JAVA UNIT – 1

What is Java?

- Java is a programming language that is exclusively object oriented developed by Sun Microsystems that plays to the strengths of the Internet.

- Object-oriented programming (OOP)is an unusual but powerful way to develop software.

- In OOP, a computer program is considered to be a group of objects that interact with each other.

What is Java?

- Java has full graphical user interface support.
- Java has full network support.
- Java is platform independent.
- Executes stand alone or "on demand" in web browser as applets.

History Of Java

- Development of Java was started under "Project Green"

- Java was created by a team of programmers (James gosling, Patrick naughton, Chris warth, Ed frank and Mike Sheridan)

- This language was initially called as "Oak" by James gosling but later renamed as "Java" in 1995 as "Oak" was already a registered name.

History Of Java

- Java was designed for the development of software for consumer electronics devices like TVs, VCRs and other electronics devices.

- When the Internet became popular in 1994, sun realized that java was the perfect programming language for the web.

Features Of Java

1) Platform Independent and Portable

The code written in one platform can be easily executed in other platform.

2) Object oriented

Object oriented though out - no coding outside of class definitions, including main()

An extensive class library available in the core language packages.

Features Of Java

3) Compiler / Interpreter Combo

Code is complied to byte codes that are interpreted by a Java Virtual Machine.

This provides portability to any machine for which a virtual machine has been written.

The two steps of compilation and interpretation allow for extensive code checking and improved security.

Features Of Java

4) Built in Networking (Distributed)

java was designed with networking in mind and comes with many classes to develop sophisticated internet communications.

5)Threading (Multi Threaded)

Lightweight processes, called threads, can easily be created to perform multiprocessing.

Features Of Java

6) Robust

Allows to implement robust application as comes up with exception handling, strong type checking, built in data types, garbage collection.

7)Automatic memory management

Automatic garbage collection memory management handled by JVM.

Features Of Java

8) Secure

No Memory Pointers.

Program run inside the virtual machine sandbox

Applets are very secure.

9) Dynamic AndExtensible

java is capable of dynamically linking in new class libraries, methods and objects.

Features Of Java

10) Ease Of Deployment

Java application can be easily deployed to the destination as includes features which allows to create reusable code.

11) Scalability and performance

java assures a significant increase in scalability and performance by improving startup time and reducing the amount of memory used in java runtime environment.

Features Of Java

12) Simple , small and Familiar

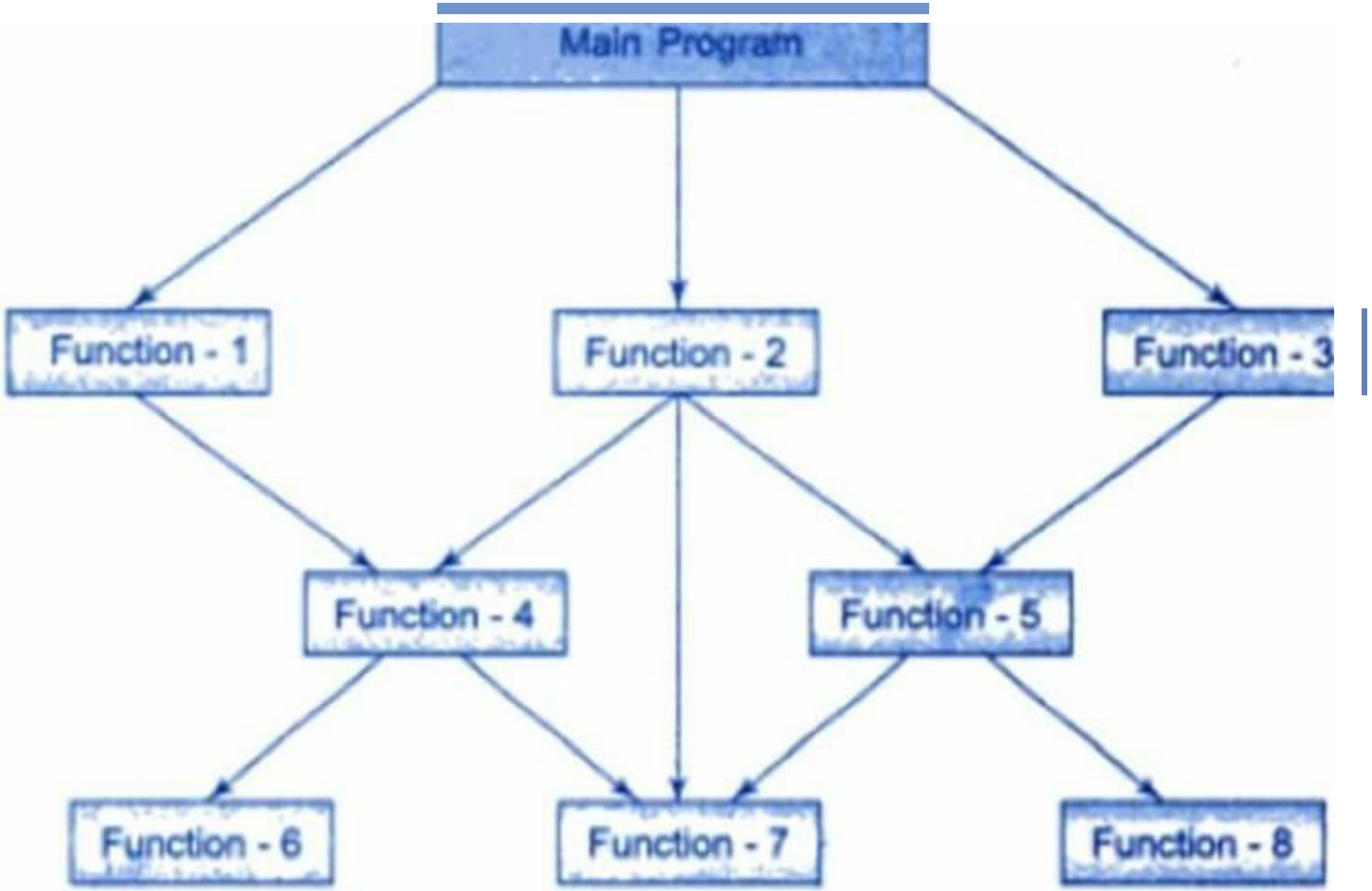
java is small and simple language.

Many features of C and C++ that are either redundant or sources of unreliable code are not the part of Java.

To make java familiar to the programmer , it was modeled on C and C++ languages.

Java Code "looks like a C++" code.

A Look At Procedure Oriented Programming



Characteristics exhibited by procedure programming

- Emphasis is on doing things (algorithm)
- Large Programs are divided into smaller programs known as functions
- Most of the functions share global data
- Data move openly around the system from function to function
- Function transform data from one form to another
- Employs top-down approach in program design.

Object oriented programming

- The major motivating factor in the invention of object oriented approach is to remove some of the flaws encountered in the procedural approach.

- OOP treats data as critical element in the program development and does not allow it to freely around the system.

- It ties data more closely to the functions that operate on it and protects it from accidental modification from outside world.

Striking features of objected oriented programming

- Emphasis is on data rather than procedure.
- Programs are divided into what known as objects
- data is hidden and cant be accessed by external functions
- Objects may communicate with each other though functions
- Follow bottom-up approach in program design.

What is object oriented programming?

An object is like a black box.

The interna l deta ils are hidden.

- Identifying *objects* and assigning *responsibilities* to these objects.
- Objects communicate to other objects by sending *messag es.*
- Messages are received by the *methods* of an object

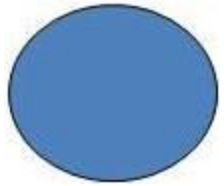
Basics concept Of OOP

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Aggregation
- Behavior and Message

What is an object?

Objects are the basic run time entities in an object oriented system. They may present..

- Tangible Things as a car, printer, ...
- Roles as employee, boss, ...
- Incidents as flight, overflow, ...
- Interactions as contract, sale, ...
- Specifications as colour, shape, ...



The two parts of an object

Object = Data + Methods

or to say the same differentl y:

An object has the responsibility to *know* and the responsibility to *do*.

+

Why do we care about objects?

- Modularity - large software projects can be split up in smaller pieces.
- Reusability - Programs can be assembled from pre-written software components.
- Extensibility - New software components can be written or developed from existing ones.

Classes

- We just mentioned that objects contain data, and code to manipulate that data.
- The entire set of data and code of an object can be made a user defined data type with the help of a class
- In fact objects are variables of the type class
- Once class has been defined ,we can create any number of objects belonging to that class.

