OBJECT ORIENTED PROGRAMMING THROUGH JAVA

G JAYA KRISHNA
ASSISTANT PROFESSOR
DEPARTMENT OF CSE
RGUKT NUZVID

M VEDAVYAS
CSE-05, 2020 BATCH
RGUKT NUZVID

COURSE OF CONTENT

INTRODUCTION TO PROGRAMMING LANGUAGE

0.1 NEED OF PROGRAMMING LANGUAGE -For Video

- Languages like Telugu, Hindi, English.... are used to communicate people among them. Likewise to communicate with computer we need programming language.
- A programming language is a set of instructions that can be used to interact with and control a computer.

0.2 TYPES OF PROGRAMMING LANGUAGES

There are three levels of programming languages are available.
They are:
1. Machine language (Low-level Language)
2. Assembly language (Symbolic Language)
3. High level language

1.MACHINE LANGUAGE

- A language that is directly understood by the computer without any translation is called Machine Language.
- It is also called as Binary Language and it is a Low Level Language.
- Programs and applications written in low-level language are directly executable on the computing hardware without any interpretation or translation.

  Advantages :

  - Execution speed is very fast.
  - Efficient use of primary memory.
  - It does not require any translation.

  Disadvantages :

  - **Machine dependent :** The machine language is different for different types of computers.
  - **Difficult to write program :** because it requires memorizing dozens of opcodes for different commands.
  - **Error prone :** Difficult to modify.

2.ASSEMBLY LANGUAGE

- It is also low-level programming language.
- Assembly language consists of special codes called mnemonics to represent the language instructions instead of 0's and 1's.
  For example : ADD, SUB, MUL, JMP, LOAD.
- The software used to convert an assembly language programs into machine codes is called an assembler.

  Advantages :

  - It allows complex jobs to run in a simpler way.
  - It is memory efficient, as it requires less memory.

– It is easier to correct errors and modify program instructions.

Disadvantages :

– It is very difficult to remember all the mnemonics.
– It requires assembler and it is time consuming.
– It is machine dependent.

3.HIGH LEVEL LANGUAGE -For Video

- It is more like human language and less like machine language.
- There are many high-level languages in use, including BASIC, C, C++, COBOL, FORTRAN, Java, Pascal, Perl, PHP, Python, Ruby and Visual Basic.
- High-level programming languages can be classified into the following :

Procedural Oriented Programming	**Object-Oriented Programming**
Program is divided into small parts called functions.	Program is divided into small parts called objects.
It follows Top-Down approach	It follows Bottom-Up approach
It has no access specifiers	It has access specifiers like private,public,protected,..
It is less secure	It is more secure
Overloading is not possible here.	Overloading is possible .
It is based on the unreal world.	It is based on the real world.
There is no concept of data hiding and inheritance.	The concept of data hiding and inheritance is used.
Examples: C, FORTRAN, Pascal, Basic, etc.	Examples: C++, Java, Python, etc.

0.3 PARTS OF PROGRAMMING LANGUAGE : -For Video

STATEMENT : It is a command given to the computer that expresses some action to be carried out.
PROGRAM : Collection of statements is called as a program.
SOFTWARE : Collection of programs is called as a software.

NEED OF SOFTWARE : To automate the task and to reduce the errors in task.

TYPES OF SOFTWARE :

