Program vs Software:

A single program can be called as a software when it has proper documentation.

Program	Software
Program is a set of instructions. which is small in size.	Software is a collection of many different programs working together
Lacks proper documentation	Well documented & user-manual
Lacks proper user interface and may be of sole user	Well-designed user interface which facilitates more number of users
Program is about ad hoc (improvised) development	Software involves systematic development.

What is proper documentation?

Documentation includes comments, naming conventions, indentation (proper bracing) of the code along with header files and other stuff.
Documentation is the most important thing that should be kept in mind before we jump into coding. A code is not a good one if it is not readable. Documentation makes the code readable.

Software Failures:

The program we develop may not be satisfying to the client even though it is at its best; {Then we need to mold the program in a client-oriented manner}, fail to meet user requirements, frequent crashes, Expensive, difficult to alter, debug, and enhance.

Debugging is the process of locating errors in the program. It is also a part of programming.

Evolution of Design Techniques :

Ad hoc -> Control flow based -> Data structure based -> Data flow based -> Object- oriented

Formal Representation : A method of interpreting something which everyone will interpret in the same way.

Software Maintenance :

There are 4 types of maintenance:

1. Adaptive maintenance:

Making changes in the software according to the desired platform like if the software is made for windows and if the client is a MAC user certain changes are to be made for the usage of client

Ex: before CMS was maintained by campus now it's done by cloud

2. corrective maintenance:

Maintenance done to correct a bug after the product is sold to the client.

3.perfective maintenance:

Maintenance done for some enhancements in user interface or to the performance to the activation

4. preventive maintenance:

Making changes in app/software in anticipation to avoid certain threats/attacks or bugs which you feel may come up in future.

OOP: OBJECT ORIENTED PROGRAMMING

OOP is a programming approach that solves real world problems (Client preferences).

Popular Object-Oriented Languages(OOL) :
Java, Python, C++, Ruby, C#

C is procedure-oriented programming.

C++ is a partial object-oriented programming.

Java is a complete object-oriented programming.

C is a procedure-oriented programming approach, which is popular. Why do we need an object-oriented approach? Are they competing with each other (or) complementing each other?

The object oriented paradigm is not to replace process oriented approach. There are certain class of problems which are solved by a procedure-oriented approach that would not be as good enough (efficient) as it would be when solved by an object -oriented approach. Object-oriented approach still involves procedure-oriented approach so they are not competing with each other indeed they are complementing each other.

FEATURES OF OOP:

1. Object:

An entity (real-world/ non-real world) is an object which can be defined with some features.

2. Class:

Class is a generalized representation of individual objects. object is the projection/specific representation (concrete instantiation) of a class.

object is an instance/specification of a class.

Which comes first?

We identify objects and group them as a class during the analysis. But while implementing (coding) we define a class first; and then we instantiate the class (instantiation of a class is an object)

A constructor is a function that lets you instantiate a class.

Analysis -> Designing -> Implementation

3.Abstraction is a representation of essential characteristics about an object and hiding the other details.

4.Encapsulation is binding the data with the function that will be operated on the data. Class is an encapsulated representation. It controls the access to fields and methods of a class from outside.

*Reusability is the ability to reuse a class in some other class. The amount of reuse depends on the accessibility (Access specifiers).

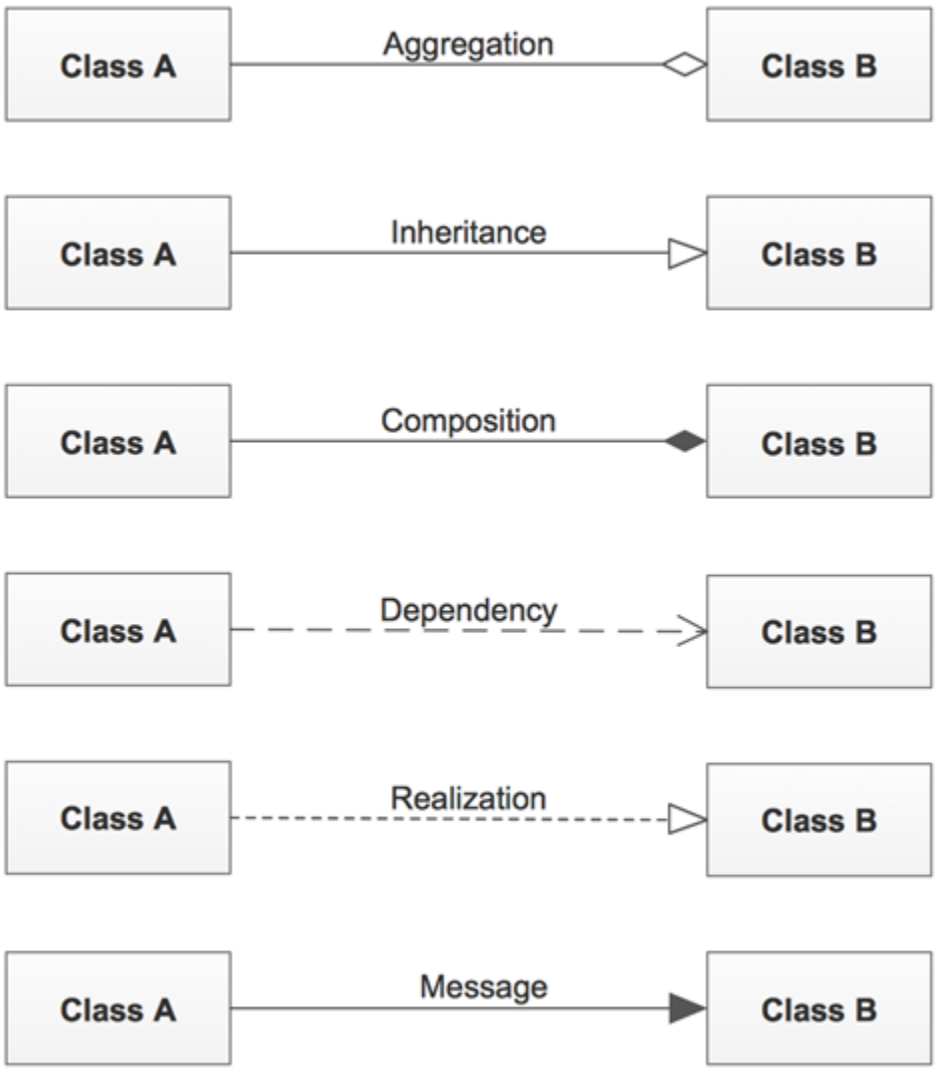
5.Inheritance is a mechanism where you can derive a class from another class that shares a set of attributes and methods.