Data Encapsulation

- The wrapping up of a data and functions into a single unit (called a class) is known as *encapsulation.*

- The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.

- This insulation of the data from direct access by the program called data hiding or information hiding.

Data Abstraction

- Abstraction refers to the act of representing the essential features without including the background details or explanations.

- Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and functions to operate on these attributes.

Inheritance

- Inheritance means that one class inherits the characteristics of another class.
  This is also called a "is a" relationship:

A car *is a* vehicle

A dog is an animal

A teacher is a person

Polymorphism

- Polymorphism is another important OOP concept.
- Polymorphism , a greek term , means the ability to take more than one form
- An operation may exhibit different behaviors in different instances.
- The behavior depends upon the data used in an operation

Dynamic Binding

- Binding refers to the linking of a procedure call to the code to be executed in response to the call.
- Dynamic binding means the code associated with a given procedure calls is not known until the time of the call at run time.

Aggregation

- Aggregation describes a "has *a"* relationship. One object is a part of another object.

- We distinguish between *composite* aggregation (the composite "owns" the part) and *shared* aggregation (the part is shared by more then one composite).

Behaviour and Messages

- The most important aspect of an object is its *behaviour* (the things it can do). A behaviour is initiated by sending a *message* to the object (usually by calling a method).

The two steps of Object Oriented Programming

- *Making Classes:* Creating, extending or reusing abstract data types.

- *Making Objects interact:* Creating objects from abstract data types and defining their relationships.

The Java Developer's Kit (JDK)

- Before you can start writing java program, you need to acquire and setup some kind of java programming software.

- Although several different products are available for development of java program, the starting place for many new java programmers is the JDK.

JDK Tools

- **javac** - the compiler, which converts source code into Java bytecode

- **Java** - the loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler.

- **appletviewer** - this tool can be used to run and debug Java applets without a web browser.

- **javadoc** - the documentation generator, which automatically generates documentation from source code comment.

JDK Tools

- **java h** - Produces the header file with native methods

- **java p** - Java disassembler, which enables us to convert byte code files into a program description.

- **jdb** - Java debugger, which helps us to find errors in our programs

Java Runtime Environment

- Java runtime environment facilitates the execution of programs developed in java. It comprises of the following.

1) Java Virtual Machine :

It is a program that interprets the intermediate java byte code and generates the desired output.

2) Runtime Class Libraries :

These are set of core class libraries that are required for the execution of the java program.

Java Runtime Environment

3) User Interface Toolkits

AWT and swing are examples of toolkits that support varied input methods for the user to interact with the application platform

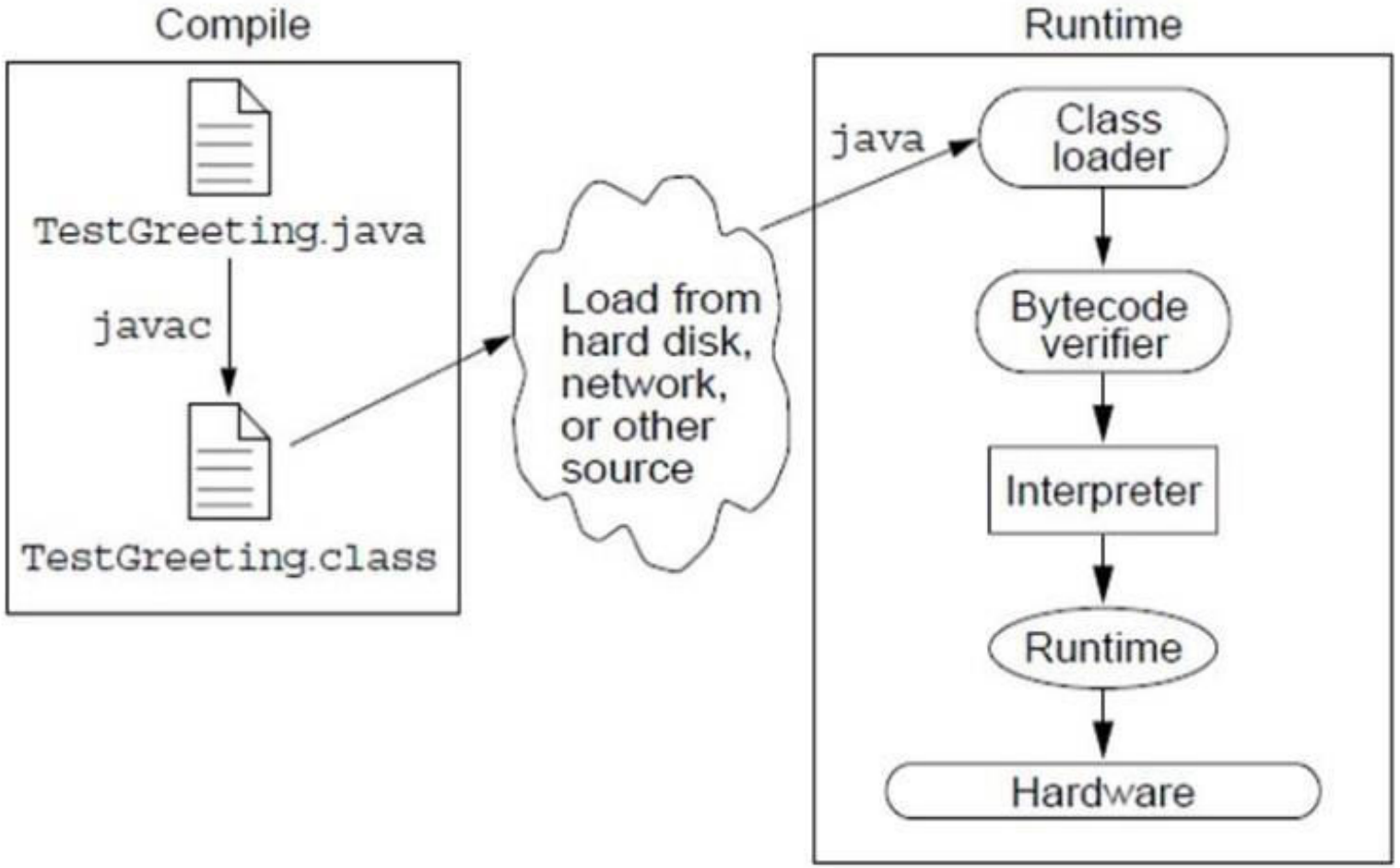
4) Deployment Technologies

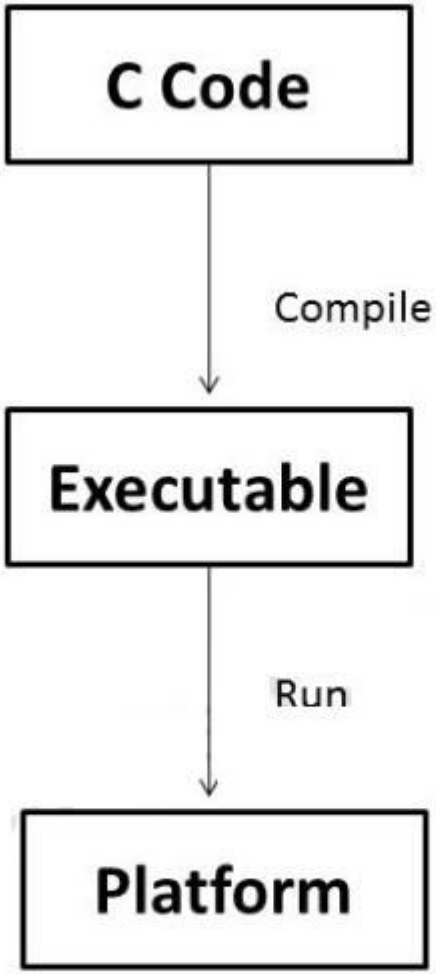
1) Java Plug in : Enables the execution of a java applet on a browser.

2) Java web start : Enables remote deployment of an application.

