

UNIT-1

Introduction to OOP and Java Basics

History of Java:

- Java is an efficient powerful Object-Oriented Programming language developed in the year of **1991** by **James Gosling** and his team members at Sun micro systems.
- **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- Initially Java is called with a name **Oak** is a symbol of strength and choosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.
- In 1995, Oak was renamed as "**Java**".
- Java is an island of Indonesia where first coffee was produced (called java coffee).
- Sun micro systems purchased by Oracle Corporation in the year of 2010.
- JDK (Java Development tool Kit) 1.0 released in (January 23, 1996).

Java Version History

There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)

Where it is used? According to Sun, 3 billion devices run java. There are many devices where java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus etc.
2. Web Applications such as irctc.co.in, javatpoint.com etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games etc.

Types of Java Applications

There are mainly 4 type of applications that can be created using java programming:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

4) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

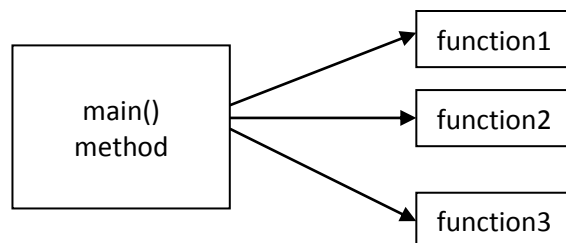
-----*****-----

Need for Object Oriented Programming paradigm:

- Computer programs consist of two elements: code and data. Furthermore a program can be conceptually organized around its code or around its data.
- These are the two paradigms that govern how a program is constructed.
 - 1) Procedure-Oriented Programming model.
 - 2) Object-Oriented Programming model.

Procedure Oriented Programming(POP):

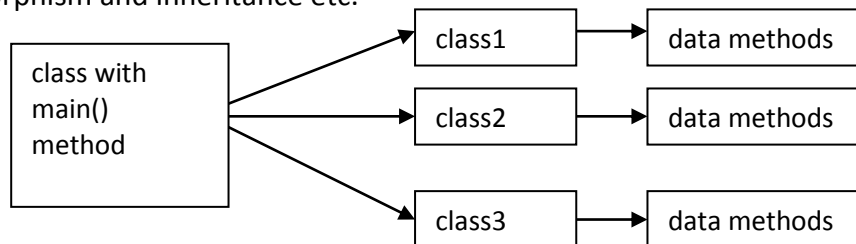
- In the Procedure oriented programming model importance is not given to data but to functions as well as sequence of actions to be done.
- C, Pascal, Fortran etc are called procedure oriented programming languages.
- In POP a programmer uses procedures or functions to perform a task.
- When the programmer wants to write a program, he will first divide the task into separate sub tasks i.e as functions.
- In C programming several functions are controlled from a main() function.



- Even though POP is a efficient approach but if program size increases the Complexity (readability is difficult) also increased.
- In the POP approach there is no data hiding concept i.e security for the data is less.

Object Oriented Programming(OOP):

- In OOP, Importance is given to the data rather than procedures or functions because it works as a real world.
- In the object oriented programming classes and objects are used inside programs.
- A class is a module or template contains data and methods (functions) to achieve the task.
- The main task is divided into several modules , these modules are called classes, each class can perform some tasks for which several methods are written in a class.
- In the OOP approach we can achieve security for the data using data hiding concept.
- In java we can develop efficient programs using the concepts encapsulation, polymorphism and inheritance etc.



-----*****-----

Difference Between Procedure Oriented Programming (POP) & Object Oriented Programming (OOP):

	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into small parts called functions .	In OOP, program is divided into parts called objects .
Importance	In POP, Importance is not given to data but to functions as well as sequence of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a real world .
Approach	POP follows Top Down approach .	OOP follows Bottom Up approach .
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named public, private, protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.

Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function, it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is less secure .	OOP provides Data Hiding so provides more security .
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Examples of POP are: C, VB, FORTRAN and Pascal.	Examples of OOP are: C++, JAVA, VB.NET, C#.NET.

-----*****-----

Principles of OOP:

The object-oriented paradigm supports four major principles, they are also known as four pillars of the object-oriented paradigm.

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

Abstraction:

- Abstraction means hiding the implementation details from the user and gives only functionality details.
- For example, when you drive your car you do not have to be concerned with the exact internal working of your car. What you are concerned with is interacting with your car via its interfaces like steering wheel, brake pedal, accelerator pedal etc. Here the knowledge you have of your car is abstract.
- In java abstraction can be achieved using abstract class and interfaces.

Encapsulation:

- Wrapping data(variables) and methods(code acting on data) within classes in combination with implementation hiding (through access control) is often called encapsulation.
- In java encapsulation can be achieved using classes.
- Using this encapsulation we can provide security for the data inside a class using object reference, this is called 'Data hiding'.

- **Example:**

```
class Sample
{
    int a;
    float b;
    public void add()
    {
        //body of method
    }
}
```

Inheritance:

- Inheritance in java is a mechanism in which one object acquires the properties and behaviors of parent object. It's essentially creating parent-child relationship between classes. In java, you will use inheritance mainly for code re-usability and maintainability.
- In java inheritance is also called as IS-A relation ship, and it can be achieved using “**extends**” key word.

- **Example:**

```
class A
{
    //members of a class A
}
class B extends A
{
    // members of class B
}
```

Polymorphism:

- The word polymorphism came from two greek words ‘poly’ meaning ‘many’ and ‘morphos’ meaning ‘forms’.
- Polymorphism represents the ability to assume several different forms.
- Polymorphism is the ability by which, we can create functions or reference variables which behaves differently in different programmatic context.
- In java language, polymorphism is essentially considered into two versions:
 - Compile time polymorphism (static binding or method overloading)
 - Runtime polymorphism (dynamic binding or method overriding)

-----*****-----

Java Characteristics (or) Features of java (or) Java Buzz words:

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

Simple

According to Sun, Java language is simple because:

syntax is based on C++ (so easier for programmers to learn it after C++).

removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.

No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Object-oriented

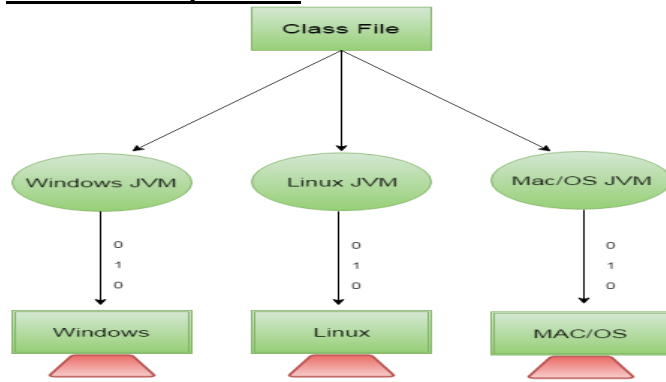
Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation.