6. Polymorphism is implementing the same thing in different ways.

*Message passing is the way in which one object passes the info to the other object which happens using a function call.

We may need to connect the classes in the program so the connections are known as CLASS RELATIONSHIPS; the 6 UML notations are as follows.

CLASS RELATIONSHIPS:



***In inheritance;**

Specialization is top -down (Account can be savings and current) Generalization is bottom-up (Savings and current are types of accounts).

1. Association:

Association is a relationship between two objects i.e.; the multiplicity between objects. Ex: Student & Faculty

2. Inheritance:

Inheritance is a relationship between two/more classes where derived class (sub class /Child class) inherits the properties of base class (Parent class /super class). Inheritance is also called a **"is-a"** relationship.

Ex: Savings Account inherits Account

3. Aggregation :

Aggregation is a special form of association; it is the directional association between objects. Direction between them specifies which object contains the other object. Aggregation is also called a **"Has-a"** relationship.

Ex: "House has a room". Here the room is known as part (can exist independently) and house is known as whole.

4. Composition:

Composition is a special form of aggregation; restricted aggregation is called composition. Here part cannot exist independently without the existence of whole. When an object contains the other object (House contains a room) if the contained object Room (part) cannot exist independently without the existence of a container object House(whole), then it is called composition.

Ex: A class contains students. A student cannot exist without a class. There exists composition between class and students.

5. Dependency :

Dependency exists between two classes when Change in structure of a class affects the other related class. It need not be the same vice-versa. It generally happens When one class contains the other class.

Ex: Relationship between shape and circle is dependency

6. Realization/Implementation:

Realization means Interfaces. Interfaces are the same as classes but not as detailed as classes. Data members in an Interface are constant and are just declared but not defined (All the member functions will be abstract). A and B are interfaces and if B uses A then we say **B implements A : (B A)**

*when a class itself is used as a data type then it is known as Abstract DataType(ADT).

Why is java's main method static?

To avoid object Creation(Instantiation of class); if it was a non-static method JVM creates an object first then call a main () method that will lead the problem of extra memory allocation

Introduction to JAVA :

Java is an object oriented Programming language. We cannot write any program without using a class and object. We can easily get started (simple) with the support of in-built libraries. Unlike C++ where pointers are one concern of security; java is more secure which is used to develop unpredictable random numbers (OTP's). Java is robust i.e.; durable and works Perfectly. Java is dependent on JVM only and is platform independent & portable regarding the OS used. Unlike C++, Objects cannot be created in a stack for java. Object is always created in Heap memory, hence it is Architecture hi Neutral & Dynamic. Unlike C++ (compiled language) java is an interpreted language which involves compilation; program lines are read one by one and are converted to the byte code which takes more time and that's the reason java introduced JIT compiler(it takes a bunch of statements and compiles it into a byte code).

Java displays high performance. Java is distributed in client server applications many people from different locations can access and replicate the programs. Java is Multi-threaded.

★ **File name should be same as the class containing main function**

psvm : public static void main

public is an access specifier, **static** means this method works without instantiating the object, **void** is the method return type

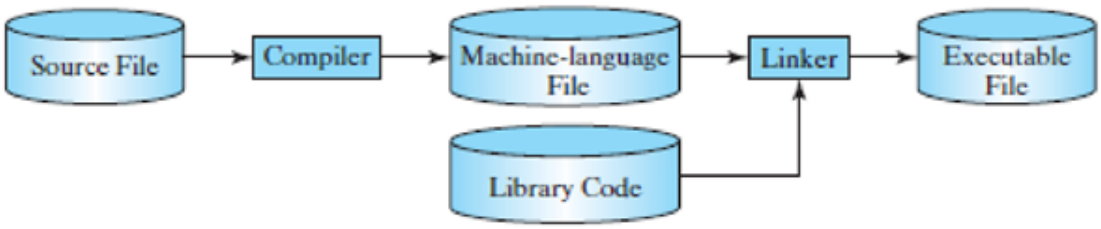
Unlike C++ Java is an **Auto garbage collector**. All the memory locations created will be moved to garbage if not in use but other external factors can

make use of them. Default value is no more a garbage value every variable has a certain value.

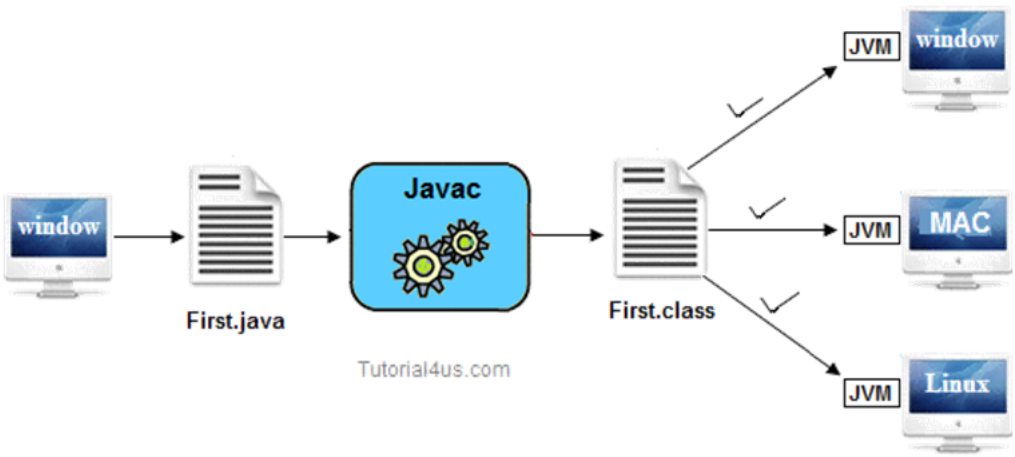
Command-Line Arguments

When you type java file_name on the command prompt, **JVM** (Java virtual machine) accesses the main function as class_name.main();Data is provided in the String args [] of the main function. At the run-time data is provided; if not by default the value will be stored as 0. In a command prompt values are entered in the line where we write the command to run. In eclipse IDE; Right click on java file – run as – configuration – arguments – enter values.