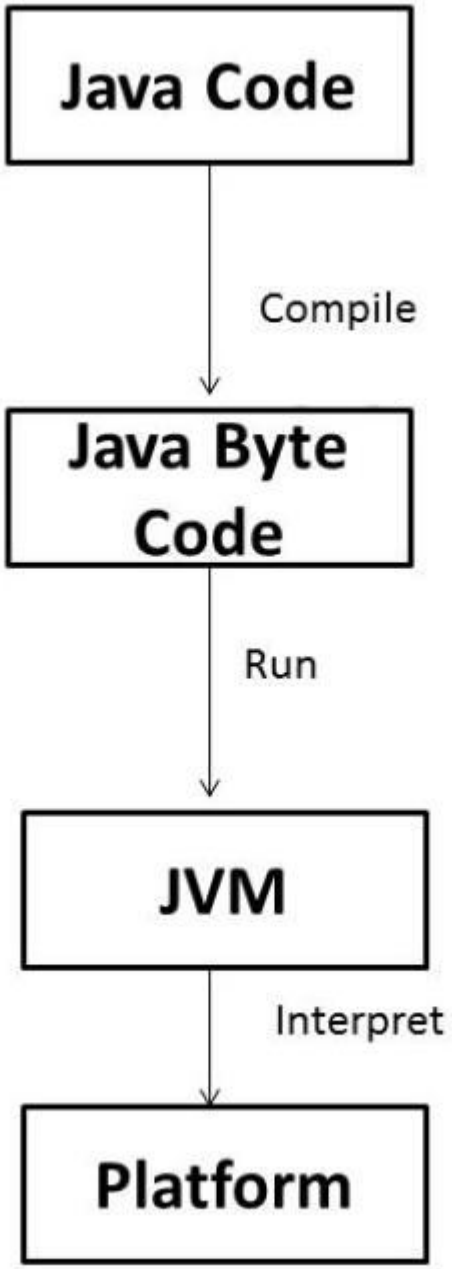
The Java Runtime Environment

The Java application environment performs as follows:





Compilation and execution for C



Compilation and execution for Java

Creating an Application In Java

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

- This class contains **public**, **static** and **void** java keywords

public keyword specifies the accessibility issues of a class. Here, It specifies that method can be accessed from anywhere.

static keyword indicates that this method can be invoked simply by using the name of the class without creating any class-object.

void keyword specifies that this method will not Return any type of data.

main() method is the main entry point of the program, to start execution.

args is a string array that takes values from java command line. We can access values by writing args[0], args[1] etc.

println() function prints the output to the standard output stream (monitor).

out represents the standard output stream (monitor).

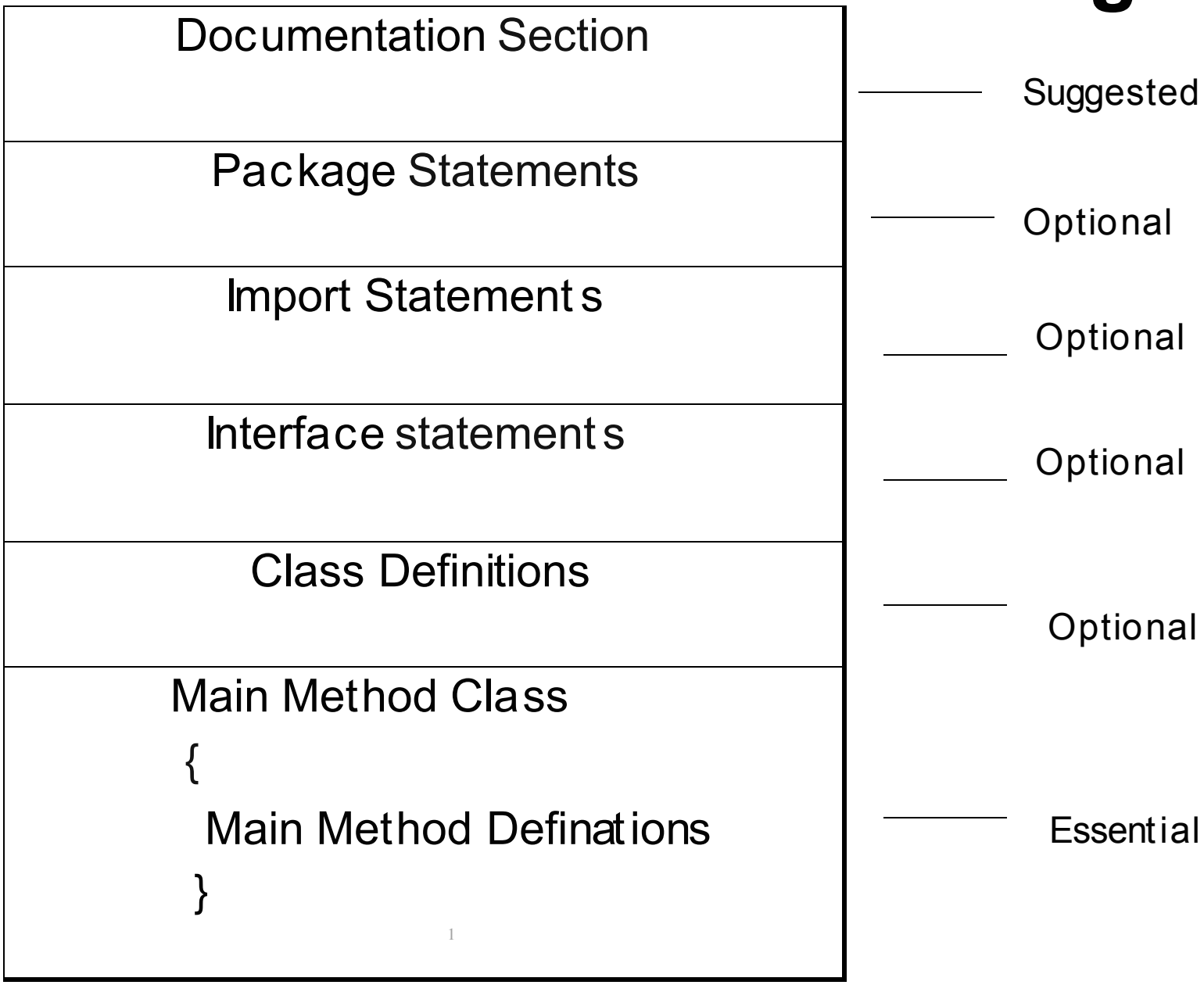
Compiling and Executing

- Open the command prompt and go to the directory where your program is saved.

- Java uses the **"java"** command to compile your program (Type **javac createfirstprogram.java)** .

- After compiling successfully, you can run it. Java uses the "java" command to execute your program (Type **java createfirstprogram)**

General Structure Of Java Program



1) Documentation Section

The Documentation section comprises a set of comment lines giving the name of the program.

2)Package Statement

This statement declares a package name and informs the compiler that classes defined here belongs to this package.

For example,

```
package student;
```

3)Import Statements

This statements instructs the interpreter to load certain class files.

```
import java .io.*;
```

4)Interface statements

An interface is like a class but includes a group of method declarations.

5) Class Definitions

A java program may contain multiple class definitions. Classes are primary and essential elements of a java program.

6) Main Method Class

since every java standalone program requires a main method as its starting point, this class is the essential part of java program.

Comments in Java Code

The Java language supports three kinds of comments:

- /* *text* */The compiler ignores everything from /* to */.

- ***/** documentation **/***

  This indicates a documentation comment *(doc comment,* for short). The JDK javadoc tool uses doc comments when preparing automatically generated documentation.

- *// text*

  The compiler ignores everything from // to the end of the line.

Constants

- Constant refers to fixed values that do not change during the execution of program.

```
123         // integer constant
12.34       //real constants
'A'         //character constant
"C++"       //string constant
037         //octal integer constant
0X2         //hexadecimal integer
```

Backslash Character Constants

- Java Supports some special backslash character constants that are used in output method.

Constant	Meaning
'\b'	back space
'\f'	Form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\''	single quote
'\"'	double quote
'\\'	backslash

Variables

- A variable is a named data storage location in your computer's memory.
- For Example

$$i=1;$$

where the symbol 1 is a constant because it always has the same value (1),

and i is a variable which can contain different value

for example

$$i=10;$$

VARIABLE NAMING CONVENTION

- The name can contain letters, digits, and the underscore character(_).

- The first character of the name must be a letter. The underscore is also a legal first character, but its use is not recommended.

- Case matters (that is uppercase and lowercase letters). Thus, the names count and Count refer to two different variables.

- Java keywords can't be used as variable names. A keyword is a word that is part of the java language.