Platform Independent



A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

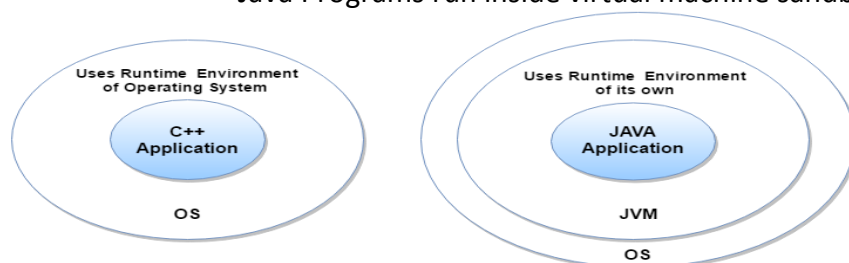
- Runtime Environment
- API(Application Programming Interface)

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Secured

Java is secured because:

- No explicit pointer
- Java Programs run inside virtual machine sandbox



Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

Architecture-neutral

There are no implementation dependent features e.g. size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

Portable: We may carry the java bytecode to any platform.

High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

*****_*****

Java program Compilation and Execution Process:

Sample java program:

```
import java.lang.*; // package import section
class Sample // creating a class
{
    public static void main(String args[]) // main() execution starts from here
    {
        System.out.println("Hello world"); // writing "Hello world" on console
    }
}
```

- Save the above program with an extension .java i.e Sample.java in a specific location.

Compilation:

- for compilation of java program we use "javac" keyword.

javac Sample.java

Execution:

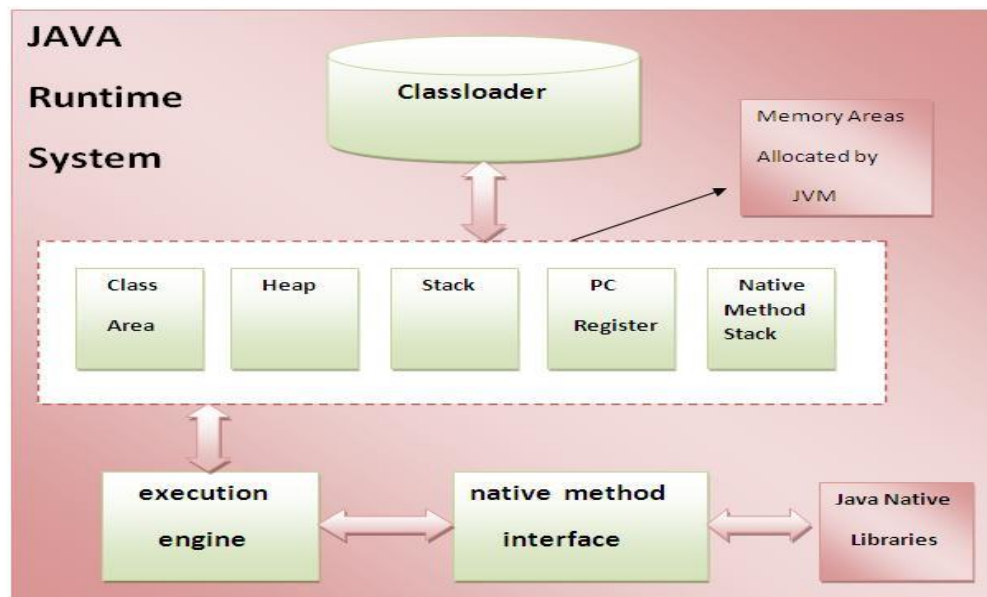
- After successful compilation of java program .class file will be created. This is called bytecode.
- _for executing java program we use .class file name with "java" keyword as follows.



*****_*****

Java Virtual Machine(JVM):

- JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
- JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).
- The JVM performs following operations: Loads code, Verifies code, Executes code and Provides runtime environment
- **Internal Architecture of JVM:**



Classloader: Classloader is a subsystem of JVM that is used to load class files.

Class(Method) Area: Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

Heap: It is the runtime data area in which objects are allocated.

Stack: Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

Program Counter Register: PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

Native Method Stack: It contains all the native methods used in the application.

Execution Engine: It contains

- A virtual processor
- Interpreter: Read byte code stream then execute the instructions.

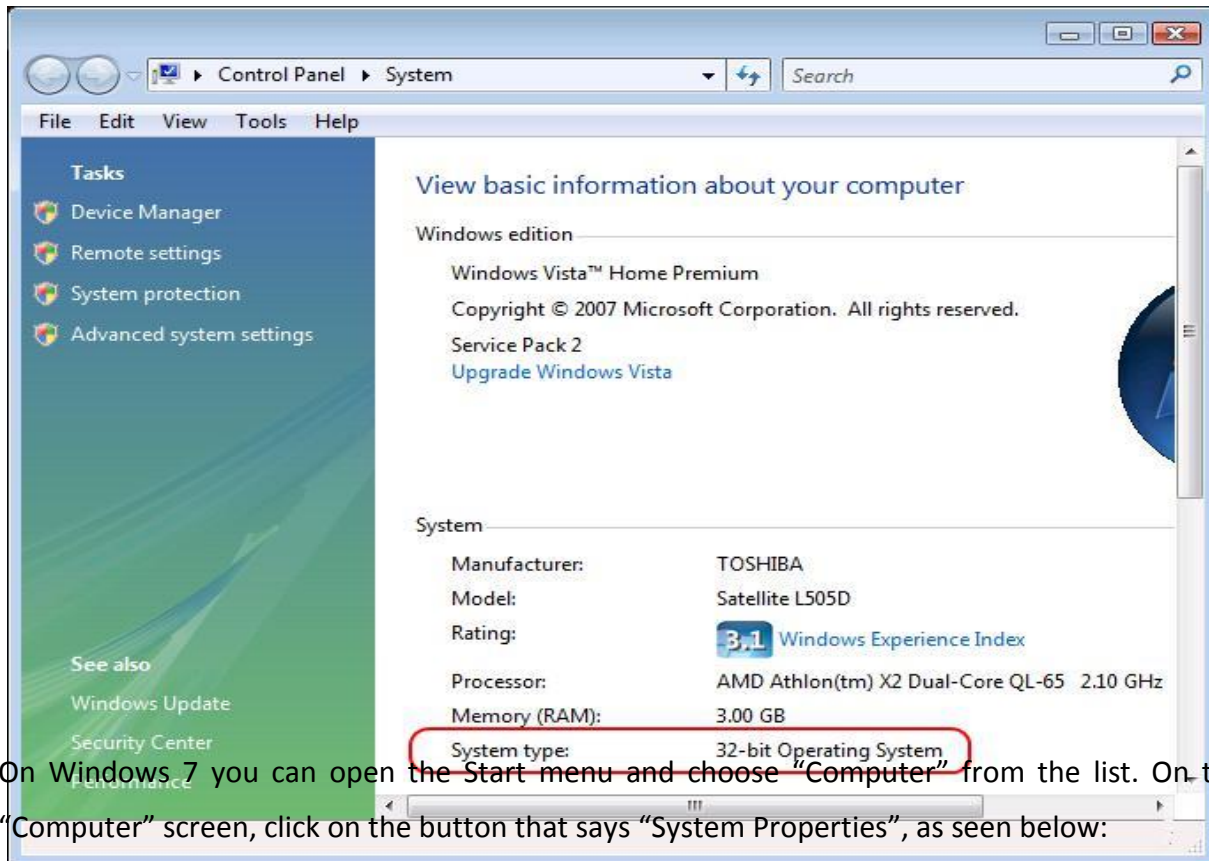
- Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and Hence reduces the amount of time needed for compilation. Here the term compiler refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