Compiling & Executing :



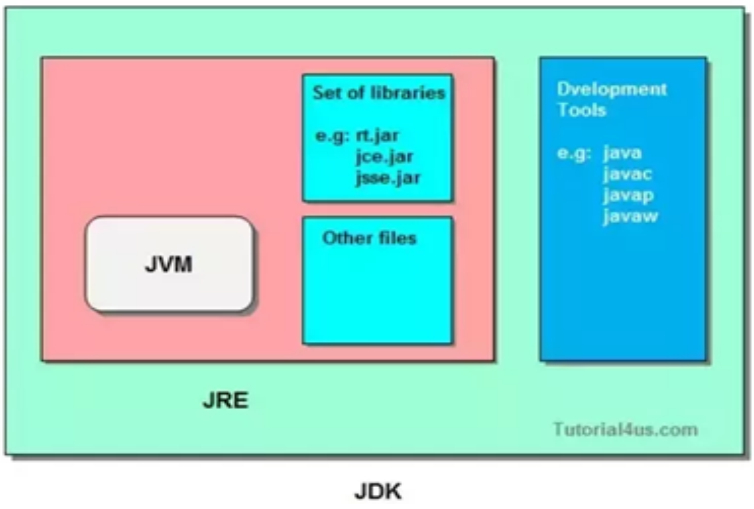
- For each class a byte code is created. Byte code is created with extension **.class** and every java file name should have **.java** extension.



- C++ Application uses runtime environment of Operating system whereas Java uses runtime environment of its own i.e JVM

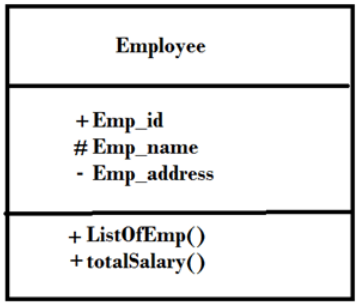
- JDK(Java Development Kit) contains JRE and development tools. JRE contains JVM (place where program translation occurs) library files (files that are important to the program) and other files (files which are OS specific, architecture specific/system specific).

- Program runs on hardware (OS) and JVM takes care of the execution & whole environment of the program. JRE takes care of the run time.



JAVA CLASS

A class has a Class name, class attributes (data members), class functions. Observe the class diagram below; An Object diagram has rounded corners and is similar to class diagram.



These notations (+, #, -) are fixed and are given by unified modelling language (UML). (/ is used for derived attribute)

Pascal Naming Convention : A class name Should begin with a capital alphabet, should contain no spaces, and should emphasize each new word with capital letters.

Adding Comments to a Java Class [Documentation]

Line comments start with two forward slashes (//) and continue to the end of the current line. Line comments do not require an ending symbol.

Block comments start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/). A block comment can appear on a line by itself, on a line before executable code, or on a line after executable code. Block comments also can extend across as many lines as needed.

Javadoc comments are a special case of block comments called documentation comments because they are used to automatically generate nicely formatted program documentation with a program named javadoc. Javadoc comments begin with a forward slash and two asterisks (/**) and end with an asterisk and a forward slash (*/).

Input and output functions

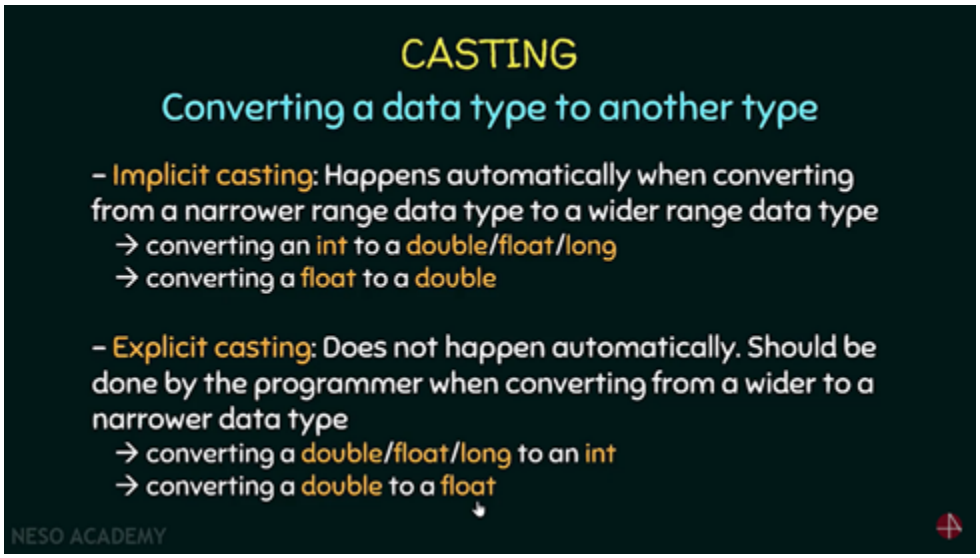
Output function: No special importation is required. **System.out.println("");**

Input function: we need to use import java.util.*; package prior to its usage. **Scanner s = new Scanner (System.in);** Here 's' is an object of class Scanner & System.in is the input source.

s. next_datatype(); //it's a command used to read values

But for character, we use s.next().charAt(0); & for String we use s.nextline();

Typecasting :



Explicit casting: int i1 = (int) 8.2;
int i2 = (int) 4L;

Integer.parseInt(); used to convert a string into integer.
Double.parseDouble(); used to convert a string into double.