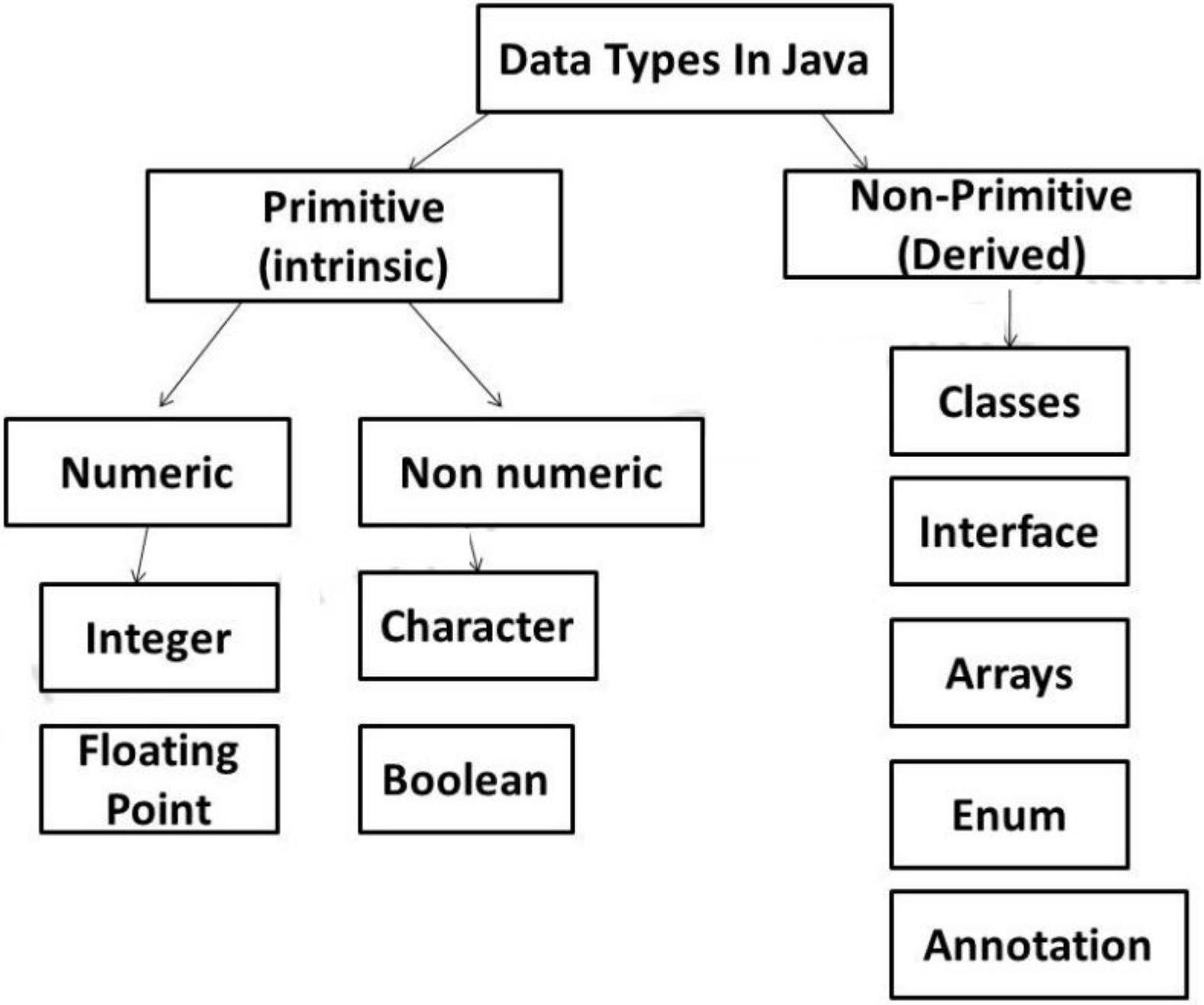
Variable examples

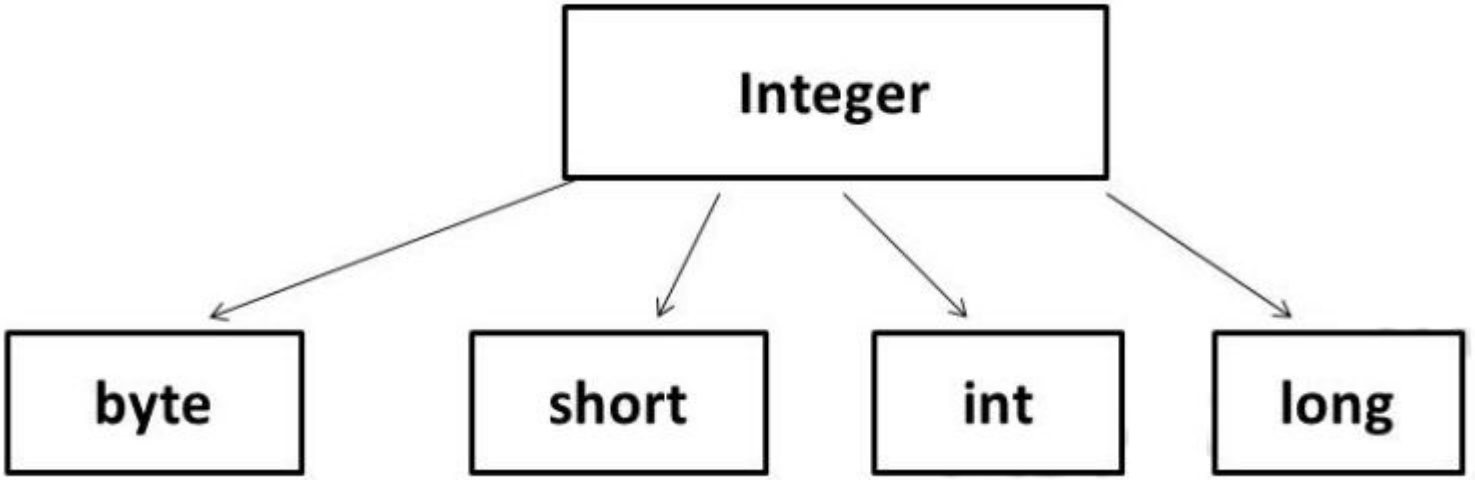
Correct	Incorrect
count	1count
test23	hi!there
high_balance	high . . . balance

Data types in Java

Data type defines a set of permitted values on which the legal operations can be performed.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

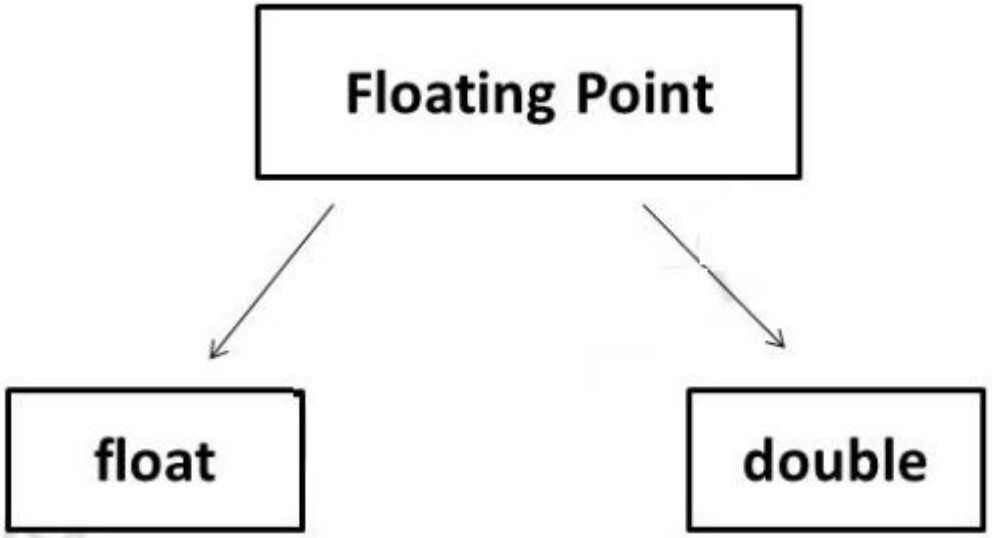

Data Types In Java
Primitive (intrinsic)
Non-Primitive (Derived)
Numeric
Non numeric
Classes
Integer
Character
Interface
Arrays
Floating Point
Boolean
Enum
Annotation



Integer Types can hold whole number such as 123,-96.the size depends upon the type we choose

Type	Size	Minimum Value	Maximum Value
byte	One byte	-128	127
short	Two bytes	-32768	32767
Int	Four bytes	- 2147483648	+2147483647
long	Eight bytes	-9223372036854775807	9223372036854775808

- Floating point can hold whole number containing fractional part also such as 12.12



Type	Size	Minimum Value	Maximum Value
float	4 bytes	1.4012984632481707e-45	3.4028234663852860e+38
double	8 bytes	4.9406564584124654e-324d	1.7976931348623157e+308d

boolean

- boolean data type represents one bit of information.
- There are only two possible values : true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example : boolean one = true

char

- char data type is a single 16-bit Unicode character.
- Char data type is used to store any character.
- Example . char letterA ='A'

Declaration Of variables

- We must declare variable to store value in it
- Declaration does three things

1) It tells the compiler what the variable name is

2) It specifies what type of the data variable will hold

3) The place of declaration decides the scope of variable

The general form

type variable1,variable2....variableN ;

For example,

```
int count;
float x,y;
double pi;
char c1;
```

Storing a values in to the variable

- A variable must be given a value after it has been declared before we use it

- This can be achieved in two ways.

