
    void simpleIntrest()
    { double si;
```

Example of Adding Methods to class

```
    Si=p*r*n/100;
    System.out.println("Simple Intrest is:"+si);
    }
}
    class simpleclass2 {
    public static void main(string args[]) {
    Banking b1=new Banking();
    b1.p=5000;
    b1.r=12;
    b1.n=5;
```

Example of Adding Methods to class

```
    b1.simpleIntrest(); // calling method on object b1
    Banking b2=new Banking();
    b2.p=10000;
    b2.r=10;
    b2.n=4;
    b2.simpleIntrest() ; // calling method on object b2
    }
}
```

Constructors

- The constructor is a special method which has the same name as its class. It initializes the objects of its class automatically at the time of creation.

- One important point about constructor is that they return type. Not even void. This is because they actually return the object of its class.

- When we create an object, we call the constructor after new keyword.

Example of Constructors

- // adding constructor to Banking class

```
Class Banking {
   double p,r;
   int n;
   Banking() {
   System.out.println("Executing constructor");
   p=10000;
   r=12;
   n=5;
   }
```

Example of Constructors

```
double simpleIntrest()
{ return(p*r*n/100);
class constructorEx  {
public static void main(String args[])
{
Banking b1=new Banking(); // calling constructor
double si=b1.simpleIntrest(); // method on b1
System.out.println("Simple intrest of b1 is"+si);
Banking b2=new Banking();
System.out.println("Simple intrest of b2
is:"b2.simpleIntrest());     } }
```

Access Protection

- In a class you can specify which member of your class can be accessed by other veriables.
- To do this, java provides three access specifier, public,private and protected.
- **1) Public :** When a member is defined by public specifier, it can be accessed from anywhere.
- Public keyword is used to declare a member as public.

Access Protection

- **2) Protected :** This specifier is used in inheritance only. The protected members can be accessed by the members of other package, but only to the subclass of this class.

- **3) Private :** A member modified as private can be accessed only by the members of its class. Private keyword is used to declare a member as private.

- **4) Default :** When you do not specify any access specifier, the default specifier is applied to that member.

Example of Access Protection

```
Class simple
{
    int i;
    public int j; //  Public access
    private int k; // specify Private access

}
```

Garbage Collection and finalize() method

- When you create an object using new operator, the memory is allocated dynamically to the object. So if we create more than one object, memory is allocated to those objects.

- If object goes out-of scope, then memory should be deallocated to that object.

- The process of deallocating memory for objects is known as garbage collection.

Finalize() method

- Sometimes you will need to do some actions when an objects is destroyed by garbage collection.

- For example when your program is using non-java resources, you have to free the memory for these resources. This process is known as finalization.

```
Protected void finalize()
{ // statements to be executed when finalization occurs...
   }
```

Command-line arguments

- You can pass arguments to the main(). This can done at run-time. As we know that the main() takes array of String as argument.

- **For example : // passing arguments to main()...**

```
Class cmdline {
   public static void main(String arg[]) {
        for(int i=0;i<arg.length;i++)
                System.out.println("Argument"+i+":"+arg[i]);
   } }
```