Data Types

1.Primitive datatypes 2. Reference data types

Primitive data types:

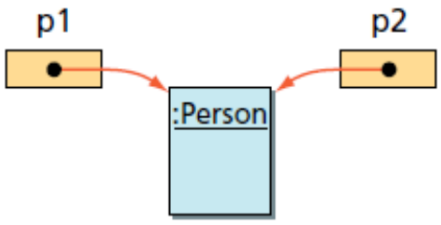
There are 8 primitive/value data types for which direct Declaration & Initialization can be done.

Data type	Number of bytes
byte	1
boolean	1
short	2
char	2
int	4
float	4
long	8
double	8

int, double can be assigned directly but when long/float are used we need to assign them ending with the suffix L/F

We don't need to allocate memory, and it doesn't have any members. values are independent of each other variables as both refer to different address locations.

Reference data types:



These include Object Reference.

We need to allocate memory using the "new" operator. Here, values are dependent on each other variables as both refer to one address location.

```
Person p1, p2;
p1 = new Person ();
p2 = p1;
```

```
Person p1=new Person();

//Declaration  Initialization Assignment
```

Here p2 = p1; indicates that p2 is a variable that is used to store the same address p1 does. Person is a memory created whose address is stored in p1, p2 variables.

static :

- ★ Any instance method can be invoked using an object. Any instance variable can be initialized by using an object.
- ★ Creation of many objects for the same class leads to wastage of memory (Memory efficient).
- ★ But use of static variables and methods doesn't require creation of an object.

static variable:

These are global variables. when objects are declared, all objects share the same static variable. These can't be changed during execution.

values are initialized for static variables in the static block. if values are not initialized then they will be 0 by default.

static block:

used to initialize static variables, A static block is executed only once when the class is loaded.

Static method:

They can directly access only static variables

Non-static methods can access any static method and static variable, whereas static method can't access any non-static method & non-static variable; they can access only static methods & static variables of their class.

Who's first?

At first the class gets loaded and static statements are run and then a static block (if any) is run & then the main function is called; which calls static methods in it.

this:

Keyword 'this' in Java is a reference variable that refers to the current object, whose method is being called upon.

It can be used to refer current class instance variable

It can be used to invoke or initiate current class constructor

It can be passed as an argument in the method call

It can be passed as argument in the constructor call

It can be used to return the current class instance

If it is not used then there will be no difference between local variable and instance variable and the local variable hides the instance variable this feature is known as **Shadowing**.

You can use "this" to avoid naming conflicts in the method/constructor of your instance/object.

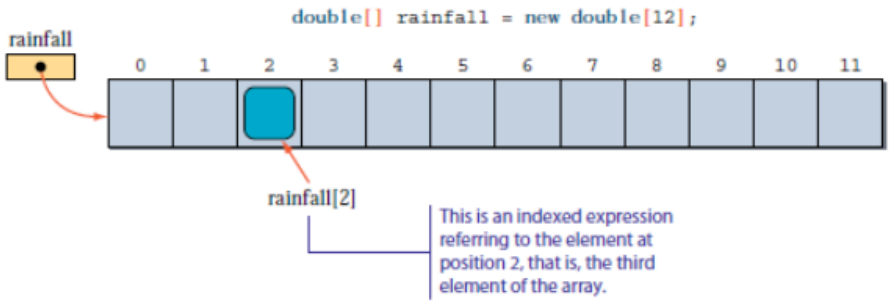
ARRAYS:

An array is a reference data type which is a collection of data values of the same type contiguously.

The way it is implemented is like a pointer in C. The new operator allocates the memory to store the elements continuously.

Declaration & creation:

```
double[] rainfall;    or    double rainfall[];          //Declaration
double[] rainfall = new double[12];                     //creation
```



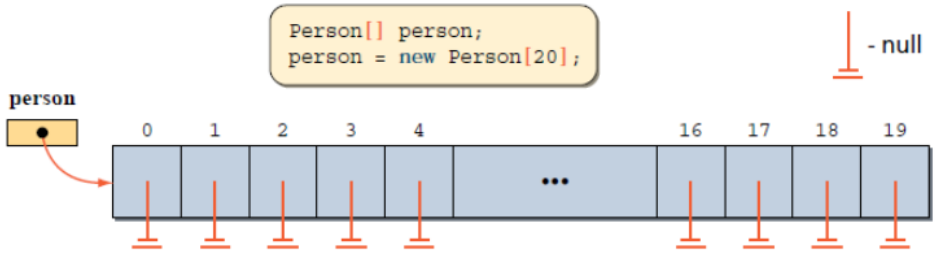
Here, rainfall is not the name of the array, it is a variable that stores the address of the 1st element of the array(rainfall[0])

Length of an array: It is a function represented by Array_name.length

Arrays in Java can be static as well as dynamic. We can add elements to an array during the course of the execution.

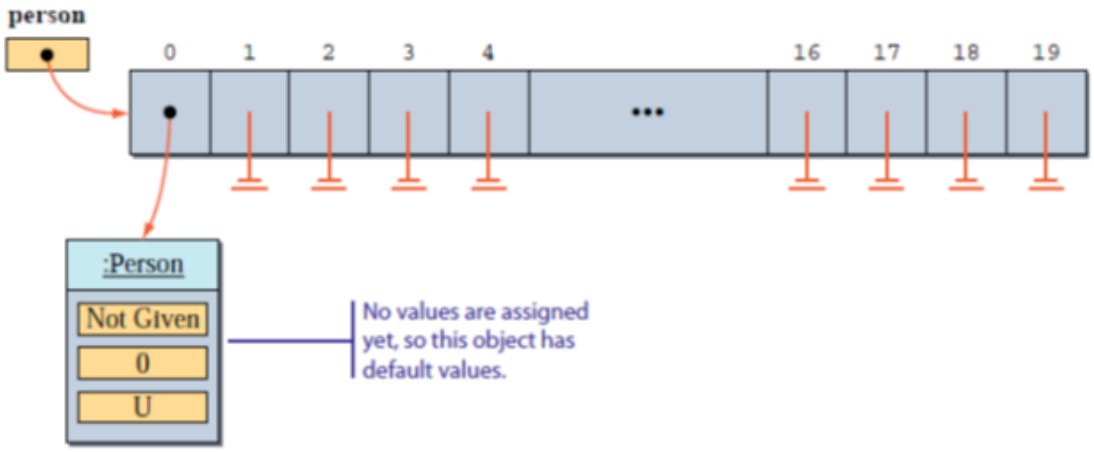
Array of Objects:

We know that arrays are declared in this fashion:

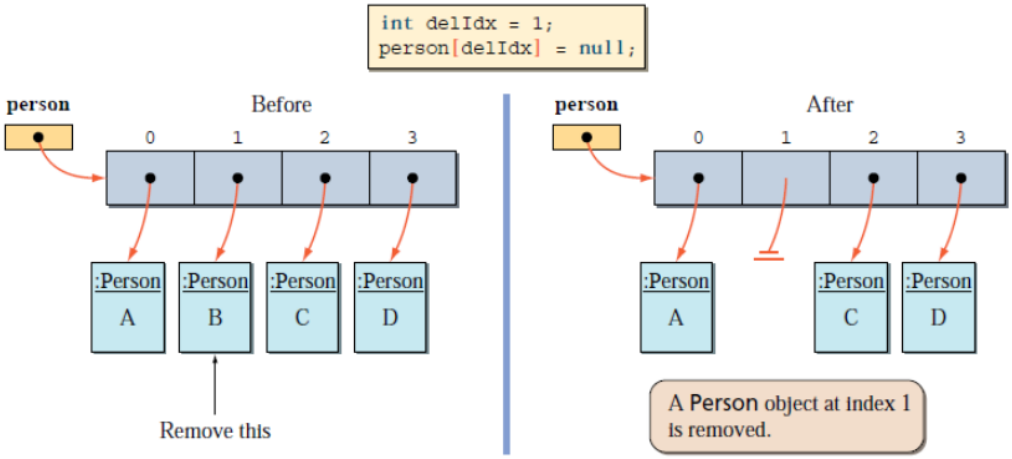


Notice that the elements of the array are not yet created; only the array is created and the elements are declared & yet to be created. Here, array elements are initially null. Each individual element is an object.

To create a Person object and set it as the array's first element, we write person[0] = new Person( );



Deletion in Array of Objects : We just need to make them null.

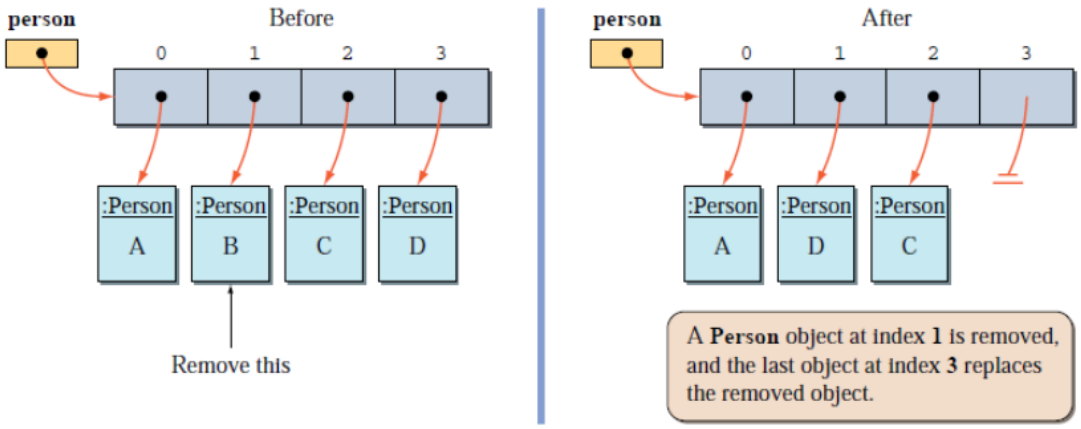


OR

```
int delIdx = 1;
int last   = 3;

person[delIdx] = person[last];

person[last]   = null;
```



for loop & for-each loop:

In the for loop, we can jump over the elements(updation by any leaps) in the way we want to but in for-each loop we can't jump we need to go sequentially (updation by only one).

In for loop termination and updation can be altered which is not possible with for-each loop as they are fixed. Its loop termination is till the last element of array and updation is only by 1.

The for-each loop is mainly used to iterate over Arrays , it only allows access to the elements. The elements cannot be changed. Although we cannot change the elements of an array, we can change the content of an object if the element is a reference to an object.

for loop	for-each loop

int[] A = {1,2,3,4,5,6};
int sum = 0;
 	
for(int i = 0; i<A.length; i++)
{
sum += A[i];
}
	int[] A = {1,2,3,4,5,6};
	int sum = 0;
	
	for(int k : A)
	{
	sum += k;
	}

★ String Access using for-each loop:

We cannot use the for-each loop directly to a string to access individual characters in it. However, the String class includes a method named " toCharArray " that returns a string data as an array of characters and we can use the for-each loop on this array of characters data.

Example :

```
String s = "Hi Koushik !";
char[] c = s.toCharArray();
```

```
for(char ch : c)
   {
       System.out.print(ch + " ");
    }
```

Passing Arrays to Methods:

At first, the actual parameters begin referring to a particular array in a memory location; once the function call is made and arguments are passed, the formal arguments also begin referring to the same array in the same memory location.
But, Once the formal parameter gets used inside the function it no longer refers to the array which is being referred by actual parameters. It starts to refer a new array in another memory location.