  1) By using an assignment statement

  2) By Using a read statement

Using Assignment Statement

```
variable1=10;
character_variable='u';
```

You can give the values while you are declaring it

```
int value=100;
char yes='y';
```

Using a read statement

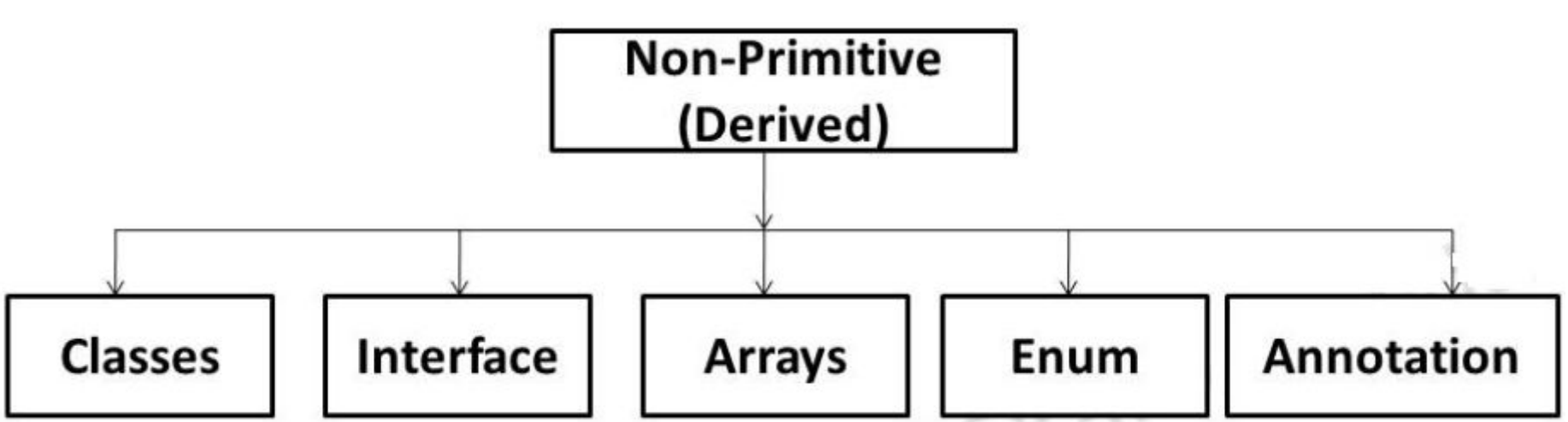
```
import java.io.*;
class Trial
{
   public static void main(String args[]) throws
   Exception
   {
        int i;
        DataInputStream in=new
        DataInputStream(System.in);
        i=Integer.parseInt(in.readLine());
        System.out.print("The Value of i is=" + i);
   }
```

Reference Type Data Type

- The non-primitive data types in Java are objects and arrays.

- These non-primitive types are often called "reference types" because they are handled "by reference"--in other words, the address of the object or array is stored in a variable, passed to methods, and so on.

- By comparison, primitive types are handled "by value"--the actual primitive values are stored in variables and passed to methods.



In Java a reference data type is a variable that can contain the reference or an address of dynamically created object

Reference type data type classes

- As you know that Java is an object-oriented programming language where an object is a variable, associated with methods that is described by a class.

- The name of a class is treated as a type in a java program, so that you can declare a variable of an object-type, and a method which can be called using that object- type variable.

- Whenever a variable is created, a reference to an object is also created using the name of a class for its type i.e. that variable can contain either null or a reference to an object of that class.

- It is not allowed to contain any other kinds of values.

class type example

```
class Fruit
{
   fColor(){....}
   fSize(){....}
}
Fruit apple;
```

Whenever a variable is created, a reference to an object is also created using the name of a class for its type

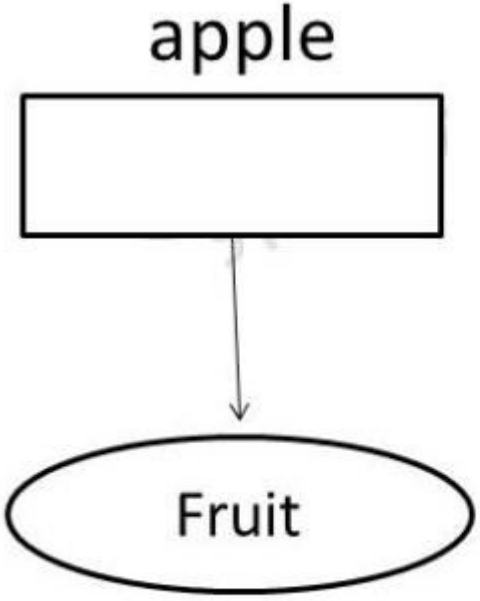
i.e. that variable can contain either **null** or a **reference** to an object of that class.

```
Fruit apple=new
   Fruit();
```

The object becomes an **instance** when the memory is allocated to that object using **new** keyword.

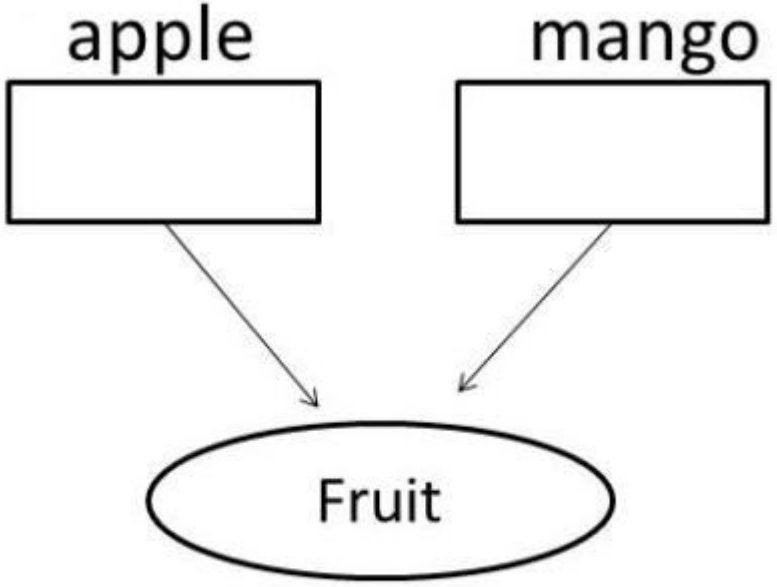
Single Reference to a single instance

```
Fruit apple;
Fruit apple=new Fruit();
```



Multiple Reference to a single instance

```
Fruit apple, mango;
apple=new Fruit();
mango=appIe;
```



Reference type data type
Arrays

- An array is a special kind of object that contains values called elements.

- The java array enables the user to store the values of the same type in contiguous memory allocations.