*****_*****

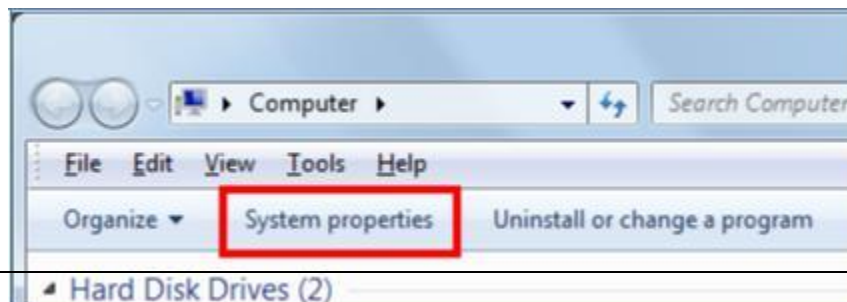
Installation of JDK:

The JDK software has two different versions: a 32-bit Windows version and a 64-bit Windows version. Before you download the software, you will need to know which version of Windows you are using. If you don't already know what version of Windows you have, you can find out fairly quickly!

On older systems find your “Computer” icon from the Start menu or on your desktop. Right-click on this icon and select “Properties”. You can then read the “System Type” to determine if you have a 32 or 64-bit operating system



On Windows 7 you can open the Start menu and choose “Computer” from the list. On the “Computer” screen, click on the button that says “System Properties”, as seen below:



This will bring up a screen with information about your computer and your version of Windows. Look for a section titled “System”. This section will easily tell you what version of Windows you have installed next to the “System Type” label:



Keep this information handy! We will use it to determine which version of the JDK to download.

Downloading the JDK

To download the *Java Development Kit (JDK)*, launch your web browser (e.g. Internet Explorer) and goto address: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. This page shows many download options. The top of the page shows the most common JDK download options:



For this course, you can select either JDK version 6 (which we have chosen in the textbook) or JDK version 7 (which is newly released). Select the “JDK” download button for the corresponding version further down on the screen:

Here are the Java SE downloads in detail:



Or if you prefer JDK 6.0 (1.6):

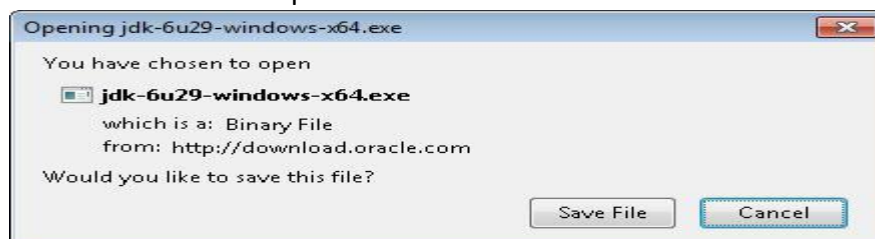


The Download button will open another page which will list the various JDK installation files. You must choose “Accept License Agreement” in order to continue with the download and installation.



Once you have accepted the license agreement, you will need to choose a file to download. The Windows files are at the bottom of the screen. If you have a 32-bit Windows system, you will choose the “Windows x86” file. If you have a 64-bit Windows system, you will need to choose the “Windows x64” file.

As soon as you click to download the file, a pop-up window will appear asking you to either “Save” or “Run” the program. The look of this window will depend on what version of Windows and which Internet browser you are using to download the file. The following screenshots are from Mozilla Firefox and Internet Explorer:



Select “Save File” or “Save” to save the file to a location on your local hard drive. You can save it to your Desktop or some other file folder. Remember this location so you can find it later!

Oracle updates the exact version of the JDK frequently. Our examples show JDK version 1.6.29 (6.0.29), but keep in mind that the version available to you at the time of download will likely be different! Or you may choose JDK 1.7.X (7.0.X) if you prefer.

Once the file is saved, use your Windows Explorer to find and run the program by double-clicking in it. Depending on your version of Windows and security settings you may get a security popup as shown below. Click on “Run” to continue.

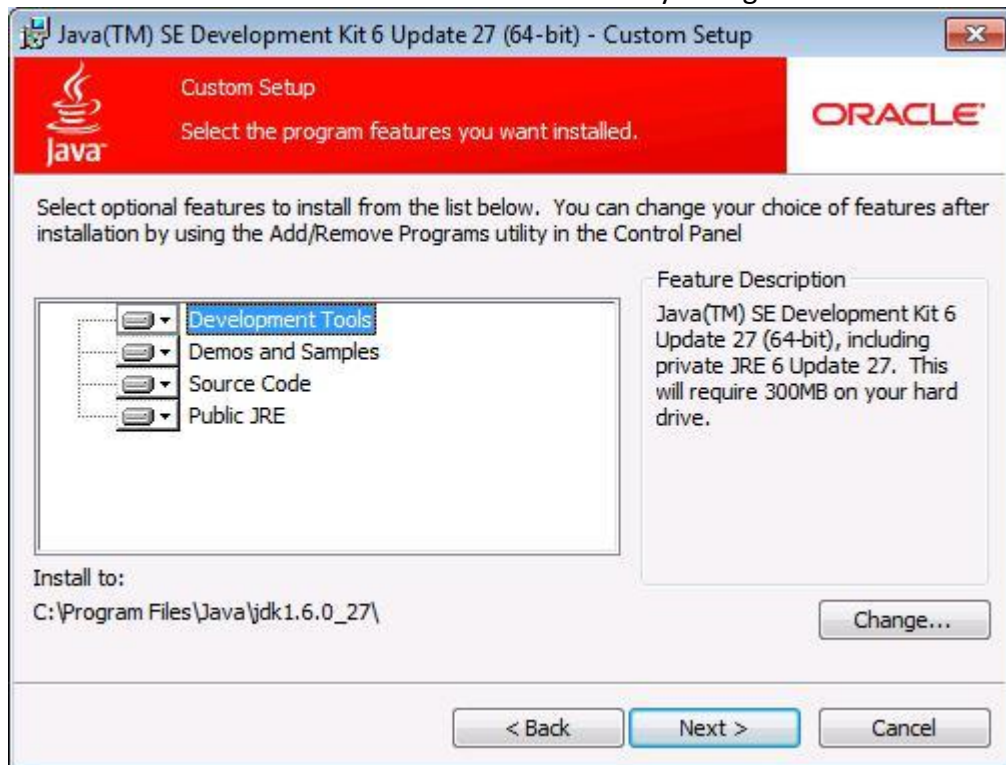


When setup is launched you should see the following screen:

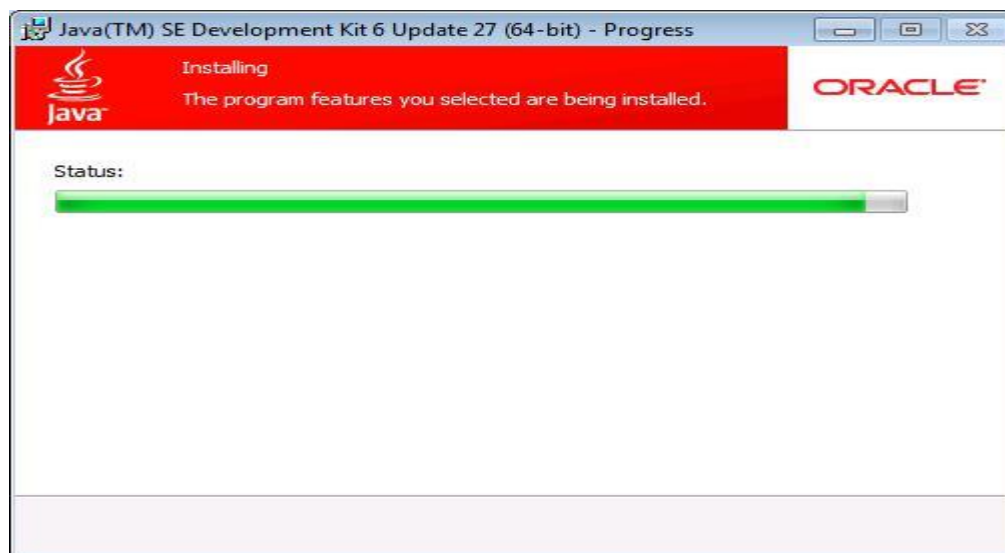


This is the first screen in the install process. Click “Next” to continue.

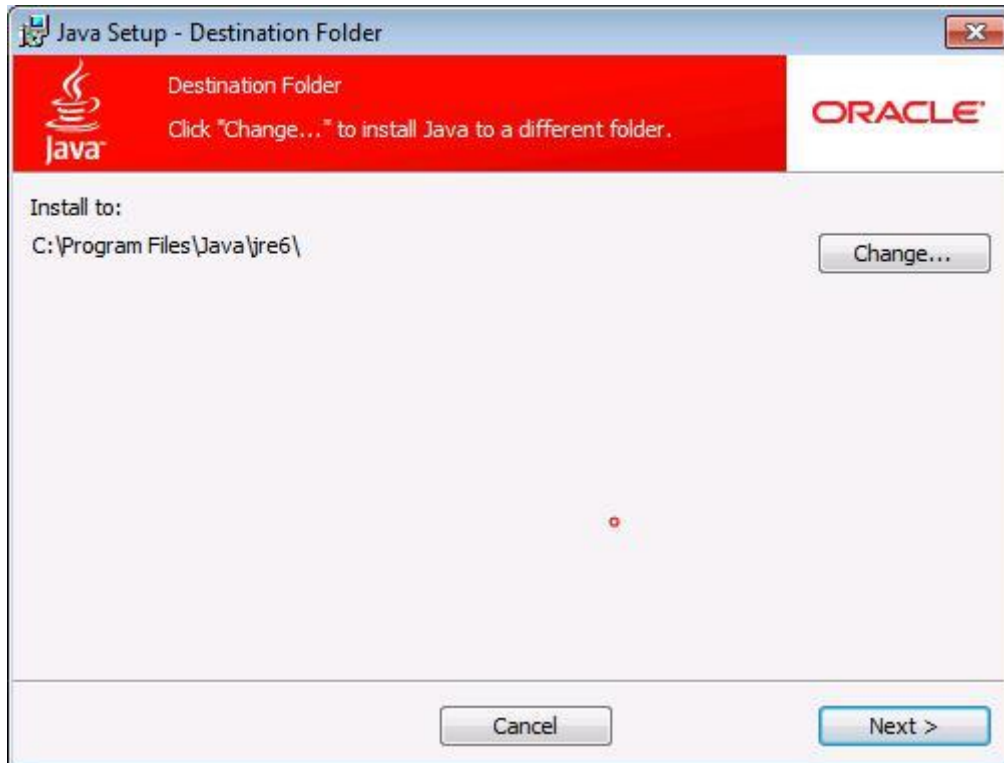
This next screen lists all of the possible JDK options that can be installed. Since we will be covering the basics of Java in this course, you can just accept the defaults and simply click on the 'Next' button to continue. There is no need to make any changes on this screen.



The next screen will display a simple progress bar while the JDK files are being installed. This process could take anywhere from seconds to minutes, depending on the speed of your computer.



When the JDK is finished installing, the installation program will install the JRE files. The screen above will allow you to choose the directory where the JRE will be located. We recommend that you allow the files to install in the default directory, as shown below.