2D ARRAYS:

Declaration :

```
double[][]  payScaleTable =  new double[4][5];
```

```
payScaleTable = new double[4][ ];

payScaleTable[0] = new double[5];
payScaleTable[1] = new double[5];
payScaleTable[2] = new double[5];
```

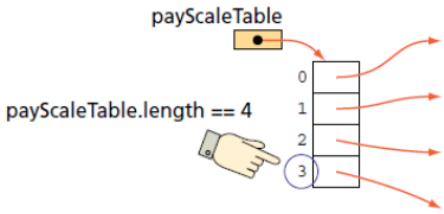
```
payScaleTable[3] = new double[5];
```

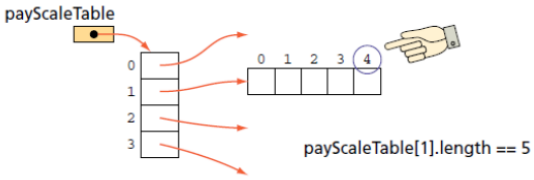
```
payScaleTable = new double[4][ ];


for (int i = 0; i < 4; i++)
 {
   payScaleTable[i] = new double[5];
 }
```

Lengths:

1. payScaleTable.length;
2. payScaleTable[i].length;





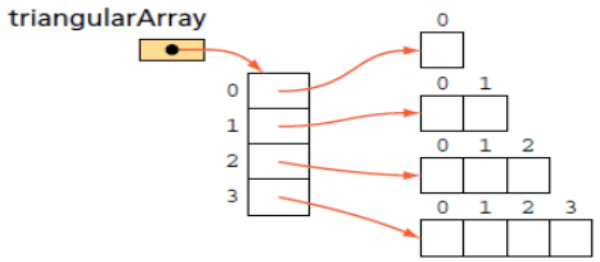
payScaleTable.length refers to the length of the payScaleTable array itself. It returns no. of rows/no. of one-dimensional arrays

payScaleTable[i]. length refers to the length of an array stored at ith row of payScaleTable. It returns no. of columns to the particular row.

Triangular Array :

Since we allocate the subarrays individually, we can create subarrays of different lengths [There may not be the same no. of columns for all the rows].

```
triangularArray = new double[4][ ];
for (int i = 0; i < 4; i++)
{
  triangularArray[i] = new double[i+1];
}
```



Inheritance:

Derived class (sub class /Child class) inherits the properties of Base class (Parent class /super class). Derived class extends a base class

Suppose two variables a1, a2 are present in class A and one variable b1 is present in class B then it means Class B has {a1, a2} (hidden), b1(seen). Total 2 variables with 8 bytes in class A and 1 variable with 12 bytes in class B. So, **based on size** also we can determine what inherits what.

★ Private data members (variables/methods) under no circumstances are inheritable. All the default, public, protected data members are inheritable (Obviously the ones available in base class).

★ Public constructors are also not inherited. Constructors are used to initializing objects of that class.

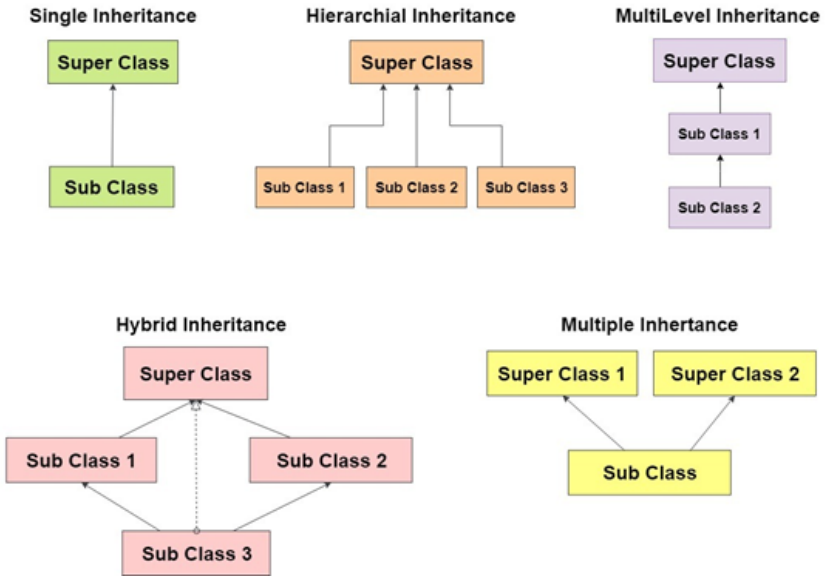
★ Multiple inheritance is not supported for classes in Java but is supported by using interfaces in Java. (why ?)

★ Also, too many levels (>7) of Multilevel inheritance is regarded as bad implementation.

★ Hybrid Inheritance is a combination of hierarchical and Multilevel inheritance.

★ Multiple Inheritance is avoided in classes to remove the ambiguity among the same variables in different parent classes.

Types of inheritance



Visibility Control : Access Specifier

Table 8.1 Visibility of Field in a Class

Access location \ Access modifier →	*public*	*protected*	*friendly (default)*	*private protected*	*private*
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

Constructor Invocation in Inheritance:

★ Once an Object gets created for the derived class; before the derived class constructors get executed its Base class constructors are executed.

super:

To access the variables or to invoke the constructors/methods of immediate base class, in derived class with the same values/name we use a key word called "super". It is always used in the derived class.

```
public class C {

private int idNumber;
private double balanceowed;

public C(int id, double bal)
{
   idNumber = id;
   balanceowed = bal;
}

public void display()
{
System.out.println( idNumber +balanceowed);
}
```

```
public class PC extends C {

double discountRate;

public PC (int id1, double bal1, double dis)
{
    super(id,bal);
   discountRate = dis;
}

public void display()
{
   super.display();
  System.out.println(discountRate);
}
```

super keyword line **should always be the first line** inside the derived constructor.[Since, Constructor Invocation in Inheritance]

Limitation:

super keyword can't be used inside a static method.

Method overriding:

Generally, Derived class can hide or override the base class members that include variables, methods, constructors.when a method defined in base class is again defined in the derived class then we say it as method overriding.

Limitation:

A non-static method cannot override a static member (method/variable).

Final:

Final methods:

Base class can mark some/all of its methods as Final methods and the final methods cannot be overridden.

Final classes:

A final class cannot have any sub classes i.e.; we cannot derive new classes from final classes.

abstract:

Method with no body is an **abstract method** and the class containing the abstract method is an **abstract class**.