- The elements in an array are identified by an integer index which initially starts from 0 and ends with one less than number of elements

Creating an Array

- Like any other variables ,arrays must be declared and created in the computer memory before they are used
- Creation of arrays involves three steps

1)Declaring the array

2)Creating Memory Locations

3)Putting values into the memory locations

Declaration Of Arrays

- Arrays in java my be declared in two forms

  Forml

  type arrayname[ ];

  For Example

  int number[];

  float avg[];

  Form2

  type [] arrayname;

  For Example

  int [] number;

  float[] avg;

Creating Memory Locations

- After declaring an array, we need to create it in the memory.
- Java allows us to create arrays using new operator only, as shown below

Arrayname=new type[size];

For Examples,

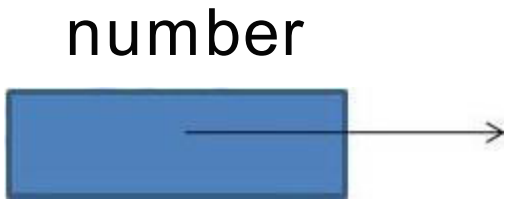
number=new int[5];

Now number refers to an array of 5 integers

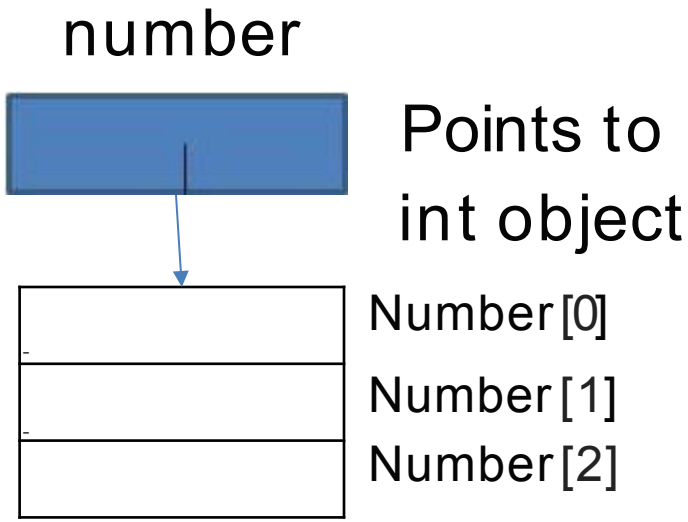
- It is also possible to combine the two steps - declaration and creation into one as shown below

```
int number[]=new int[3];
```

```
int number[];
```




```
number=new  int [3];
```



Initialization of Arrays

- The final step is to put values into the array created.
- This process is known as initialization
- This is done using the array subscripts as shown below

```
arrayname[subscript]=value;
```

for example,

```
number[0]=35;
number[1]=40;
```

Initialization of Arrays

- Unlike C, Java protects array from overruns and under runs.
- We can also initialize arrays automatically in the same way as ordinary variables when they are declared as shown below:

type arrayname[]={list of values};

For example,

int number[]={34,33,34,23};

The compiler allocates enough space for all elements specified in list

Array Of Reference type

- Arrays can be created for any data types, including the reference data types.
- For Examples,

```
String [] array=new String[5];
```

This would allocate an array with 5 elements

We could initialize the elements in an array using the assignment given below

```
array[0]="Hello";
array[ ]="World";
```

Two Dimensional Array

- There may be a situation where you may need to store values in to tabular format
- Consider the following data table, which shows the value of sales of three items by sales man

	Item 1	Item2	Item3
Sales Man 1	12	123	11
Sales Man 2	15	45	45

- We can think of this table as a matrix consisting of two rows and three columns.

Declaration Of Two Dimensional Array

- We can declare two dimensional array in the following ways

```
int    two_d_array[][];
int  []  two_d_array[];
int [][] two_d_array;
```

Allocating a memory for two dimensional array

- There are two ways of allocating two dimensional array using new operator

```
two_d_array=new int[10][5];
two_d_array=new int[10][];
```

When allocating a memory for two dimensional array the size of first dimension is mandatory and size of second dimension is optional

Variable size array

```
int x[ ]=new int[3][ ]; x
[10]=new int[2];

x[1 ]=new int[3];

x[2]=new int[4];
```

Unicode Escapes In Java Source Code

- Most characters in a Java source program are ASCII characters, although all Unicode characters can be used in comments, character and string literals.

- Unicode characters can also be expressed through Unicode Escape Sequences.

- Unicode escape sequence may appear anywhere in a Java source file.

Unicode escape sequences

- Unicode escape sequences consist of

  1) a backslash '\' (ASCII character 92, hex 0x5c),

  2) a 'u' (ASCII 117, hex 0x75)

  3) optionally one or more additional 'u' characters, and

  4) four hexadecimal digits (the characters '0' through '9' or 'a' through 'f' or 'A' through 'F').

- Such sequences represent the UTF-16 encoding of a Unicode character, for example,

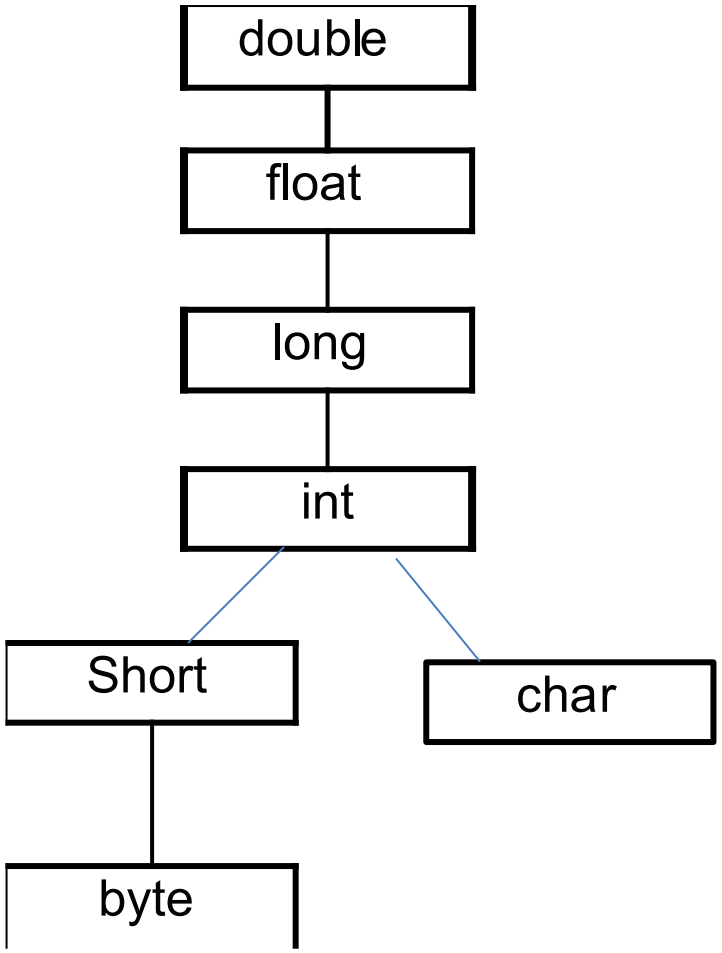
- For example \u2122 is the trademark symbol(TM)

Super Types and Sub-types

- In java we have a relationship of super type and sub type between data types
- A data type is a sub-type of a given data type if it is a special kind of the given data type.
- The sub-type would then always be usable wherever the super type is usable
- In case of primitive data types , int is a sub type of long, as int can always be used wherever long is used

Super Types and Sub-types

- All ints are long data type ,but not all long data type may be ints
- We cant use the super-type for sub type.



Super Types and Sub-types relationships in primitive data type.

OPERATORS

- Operator is symbol, which represents, a particular operation that can be performed on some data.

- The data itself (which can be either a variable or a constant) is called the "operand". The operator thus operates on operand.

- They are Classified as

  (1) Arithmetic (2) relational (3) logical

  (4) Assignment (5) increment/decrement

  (6) conditional (7)Bitwise operators

  (8)instanceof operator

  (9)0ther Operators

Arithmetic operators

OPERATOR	PURPOSE
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder after integer division

Example (Suppose A=10 and B=3)

Expression	Values
A+b	13
a-b	7
A*b	30
A/b	3
A%b	1

String Concatenation

- You can concatenate string using a + operator in java

  For example

  "Hello " + "world"

  Will results in "Helloworld"

Relational Operators

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

Relational Operators

- Suppose that i, j and k integer variable whose va lues are 1,2 and 3 respectively

Expression	Interpretation	value
i<j	True	1
(i+j)>=k	True	1
(j+k)>(I+5)	False	0
K!=3	False	0
J = = 2	True	1

Logical Operators

Operator	Meaning
&&	AND
||	OR
!	Not

The following table shows the result of the combined expressions depending on the value of the expression :

Operands		Results	
Exp1	Exp2	Exp1 && exp2	Exp1 || exp2
0	0	0	0
0	Non-zero	0	1
Non-zero	0	0	1
Non-zero	Non-zero	1	1

Assignment Operators

- Assignment operators are use to form assignment expressions, which assign the value of an expression to an identifier.
- The most commonly used assignment operator is =

- For Example

a= a+l	is equivalent to	a+=l
a=a-1		a-=1
a=a*(c + d)		a*=(c + d)
a=a/(n-1)		a/=(n-1)
a=a%10		a%=10

Increment and Decrement Operators :

- "Java" offers two special operator ++ and -- called increment and decrement operators, respectively.
- There are "unary" operators since they operator on only one operand.
- The operand has to be a variable and not a constant.
- The use of these operates resulting incrementing or decrementing value of the variable is 1.
- For example

```
int a=5;
a++;        //a=a+1; post increment
++a;        //pre increment
```

Conditional operators

- Simple conditional operations can be carried out with the conditional operator (?:)

- An expression that makes the use of the conditional operator is called a conditional expression.