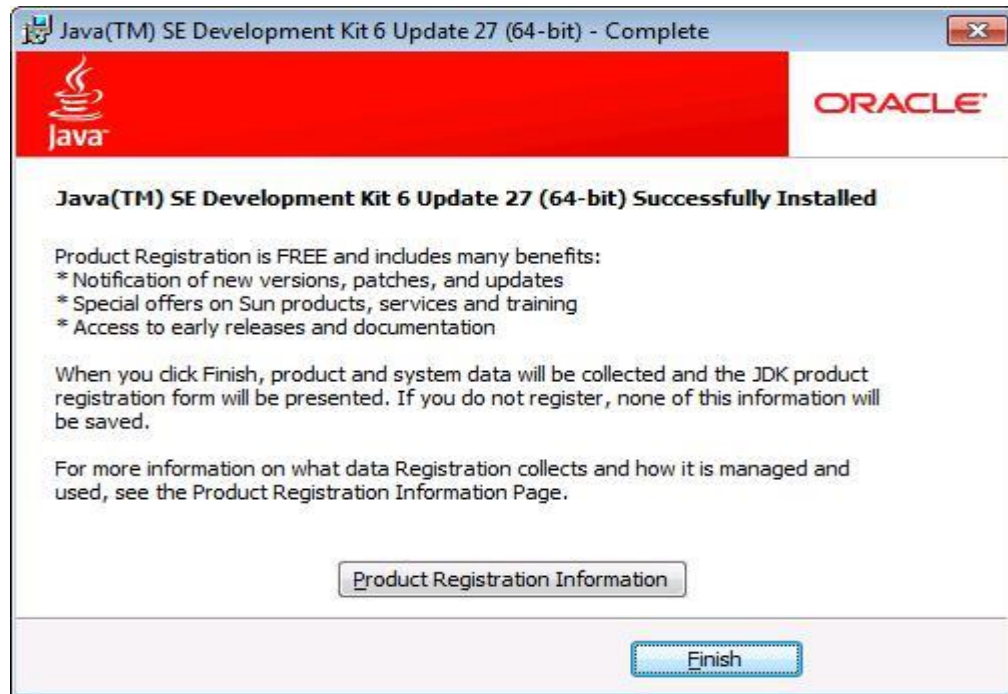


Once you choose the “Next” button, the installation will display another progress bar. This will show the progress of the installation of the JRE files.

The next screen will simply show the progress of your JRE installation. In this first step, the installation program will automatically download additional files from the Oracle website. This is a large program and will take some time!



At this point, the installation of the JDK files is complete. When you click on the “Finish” button on this screen, a browser window will appear, displaying registration information for Java.



Registration for the JDK software is optional and is not necessary for the completion of this course. If you choose not to register, simply close this window.



Congratulations! You have finished the installation of the JDK and JRE in your Windows computer.

*****_*****

Java Basics :

Identifiers:

- Identifiers are used for class names, method names, and variable names.
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- They must not begin with a number, lest they be confused with a numeric literal. Again, Java is case-sensitive, so **VALUE** is a different identifier than **Value**. Some examples of **valid identifiers** are:

AvgTemp count a4 \$test this_is_ok

- **Invalid variable names** include:

2count high-temp Not/ok

Variables:

- The variable is the basic unit of storage in a Java program.
- A variable is defined by the combination of an identifier, a type, and an optional initializer.
- In addition, all variables have a scope, which defines their visibility, and a lifetime.
- Declaring a Variable:

In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:

type identifier [= value], identifier [= value], ... ;

- Here are several examples of variable declarations of various types. Note that some include an initialization.

```
int a, b, c;           // declares three ints, a, b, and c.
int d = 3, e, f = 5;   // declares three more ints, initializing d and f.
byte z = 22;          // initializes z.
double pi = 3.14159;  // declares an approximation of pi.
char x = 'x';         // the variable x has the value 'x'.
```

- Types of Variable:

There are three types of variables in java:

- **local variable:** declared inside the method is called local variable.
- **instance variable:** declared inside the class but outside the method, is called instance variable . It is not declared as static.
- **static variable:** declared as static is called static variable. It cannot be local.

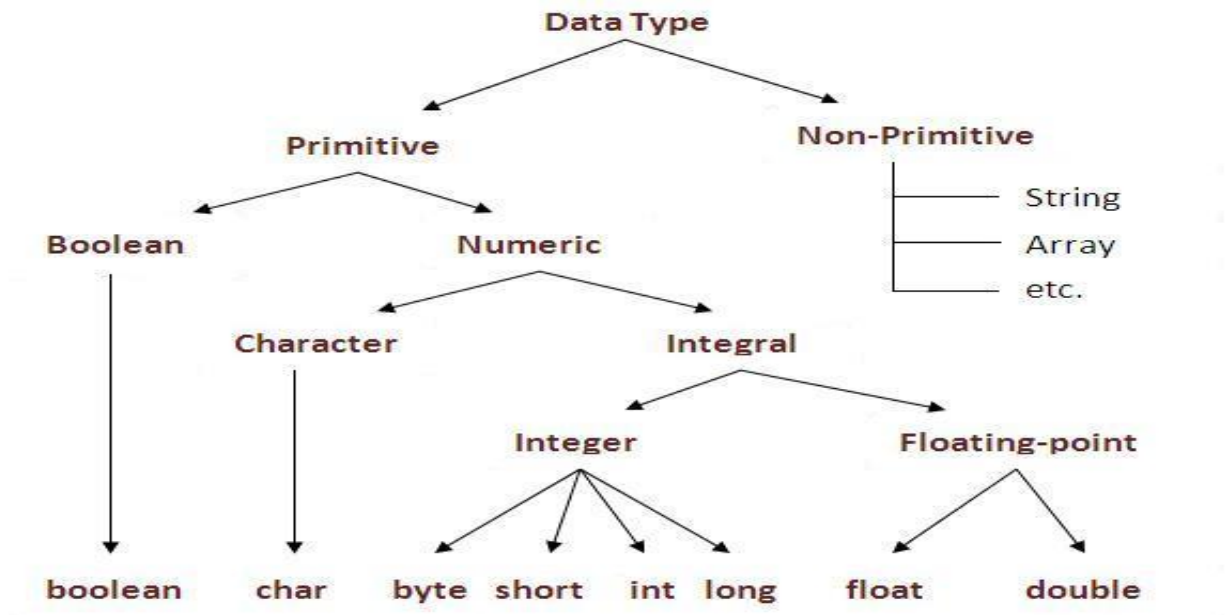
Example:

```
class A
{
    int data=50;//instance variable
    static int m=100;//static variable
    void method()
    {
        int n=90;//local variable
    }
}
```

Data types:

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive data types
- Non-primitive data types



Data Type	Default Value	Default size	Range
boolean	false	1 bit	-
char	'\u0000'	2 byte	0 to 65,536
byte	0	1 byte	-128 to 127
short	0	2 byte	-32,768 to 32,767
int	0	4 byte	-2,147,483,648 to 2,147,483,647
long	0L	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	0.0f	4 byte	1.4e-045 to 3.4e+038
double	0.0d	8 byte	4.9e-324 to 1.8e+308

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system than ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

Example on variables:

```
class Simple
{
    public static void main(String[] args)
    {
        int a=10;
        int b=10;
        int c=a+b;
        float f1=10.25f;
        double d=123.4567;
        char ch='a';
        boolean b1=true;
        boolean b2=false;
        System.out.println("int c="+c);
        System.out.println("float f1="+f1);
        System.out.println("double d =" +d);
        System.out.println("char ch =" +ch);
        System.out.println("boolean b1="+ b1);
        System.out.println("boolean b2="+ b2);
    }
}
```

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:
 byte a = 68;
 char a = 'A'
- byte, int, long, and short can be expressed in decimal(base 10), hexadecimal(base 16) or octal(base 8) number systems as well.
- Prefix 0 is used to indicate octal, and prefix 0x indicates hexadecimal when using these number systems for literals. For example:
 int decimal = 100;
 int octal = 0144;
 int hexa = 0x64;
- 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:
 "Hello World"
 "two\nlines"
 "\\"This is in quotes\\""
- String and char types of literals can contain any Unicode characters. For example:
 char a = '\u0001';
 String a = "\u0001";

- In java we have some escape sequences.