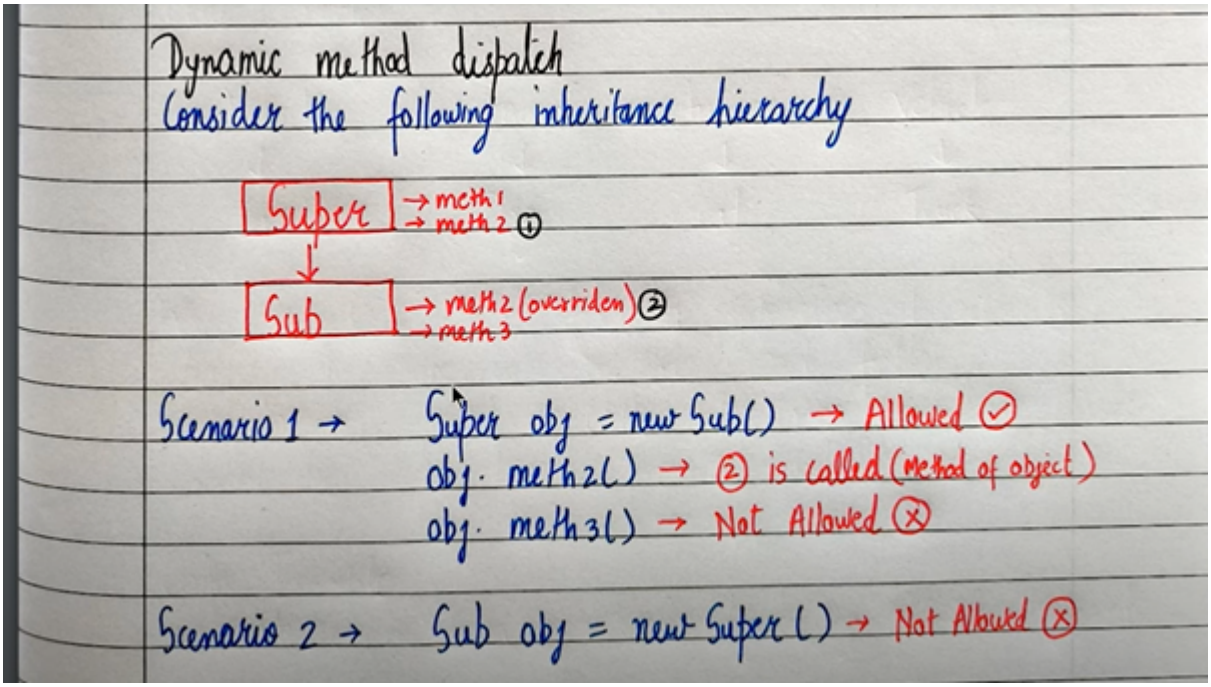
An abstract class can have both abstract and concrete methods. These concrete methods are implemented in the same class and simply derived, whereas the abstract methods **must** be implemented inside the derived class. We cannot create an object for an abstract class.[Abstract classes cannot be instantiated].

If the derived class doesn't implement the abstract methods in the base class then it'll become abstract by default.

Dynamic method dispatch:

A base class can always refer to a derived class object.

Super obj = new Sub();



Neglect the arrow mark(Inheritance representation)

Interface:

★ Interfaces are used where there is a generalised abstract thought that needs to be implemented to make sense.

★ Used for Multiple Inheritance. An Interface consists of only **final** data variables and **abstract** methods.

★ Every method declared in the interface must be defined in the class that implements the interface.

★ Access specifiers must be mentioned alongside method definition in the class that implements interface.

```
interface A
        {
                void Adisplay();
        }

interface B
        {
                void Bdisplay();
        }
```

```
class AB implements A,B
{
    public void Adisplay()
    {
        System.out.println("class Implements interface A");
    }

   public void Bdisplay()
    {
        System.out.println("class Implements interface B");
    }
}
```

★ The object created in the class that implements a number of interfaces is sufficient enough to invoke the methods defined in them(interfaces).

Multiple Inheritance is avoided in classes to remove the ambiguity among the same variables in different parent classes. But it is Possible with interfaces because it inherits the final/constant data values and abstract methods(methods without the body)

class	interface
Methods are declared and defined	Methods are only declared but not defined
A class has concrete methods	interface has abstract methods
Instantiation of class is done	No such Instantiation is done.
A class will have static, instance, final data variables	interface will only have final data variables by default.
There is no necessity of having an interface	There must be a class that implements the interface

Wrapper Classes:

1. Check the ppt
2. Just remember the int ones the rest follow the same way…. Wrapper classes are to be memorized as these are used in collection frameworks.

Strings:

It is often confusing to differentiate between empty Strings and null Strings. String variables cannot be changed; it may not show an error but they will be referred to at two different locations.

String word1 = ""; empty string
String word2 = null; null string
String word3; null string

Two strings can be added using a + sign and it is known as concatenation which is an example of operator overloading (polymorphism).

Length : It is a function represented by **string_name.length().**

Length of an empty string is 0.

string_name.charAt(pos); used to get the character at a given position

Difference between using and not using the **new** operator for String:

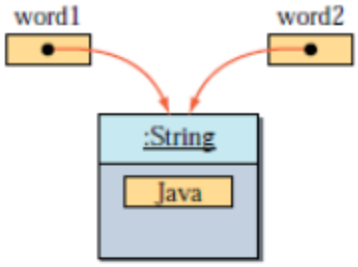
code	consequence
word1 = **new** String("Java"); word2 = **new** String("Java") ;	whenever **new** operator is used then both the strings refer to Different objects.
word1 = "Java"; word2 = "Java";	Here **new** operator is not used hence both the strings refer to Same objects.

String Comparisons:

string_name.equals(); used to compare the content of the string

If two strings refer to the same object then both the strings are equal. But, if both the strings are equal they need not refer to the same object.