- Conditional operators are also known as ternary operators.

- Syntax:

```
(condition) ? true : false
T=(i<0) ? 0 : 100
```

Bitwise Operators

Java provides Bit wise operators to manipulate the contents of variables at the bit level.

These variables must be of numeric data type (short, int, or long).

Symbol	Name of the Operator	Example
~	Unary bitwise complement	~op2
&	Bitwise AND	opI & op2
|	Bitwise inclusive OR	opI | op2
^	Bitwise exclusive OR	opI ^ op2
<<	Signed left shift	opI << op2
>>	Signed right sift	opI >> op2
>>>	Unsigned right shift	opI >>> op2

Instanceof Operator

- Java provides a run-time operator instanceof to compare a class and an instance of that class.

- This operator " instanceof" compares an object to a specified class type.

- The instanceof operator is defined to know about an object's relationship with a class.

Instanceof Operator

- It evaluates to **true,** if the object or array is an **instance** of the specified type; otherwise it returns false.
- The **instanceof** operator can be used with the arrays and objects. It can't be used with primitive data types and values.
- Its signature is written as:

object **instanceof** type

For example,

```
String name = "James";
boolean result = name instanceof String;
// This will return true since name is type of String
```

Other Operators

1) new

operator is always used to create a new instance of some class or for allocating an array of any data type.

2) [] operator

is used to access the elements in array.

3) . operator

is used to access any member of an instance or invoking methods on an instance.

Conversion Of Numeric types

- In java there are various types of conversions involving numeric types

  1)Widening conversion

  2)Narrowing Conversion

  3)Mixed Conversion

widening conversion

- A widening conversion occurs when a value of one type is converted to a wider type.
- Java performs widening conversions automatically.
- For example

  1)byte to short,int,long,float or double
  2)short to int,long,float or double
  3)char to int,long,float or double
  4)int to long,float or double
  S)long to float or double
  6)float to double

- A widening conversion is the conversion of a sub-type to one of its super-types

widening conversion

- In case of widening conversion from int to float and long to float or double , there can be a loss of precision

```
class abc
{
        public static void main(String args[])
        {
                int i=123456789;
                float f=i;
                int j=(int)f;
                System.out.println("i= "+i+",j="+j);
        }
}
```

Output : i=123456789 j=123456792

Narrowing conversion

- A narrowing conversion occurs when value is converted to a type that is not wider than it.
- A narrowing conversion is the conversion of a super-type to one of its sub-type.
- For Example,

  double to byte , short, char , int,long or float

  float to byte,short,char,int or long

  long to byte,short,char or int

  int to byte,short or char

  short to byte

Narrowing Conversion

- Narrowing conversions are not safe. There is scope for loosing data in narrow conversion
- For example,

  1)Float to int causes truncation of the fractional part float

```
float f=12.12f;
int j=(int)f;  //j=12
```

  2) double to float causes rounding of digits

```
double d=111112.5911111111;
float j=(float)d;  //j=111112.59
```

Mixed Conversion

- A mixed conversion is the conversion that involves first a widening conversion followed by a narrowing conversion
- For numeric types the cases for mixed conversions.

1) Char to byte or short
2) short to char
3) byte to char

Mixed conversion

- The mixed conversion involves the conversion of char with the other two sub types of int.

Char to byte or short

- In this conversions, first a widening conversion is done to the int type and then the narrowing conversion is done to the target type

Statements

- A java program is a set of statements, which are normally executed sequentially in the order in which they appear.

- However ,we have a number of situations where we may have to change the order of execution of statements based on certain condition

- Or repeat a group of statements until certain Specified conditions are met.

Statements

- In java we have the two common kinds of flow statements

  1) Conditional statements

     This kind of statements are responsible for how to direct the sequence of execution.

     (includes if, if-else and switch case)

  2)Loop statements

     This kind of statements are responsible for to execute sequence of statements

     repeatedly.

     (includes for loop, while loop and do-while)

The if conditional statement

The if-else statement:

- The if-else statement is used to carry out a logical test and then take one of two possible actions depending on the outcome of the test

- Thus, in its simplest general form, the statement can be written.

```
if(expression)
{
        statement;
}
```

The if-else conditional statement

- The general form of an if statement which include the else clause is

```
if(expression)
{
        statement 1;
}
else
{
        statement 2;
}
```

If the expression is true then statement 1 will be executed. Otherwise, statement 2 will be executed.

Nested If Else

```
if(expression)
{
    statement1;
}
else
{
    if (expression)
    {
        statement2;
    }
}
```

Else if ladder

```
if( expression 1 )
        statement 1;
else if( expression 2 )
        statement 2;
else
        statement 3;
```

Switch case Conditional Statement

- The switch statement:

  causes a particular group of statements to be chosen from several available groups.

- The selection is based upon the current value of an expression that is included within a switch statement.

The general form of switch-case

```
switch(expression)
{
        case vaule-1:
                statements;
                break;
        case value-2:
                statements;
                break;
        default:
                statements;
                break;
}
```

When switch statement is executed the expression is evaluated and control is transferred directly to the group of statements whose case labels value matches the value of the expression.

```
switch(choice)
{
        case 'r' :
                System.out.println("Red");
                break;
        case 'b' :
                System.out.println("blue");
                break;
        default :
                System.out.println("Error");
                break;
    }
```

LOOP STATEMENT

- If we want to perform certain action for no of times or we want to execute same statement or a group of statement repeatedly then we can use different type of loop statement available in Java.

- Basically there are 3 types of loop structure available in java

  (1) While loop

  (2) Do..while

  (3) For loop

While Loop

- The while statement is used to carry out looping operations.
- The general form of the statements initialization;

```
while(exp)
{
        statement 1;
        statement 2;
        increment/ decrement;
}
```

While loop example

```
int digit = 0;
while(digit<=9)
{
        System.out.println("digit=" + digit);
        ++digit ;
}
```

The while loop is entry controlled loop statement. The test condition is evaluated and if the condition is true then the body of loop is executed

Do-While Loop statement

- Sometimes, however, it is desirable to have a loop with the test for continuation at the end or each pass.
- This can be accomplished by means of the do-while statement.
- The general form of do-while statement is

```
do
{
        statement1;
        statement 2;
        increment/decrement operator;
} while(expression);
```

Do-While Loop Example

```
int digit = 0;
do
{
        System.out.println("digit=" + digit);
}while(digit<=9);
```

Do.. While loop is an exit control loop where body is executed first and than test condition is evaluated

For Loop Statement

- The for statement is another entry controller that provides a more concise loop control structure.

- The general form of the for loop is :

```
for(initialization; test condition; inc/decrement)
{
    statement 1;
    statement 2;
}
```

For loop example

```
for(int digit = 0; digit<9; digit++)
{
    System.out.println("digit=" + digit);
}
```

Reverse For loop

- The for statement allows for negative increments. (or decrements)
- For example, the loop discussed above can be written as follows:

```
for( int digit=9; digit>=0; digit--)
{
    System.out.println("digit=" + digit);
}
```

The break statement

- The break statement is used to terminate loops or to exit a switch.
- The break keyword must be used inside any loop or a switch statement.
- The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

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

The CONTINUE STATEMENT

- The continue statement is used to skip or to bypass some step or iteration of looping structure.

```
for(i=1 ; i<=1 0 ; i++}
{
    if(i<5}
    continue;
    System.out.println( i);
}
```

The output of the above program will be 6,7,8,9,10.