Escape Sequence	Description
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)
\'	Single quote
\"	Double quote
\\	Backslash
\r	Carriage return
\n	New line (also known as line feed)
\f	Form feed
\t	Tab
\b	Backspace

Keywords:

- There are 49 reserved keywords currently defined in the Java language .
- These keywords, combined with the syntax of the operators and separators, form the definition of the Java language.
- These keywords cannot be used as names for a variable,class, or method.

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

- The keywords **const** and **goto** are reserved but not used. In the early days of Java, several other keywords were reserved for possible future use.
- However, the current specification for Java only defines the keywords shown above.
- The **assert** keyword was added by Java 2, version 1.4
- In addition to the keywords, Java reserves the following: **true**, **false**, and **null**. These are values defined by Java. You may not use these words for the names of variables, classes, and so on.

Operators:

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

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Boolean Logical Operators

Arithmetic Operators:

- Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.
- The following table lists the arithmetic operators:

<u>Operator</u>	<u>Result</u>
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement

- The operands of the arithmetic operators must be of a numeric type. You cannot use them on **boolean** types, but you can use them on **char** types, since the **char** type in Java is, essentially, a subset of **int**.

- Example program:

```
class Test
{
    public static void main(String args[])
    {
        int a = 10;
        int b = 20;
        int c = 25;
        int d = 25;
        System.out.println("a + b = " + (a + b) );
        System.out.println("a - b = " + (a - b) );
        System.out.println("a * b = " + (a * b) );
        System.out.println("b / a = " + (b / a) );
        System.out.println("b % a = " + (b % a) );
        System.out.println("c % a = " + (c % a) );
        System.out.println("a++ = " + (a++) );
    }
}
```

```

        System.out.println("b-- = " + (a--) );
        // Check the difference in d++ and ++d
        System.out.println("d++ = " + (d++) );
        System.out.println("++d = " + (++d) );
    }
}

```

This will produce the following result:

```

a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++ = 10
b-- = 11
d++ = 25
++d = 27

```

Compound assignment:

- Compound assignment operators are +=, -=, *=, /=, %=
- Examples:

```

int a=10;   (or)   int a=10;           int a=20;   (or)   int a=20;
a=a+10;     a+=10;           a=a/10;       a/=10;

```

Relational Operators:

- The *relational operators* determine the relationship that one operand has to the other. Specifically, they determine equality and ordering.
- The relational operators are shown here:

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

- The outcome of these operations is a **boolean** value. The relational operators are most frequently used in the expressions that control the **if** statement and the various loop statements.
- Any type in Java, including integers, floating-point numbers, characters, and Booleans can be compared using the equality test, ==, and the inequality test, !=.

```

int a = 4;
int b = 1;
boolean c = a < b; // c is false

```

- Example program:

```

class Test
{
    public static void main(String args[])
    {
        int a = 10;
        int b = 20;
    }
}

```

```

        System.out.println("a == b = " + (a == b) );
        System.out.println("a != b = " + (a != b) );
        System.out.println("a > b = " + (a > b) );
        System.out.println("a < b = " + (a < b) );
        System.out.println("b >= a = " + (b >= a) );
        System.out.println("b <= a = " + (b <= a) );
    }
}

```

Output:

```

a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false

```

Bit wise operators:

- Java defines several *bitwise operators* which can be applied to the integer types, **long**, **int**, **short**, **char**, and **byte**. These operators act upon the individual bits of their operands.
- They are summarized in the following table:

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
=	Bitwise OR assignment

BITWISE LOGICAL OPERATORS

- The bitwise logical operators are **&**, **|**, **^**, and **~**.
- The bitwise operators are applied to each individual bit within each operand.

A	B	A B	A & B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

- The Bitwise NOT**

Also called the *bitwise complement*, the unary NOT operator, **~**, inverts all of the bits of its operand. For example,

00101010 becomes 11010101 (after the NOT operator is applied.)

- The Bitwise AND**

The AND operator, **&**, produces a 1 bit if both operands are also 1. A zero is produced in all other cases. Here is an example:

```

    00101010    42
& 00001111    15
-----
    00001010    10

```

- **The Bitwise OR**

The OR operator, `|`, combines bits such that if either of the bits in the operands is a 1, then the resultant bit is a 1, as shown here:

```

    00101010    42
| 00001111    15
-----
    00101111    47

```

- **The Bitwise XOR**

The XOR operator, `^`, combines bits such that if exactly one operand is 1, then the result is 1. Otherwise, the result is zero.

```

    00101010    42
^ 00001111    15
-----
    00100101    37

```

Left Shift

- The left shift operator, `<<`, shifts all of the bits in a value to the left a specified number of times.
- It has this general form:

$$value \ll num$$
- Here, *num* specifies the number of positions to left-shift the value in *value*. That is, the `<<` moves all of the bits in the specified value to the left by the number of bit positions specified by *num*.
- For each shift left, the high-order bit is shifted out (and lost), and a zero is brought in on the right.

```

byte a = 64, b;
int i;
i = a << 1;
b = (byte) (a << 2);

```

After shifting operation `i = 128` and `b = 0` (because byte takes only 8-bits)

`a << 1` : a is left shifted by one position

	256	128	64	32	16	8	4	2	1
<code>a = (64)</code>			1	0	0	0	0	0	0
<code>i = (a << 1)</code>		1	0	0	0	0	0	0	0

`a << 2` : a is left shifted by 2 positions

	256	128	64	32	16	8	4	2	1
<code>a = (64)</code>			1	0	0	0	0	0	0
<code>b = (byte)(a << 2)</code>	1	0	0	0	0	0	0	0	0

Right Shift

- The right shift operator, `>>`, shifts all of the bits in a value to the right a specified number of times.
- Its general form is shown here:
$$value \gg num$$
- Here, *num* specifies the number of positions to right-shift the value in *value*.
- That is, the `>>` moves all of the bits in the specified value to the right the number of bit positions specified by *num*.
- The following code fragment shifts the value 32 to the right by two positions, resulting in **a** being set to 8:

```
int a = 32;  
a = a >> 2; // a now contains 8
```

a>>2 : a is right shifted by two positions

		64	32	16	8	4	2	1
a=32			1	0	0	0	0	0
a=a>>2					1	0	0	0

Bitwise Operator Assignments:

- All of the binary bitwise operators have a shorthand form similar to that of the algebraic operators, which combines the assignment with the bitwise operation.
- For example, the following two statements, which shift the value in **a** right by four bits, are equivalent:

```
a = a >> 4;  
a >>= 4;
```

- Likewise, the following two statements, which result in **a** being assigned the bitwise expression **a OR b**, are equivalent:

```
a = a | b;  
a |= b;
```

- Example program:

```
class OpBitEquals  
{  
    public static void main(String args[])  
    {  
        int a = 1;  
        int b = 2;  
        int c = 3;  
        a |= 4;  
        b >>= 1;  
        c <<= 1;  
        a ^= c;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("c = " + c);  
    }  
}
```

output :

```
a = 3  
b = 1  
c = 6
```

Boolean Logical operators:

- The Boolean logical operators operate only on **boolean** operands.
- All of the binary logical operators combine two **boolean** values to form a resultant **boolean** value.

Operator	Result
&	Logical AND
	Logical OR
^	Logical XOR (exclusive OR)
	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

- The logical Boolean operators, **&**, **|**, and **^**, operate on **boolean** values in the same way that they operate on the bits of an integer.
- The logical **!** operator inverts the Boolean state: **!true == false** and **!false == true**. The following table shows the effect of each logical operation:

A	B	A B	A & B	A ^ B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Short-Circuit Logical Operators(&&, ||):

- Java provides two interesting Boolean operators not found in many other computer languages.
- These are secondary versions of the Boolean AND and OR operators, and are known as *short-circuit* logical operators. As you can see from the preceding table,
- the OR operator results in **true** when **A** is **true**, no matter what **B** is. Similarly, the AND operator results in **false** when **A** is **false**, no matter what **B** is.

- Example:

```
class Test
{
    public static void main(String args[])
    {
        int a=10,b=30;
        if(a<10 && b<40) //false
            System.out.println("Hello");
    }
}
```

```

        if(a>5 || b>40) //true
            System.out.println("Hai");
    }
}

```

- Output:
Hai

The ? Operator:

- Java includes a special *ternary* (three-way) *operator* that can replace certain types of if-then-else statements.
- The ? has this general form:
expression1 ? expression2 : expression3
- In the above form expression1 results a Boolean value either true or false. If it is true expression2 evaluated otherwise expression3 evaluated.

- Example:

```

class Ternary
{
    public static void main(String args[])
    {
        int i, k;
        i = 10;
        k = i < 0 ? -i : i;
        System.out.print("Absolute value of ");
        System.out.println(i + " is " + k);
    }
}

```

- Output :
Absolute value of i is 10

Operator Precedence:

- Bellow table shows the order of precedence for Java operators, from highest to lowest.

Highest			
()	[]	.	!
++	--	~	!
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
&&			
?:			
=	op=		
Lowest			

- This expression first adds 3 to **b** and then shifts **a** right by that result. That is, this expression can be rewritten using redundant parentheses like this:

$$a \gg (b + 3)$$
- However, if you want to first shift **a** right by **b** positions and then add 3 to that result, you will need to parenthesize the expression like this:

$$(a \gg b) + 3$$
- The given expression $a \mid 4 + c \gg b \& 7$ can be evaluated as follows

$$(a \mid (((4 + c) \gg b) \& 7))$$

*****-----*****

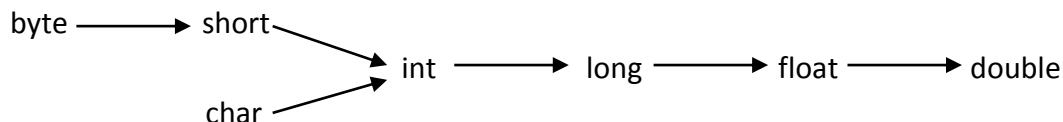
Typecasting:(primitive type conversion):

- Typecasting means converting one data type to another data type.
- Typecasting can be done in two ways
 1. Implicit typecasting
 2. Explicit typecasting

Implicit typecasting:

- Compiler perform implicit typecasting automatically when smaller data type value into larger data type.
- In implicit type casting there is no loss of information.
- Implicit typecasting also called as widening or up casting.
- Example:

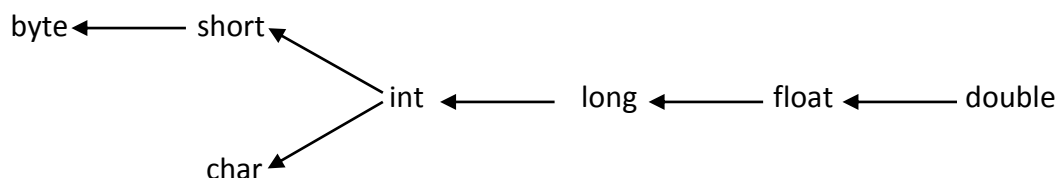
```
int n='a';
System.out.println(n); //97
```



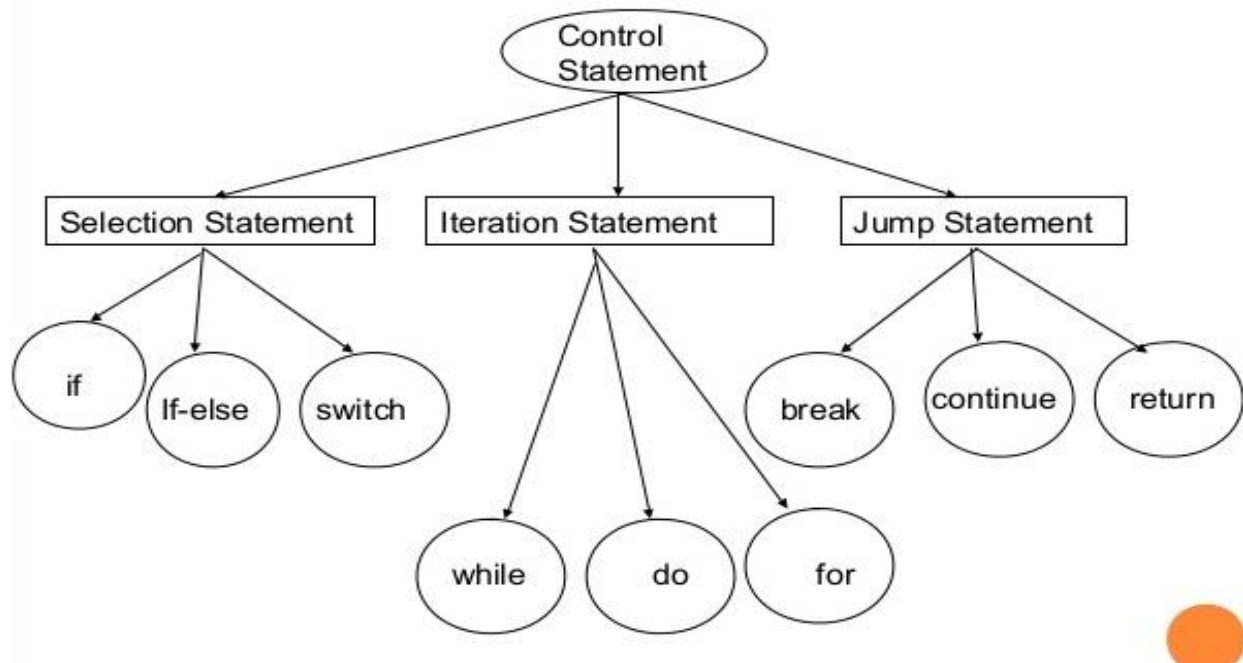
Explicit typecasting:

- Programmer performs explicit typecasting when larger data type value into smaller data type.
- In explicit type casting there may be chance of loss of information.
- Implicit typecasting also called as narrowing or down casting.
- Example:

```
double d=10.5;
int n=(int)d; // n becomes 10
```



Decision making and Looping statements:



if-else:

- General form of if-else is

```
if ( condition )
{
    // true action
}
else
{
    // false action
}
```
- In the above general form condition must be Boolean type either true or false.

Example:

```
int a=10,b=20;
if (a<b)
    System.out.println(" a is smaller than b"); // executes
else
    System.out.println(" a is larger than b");
```

- **Nested if- else:**

```
int a=10,b=20,c=30;
    if (a<b && a<c)
        System.out.println(" a is largest"); // executes
    else if( b<a && b<c )
        System.out.println(" b is largest");
    else
        System.out.println(" c is largest");
```

switch:

- The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.

Syntax:

```
switch(expression)
{
    case value1:    //code to be executed;
                   break; //optional
    case value2:    //code to be executed;
                   break; //optional
    .....
    default:        //code to be executed if all cases are not matched;
}
```

Example:

```
int choice=2;
switch(choice)
{
    case 1: System.out.println("this is case 1");
            break
    case 2: System.out.println("this is case 2");
            break;
    case 3: System.out.println("this is case 3");
            break;

    default: System.out.println("this is default case");
}
```

Output: this is case 2

for loop:

- The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
- There are three types of for loop in java.
 - Simple For Loop
 - For-each or Enhanced For Loop
 - Labeled For Loop

Simple for Loop

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

Syntax:

```
for(initialization;condition;incr/decr)
{
    //code to be executed
}
```

Example:

```
class ForExample
{
    public static void main(String[] args)
    {
        for(int i=1;i<=10;i++)
        {
            System.out.print(i+" ");
        }
    }
}
```

for-each Loop

- The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.
- It works on elements basis not index. It returns element one by one in the defined variable.

Syntax:

```
for(Type var:array)
{
    //code to be executed
}
```

Example:

```
class ForEachExample
{
    public static void main(String[] args)
    {
        int arr[]={12,23,44,56,78};

        for(int i:arr)
        {
```

```

        System.out.println(i+" ");
    }
}

```

Labeled for Loop

- We can have name of each for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.
- Normally, break and continue keywords breaks/continues the inner most for loop only.

Syntax:

```

labelname:
for(initialization;condition;incr/decr)
{
    //code to be executed
}

```

Example:

```

class LabeledForExample
{
    public static void main(String[] args)
    {
        aa:
        for(int i=1;i<=3;i++)
        {
            bb:
            for(int j=1;j<=3;j++)
            {
                if(i==2&&j==2)
                {
                    break aa;
                }
                System.out.println(i+" "+j);
            }
        }
    }
}

```

Output:

```

1 1
1 2
1 3
2 1

```


While Loop:

- The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
while(condition)
{
    //code to be executed
}
```

Example:

```
class WhileExample
{
    public static void main(String[] args)
    {
        int i=1;
        while(i<=10)
        {
            System.out.print(i+" ");
            i++;
        }
    }
}
```

Output:

1 2 3 4 5 6 7 8 9 10

do-while Loop:

- The Java *do-while loop* is used to iterate a part of the program several times.
- If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.
- The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do
{
    //code to be executed
}while(condition);
```

Example:

```
class DoWhileExample
{
    public static void main(String[] args)
    {
        int i=1;
        do
        {
            System.out.print(i);
            i++;
        }while(i<=10);
    }
}
```

Output:

1 2 3 4 5 6 7 8 9 10

break Statement:

- The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

Syntax:

jump-statement;

break;

Example:

```
class BreakExample
{
    public static void main(String[] args)
    {
        for(int i=1;i<=10;i++)
        {
            if(i==5)
            {
                break;
            }
            System.out.print(i);
        }
    }
}
```

Output:

1 2 3 4

continue Statement:

- The Java *continue statement* is used to continue loop.
- It continues the current flow of the program and skips the remaining code at specified condition.
- In case of inner loop, it continues only inner loop.

Syntax:

jump-statement;

continue;

Example:

```
class ContinueExample
{
    public static void main(String[] args)
    {
        for(int i=1;i<=10;i++)
        {
            if(i==5)
            {
                continue;
            }
            System.out.print(i);
        }
    }
}
```

Output:

1 2 3 4 6 7 8 9 10

*****_____*****

UNIT- 2

Classes, Objects and Inheritance

Syllabus:

Classes and Objects- classes, Creating Objects, Methods, constructors, overloading methods and constructors, garbage collection, static keyword, this keyword, parameter passing, recursion, Arrays, Strings, Command line arguments.

Inheritance: Types of Inheritance, Deriving classes using extends keyword, concept of overriding, super keyword, final keyword.

Class:

- A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.
- Simple classes may contain only code or only data, most real-world classes contain both.
- A class is declared by use of the **class** keyword.
- The general form of a class is

```
class classname
{
    type instance-variable1;
    type instance-variable2;
    // .....
    type instance-variableN;
    type methodname1(parameter-list)
    {
        // body of method
    }
    type methodname2(parameter-list)
    {
        // body of method
    }
    // ...
    type methodnameN(parameter-list)
    {
        // body of method
    }
}
```

- The data, or variables, defined within a **class** are called **instance variables**.
- The code is contained within *methods*. Collectively, the methods and variables defined within a class are called **members** of the class.
- Simple Class like as bellow

```
class Box
{
    double width;
    double height;
    double depth;
}
```