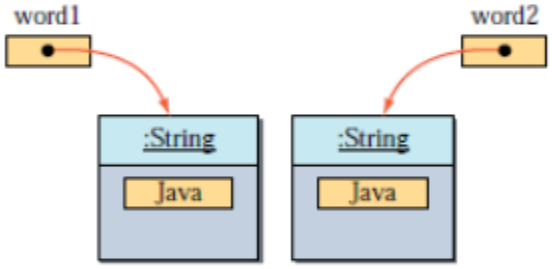
Case A: Referring to the same object.




```
word1 == word2          is true
word1.equals( word2 )   is true
```

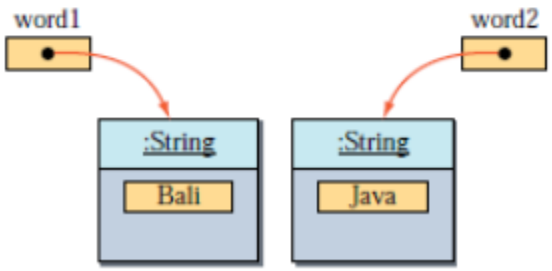
Note: If **x == y** is **true**, then **x.equals(y)** is also **true**. The reverse is not always true.

Case B: Referring to different objects having identical string values.




```
word1 == word2          is false
word1.equals( word2 )   is true
```

Case C: Referring to different objects having different string values.




```
word1 == word2          is false
word1.equals( word2 )   is false
```

Stringbuffer Class:

It is used to modify the strings in the same memory location.The following are the methods to modify the string

Method	*Task*
`s1.setChartAt(n, 'x')`	Modifies the nth character to x
`s1.append(s2)`	Appends the string s2 to s1 at the end
`s1.insert(n, s2)`	Inserts the string s2 at the position n of the string s1
`s1.setLength(n)`	Sets the length of the string s1 to n. If `n<s1.length()` `s1` is truncated. If `n>s1.length()` zeros are added to `s1`

```
public class StringBufferExample {

	public static void main(String[] args) {
		StringBuffer sb = new StringBuffer("Hello");
		sb.append("Gara");
		System.out.println(sb);
	}

}
```

String Applications :

Pattern Matching with Regular Expressions, Bioinformatics

1.Pattern Matching with Regular Expressions

★ The brackets [ ] are used here to represent **choices among** the brackets. so[123] means 1 or 2 or 3 Similarly, [abc] means a, b, or c.

★ The hyphen in the brackets shows the **range**, so [1-7] means any digit from 1 to 7.

★ The notation is **case-sensitive**. If we want to allow any lowercase letter, then the regular expression will be [a-z].

★ For any character including digits, alphabets and symbols the hat symbol ^ is used for **negation**. For example, [^abc] means any character except a, b, c.

Expression	Description
[013]	A single digit 0, 1, or 3.
[0–9][0–9]	Any two-digit number from 00 to 99.
A[0–4]b[05]	A string that consists of four characters. The first character is A. The second character is 0, 1, 2, 3, or 4. The third character is b. And the last character is either 0 or 5.
[0–9&&[^4567]]	A single digit that is 0, 1, 2, 3, 8, or 9.
[a–z0–9]	A single character that is either a lowercase letter or a digit.

Quantifiers:

We can specify the minimum and maximum numbers of repetitions also.

Expression	Description
X{N}	Repeat X exactly N times, where X is a regular expression for a single character.
X{N,}	Repeat X at least N times.
X{N,M}	Repeat X at least N but no more than M times.

We can use repetition symbols *** (0 or more times)** or **+ (1 or more times)** to designate a sequence of unbounded length.

In regular expression, we can state this definition as [a-zA-Z][a-zA-Z0-9_$]*

Regex Metacharacters:

Here are the common backslash symbols used in regular expressions:

Expression	String Representation	Description
\d	"\\d"	A single digit. Equivalent to [0–9].
\D	"\\D"	A single nondigit. Equivalent to [^0–9].
\s	"\\s"	A white space character, such as space, tab, new line, etc.
\S	"\\S"	A non-white-space character.
\w	"\\w"	A word character. Equivalent to [a–zA–Z_0–9].
\W	"\\W"	A nonword character.
\b	"\\b"	A word boundary (such as a white space and punctuation mark).
\B	"\\B"	A nonword boundary.

For the usage of regular expressions (regex) we need to import the package **import.java.util.regex.*;**

Representation :

way 1

```
Pattern p= Pattern.compile(desired_pattern);
Matcher m= p.matcher(sequence of characters);
boolean b=m.matches();
```

way2

```
boolean b= Pattern.compile(desired_pattern).matcher(sequence of characters).matches();
```

way3

```
boolean b= Pattern.matches(desired_pattern, sequence of characters);
```

It gives a boolean expression as a result verifying whether the sequence of characters matches the desired pattern or not. This matching occurs character wise (character to character).

replaceAll() method:

Using the replaceAll method, we can replace all occurrences of a substring that matches a given regular expression with a given replacement string.

str.replaceAll(oldstring, newstring);

Expression	Description
`str.replaceAll("OOP", "object-oriented programming")`	Replace all occurrences of OOP with object-oriented programming.
`str.replaceAll( "[0-9]{3}-[0-9]{2}-[0-9]{4}", "xxx-xx-xxxx")`	Replace all social security numbers with xxx-xx-xxxx.
`str.replaceAll("o{2,}", "oo")`	Replace all occurrences of a sequence that has two or more of letter o with oo.

str.replaceAll("\\btemp\\b", "temporary");

The regular expression we want here is \btemp\b . To put it in a String representation, we write \\btemp\\b.

Bioinformatics:

DNA GC-content, Transcription.

The GC-content is simply the percentage of guanine (G) and cytosine (C) in the DNA sequence. If we let the expressions C_A, C_G, C_C, and C_T stand for the count of adenine, guanine, cytosine, and thymine, respectively, in the DNA sequence, then the GC-content is computed as follows:

The length of the string that represents a DNA sequence is equal to $C_A+C_T+C_G+C_C$.

DNA transcription is a process of synthesizing an RNA (ribonucleic acid).

So, we transcribe a given DNA to an RNA by creating a new string that replaces Ts in the given DNA with Us