Return Statement

- The return statement is used inside a method or function to return a value and not to proceed further inside the method

For Example,

```
int display()
{
        return 2;
}
```

Labeled Loops

- In Java We can give a label to a block of statements.
- A label is any valid java variable name.
- To give a label to a loop, place it before the loop with a colon at the end.
- For Example,

```
outer: for(int m=1;m<11;m++)
        {
       inner: for(int n=1;n<11;n++)
              {
              System.out.print( m* n);
                    if(n==m)
                    {
                          continue outer;
                    }
              }
       }
```

```
outer: for(int m=1;m<11;m++)
        {
       inner: for(int n=1;n<11;n++)
              {
              System.out.print( m* n);
                    if(n==m)
                    {
                          break outer;
                    }
              }
      }
```

Defining a classes in java

```
class rectangle
{
   //member  of the class
}
Class TestRectangle
{
        public static void main(String [] args)
        {
                rectangle r1,r2;
        }
}
```

Various members within a class

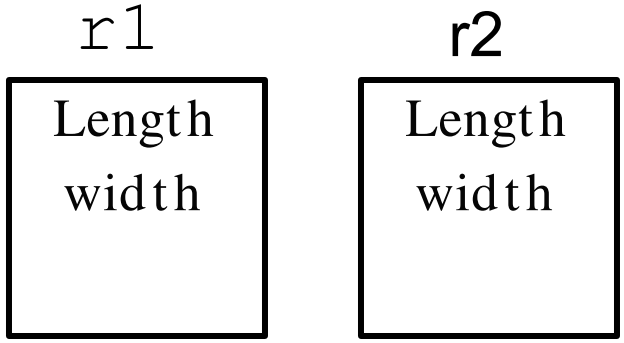
1) Instance variables

The instances of a data type may maintain some information (state)

```
class rectangle
{
        int length; //instance variable
        int width;
}
```

- The length and width in the above class definition are known as instance variables

```
Class TestRectangle
{
        public static void main(String [] args)
        {
                rectangle r1,r2;
                r1=new rectangle();
                r2=new rectangle();
        }
}
```



Various members within a class

2)Methods

To interact with instances of class we have to define methods , which is known as member functions of the class

The method can have various forms

- method with no arguments and no return values.
- method with arguments and no return values.
- method with arguments and return values.
- method with no arguments and return values.

Defining a method

```
class rectangle
{
        int length; //instance variable
        int width;
       //  method which can be invoked on any
        instance of this class
        int area()
        {
                        return length*width;
        }
```

Invoking a method

```
class TestRectangle
{
        public static void main(String [] args)
        {
                int result;
                rectangle r1,r2;
                r1=new rectangle();
                r2=new rectangle();
                result=r1.area();
        }
}
```

Method Overloading

- When method having the same name but different signature, it is known as Method Overloading.

```
class rectangle
{       void display()
        {
                System.out.println("Display 1");
        }
        void display(int x)
        {
                System.out.println("Display 2");
        }
}
```

Invoking an overloaded method

```
class TestRectangle
{
        public static void main(String [] args)
        {
                int temp=0;
                rectangle r  =new rectangle();
                r  .display();
                r  .display(temp );
        }
}
```

Various members within a class

3) Constructors

- **What is the use of Constructor**

  The main use of constructors is to initialize objects. The function of initialization is automatically carried out by the use of a special member function called a constructor.

  General Syntax of Constructor

- Constructor is a special member function that takes the same name as the class name. The syntax generally is as given below:

```
Class-name()
{}
```

Some characteristics of constructors

- Constructors are just functions with the same name as the class and it should not have a return type
- Constructors are intended to initialize the members of the class when an instance of that class is created.
- Constructors are not called directly (except through initializer lists)
- Multiple constructors for the same class can be defined. They must have different parameters to distinguish them.

```
class rectangle
{
        int length,width;
        rectangle()     //constructor of class rectangle
        {
                length=0;
                width=0;
        }
}
Class TestRectangle
{
    public static void main(String [] args)
    {
      rectangle r1=new rectangle(); // constructor will be invoked
    }
}
```

Overloading Constructor

- Just like we overloaded methods, we can overload constructors also

```
class rectangle
{       int length,width;
        rectangle()
        {               length=0;   width=0;        }
        rectangle(int l,int w)
        {
                length=l;   width=w;
        }
}
```

```
class TestRectangle
{
    public static void main(String [] args)
    {
        rectangle r1=new  rectangle();
        /*constructor with no parameters will be invoked*/

        rectangle r2=new rectangle(3,4);
        /*constructor with two int type parameters will be
        invoked*/
    }
}
```

Garbage Collector and Finalize method

- In java, only the garbage collector can deallocate any instance.

- An instance would be eligible for garbage collection only when there are no references referring to it from the application

- The garbage collector works parallel to the application, and its only job is to look for instances that can not be used from the application

Garbage Collector and Finalize method

- Now , when the garbage collector decides to deallocate any instance, it looks for a special method in that class definition for the instance called finalize()

- And would invoke it just before deallocation.

- Therefore, in a class definition we may define a method called finalize(),which is not meant to be invoked from the application, but is meant for use by the garbage collector ,before the garbage collector deallocates any instances of that class

The general form of finalize()

- The method signature for the finalize is as follows

```
protected void finalize()
```

Therefore in the life of any instance the constructor is like the first method that is invoked , and finalize() is the last method invoked in the life of any instance of a class

```
class rectangle
{       rectangle()
        {
                System.out.println("Constructor invoked");
        }
        protected void finalize()
        {
                System.out.println("Finalize invoked");
        }
}
class Rectangle_access
{       public static void main(String args[])
            {
                rectangle r=new rectangle();
            }
}
```

Static variables and Methods

- We have seen that a class basically contains two sections

- One declares variable and other declares methods.

- The variables and methods are called instance variables and instance methods.

- This is because every time the class is instantiated , a new copy of each of them is created. And they are accessed using the objects(with dot operator)

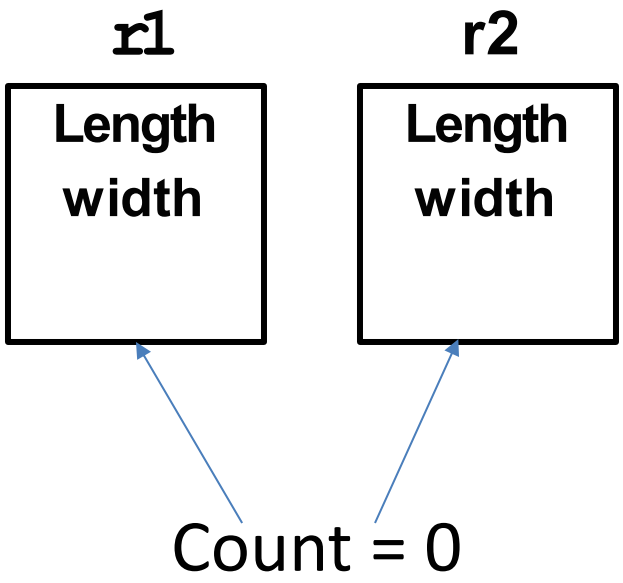
- let us assume that we want to define a member that is common to all the objects and accessed without using a particular object.

- That is member belongs to the class as a whole rather than the objects created from the class.
- Such members can be defined using static keyword.

```
static int count;
static int increment( )
```

- Static variables are used when we want to have a variable common to all instances of class
- One of the most common examples is to have a variable that could keep a count of how many objects of a class have been created

```
class rectangle
{
        static int count;  //static variable
        int length;        //instance variable
        int width;
    }
```



Static methods

- You can declare methods also as a static.
  - The most common example of a static member is main().
  - main() is declared as static because it must be called before any object exist.
  - Method declared as static have several restrictions:
    1) They can only call other static methods.
    2) They must only access static data.
    3) They cannot refer to **this or super in any way.**

```
class StaticDemo
  {
      static int a=42;  static int b=99;
      static void callme()
      {       System.out.println("a=" +a);
      }
  }
   class StaticByName
  {     public static void main(String args[])
      {
            StaticDemo.callme();
            System.out.println("b=" + StaticDemo.b);
      }
   }
```

Output: A=42 8=99

Initializer Block

- We can have a member block with no name and such a block is known as the initializer block.

- An initializer block in a class is invoked whenever any of the constructor of the class is invoked.

- This block is always executed before the constructor code is executed and after instance declaration with assignment

```
class rectangle
{
    int length=2;
    rectangle()
    {
        length=3;
        System.out.println("length :" +length);
    }
    //initializer block
    {
        System.out.println("length :" +length);
        length=4;
        System.out.println("length :" +length);
    }
}
```

rectangle r = new rectangle();

output

length :2
length :4
length :3

Class Initializer block

- A class initializer block is created just like the initializer block ,but it is declared to be static

- The block is known as static block also

- This block is executed only once when the class is loaded.

```
class UseStatic
{
    static int a=3;
    static int b;
    static void calc()
    {
        System.out.println("a=" +a);
        System.out.println("b=" +b);
    }
    static
    {
        System.out.println("Static block ");
        b=a*4;
    }
}
```

```
public static void main(String args[])
{
        UseStatic.cale();
}
```

Output:

Static block

A=3

8=12

Using command line argument

- The command line argument are available to the application (method main) though the String[] parameter

```
class Hello
{
        public static void main(String args[])
        {
                System.out.println(args.length);
                System.out.println(args[0]);
        }
}
```

Command in dos

```
c:\javac Hello.java
c:\java Hello xyz zbc
```