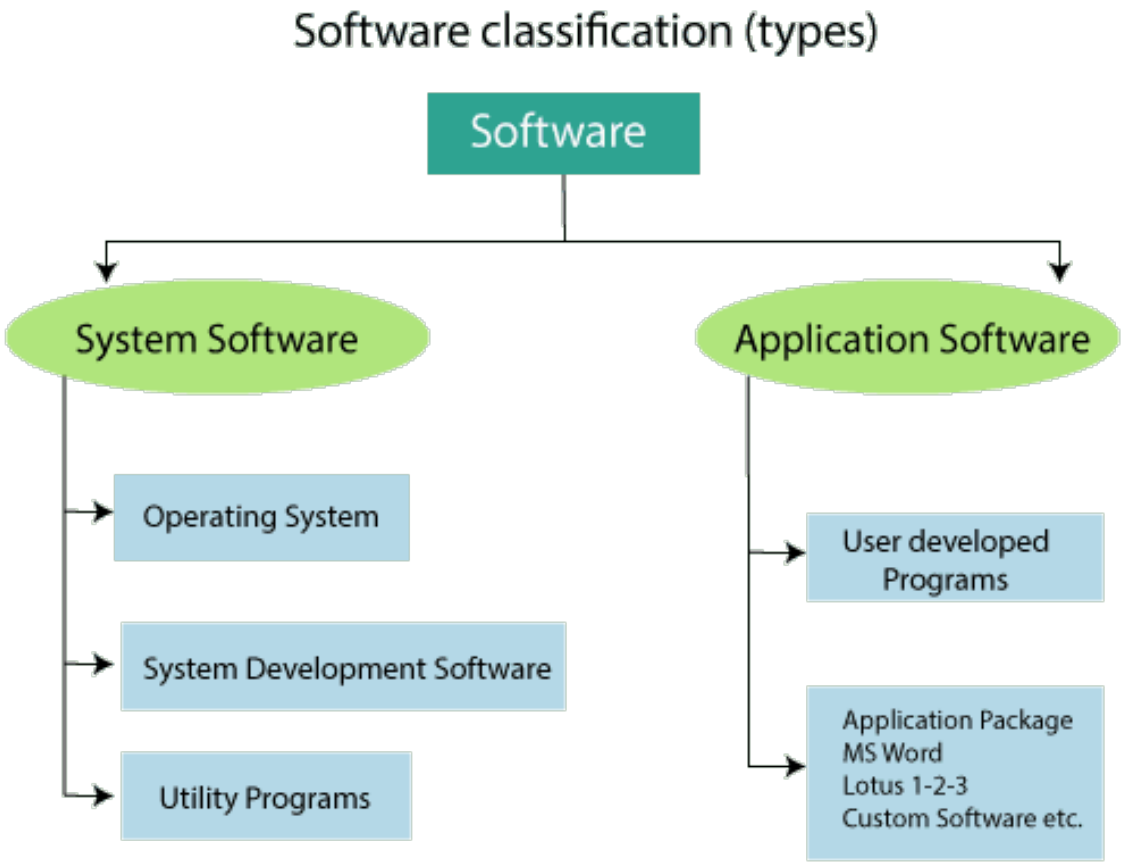
- Softwares are two types. They are
  i.System software
  ii.Application software

i. SYSTEM SOFTWARE :

- It is designed for systems only and it cannot interacted by human beings.
- It is written in a low-level language in general and is fast in speed. EX : O.S, Drivers, Compilers etc.....

ii. APPLICATION SOFTWARE :

- It is designed to perform a specific task for end-users.
- It requires more storage space and written in high-level language.
- It includes word processors, spreadsheets, database management, inventory, payroll programs, etc.
- It is classified into the following:
  a. Standalone software
  b. Distributed software



a. STANDALONE SOFTWARE :

- It have a specific solution to provide for the end-user.
- Here the result obtained is not sharable.
- Some examples are Notepad, Calculator, Microsoft Word, Google Chrome, etc.....

b. DISTRIBUTED SOFTWARE :

- This software effects to group of people.
- Here the result obtained can be sharable.
- Some examples are browsers, facebook, whatsapp,gmail, etc.....
- This is again classified as
  A. Web Applications
  B. Enterprise Softwares

  A. WEB APPLICATIONS :
  - This software will be accessed by using a web browser.
  - Web applications are delivered on the World Wide Web to users with an active network connection.
  - Some of these are Google Apps, Google Drives and Microsoft 365.

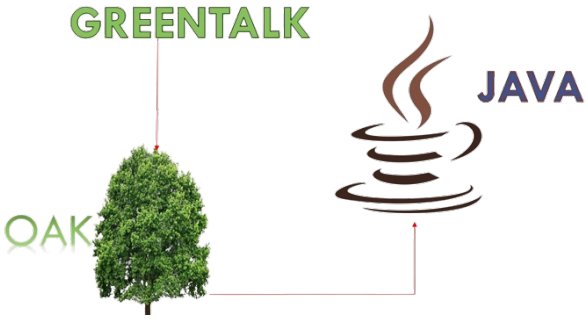
  B. ENTERPRISE SOFTWARES :
  - It is designed specifically for large organizations and businesses.
  - It needs usage of other instruments.
  - EX : ATM machines, Swiping machines.....

UNIT-1. INTRODUCTION TO JAVA

1.1 HISTORY OF JAVA

-For Video

- Java is a High-level Object-Oriented programming language developed by James Gosling in the early 1990s.
- The team initiated this project to develop a language for digital devices such as set-top boxes, television, etc.
- James Gosling and his team(green team) called their project as "Greentalk" and its file extension was .gt and later became to known as "OAK".



James Gosling

- The name Oak was used by Gosling after an oak tree that remained outside his office.
- Oak is an image of solidarity and picked as a national tree of numerous nations like the U.S.A., France, Germany, Romania, etc.
- Oak was already a trademark by Oak Technologies, so due to copyright issues the team searched for another name.
- They came up with several names such as JAVA, DNA, SILK, RUBY, etc.
- Java name was decided after much discussion since it was so unique.
- Gosling came up with this name while having a coffee near his office and also Java is the name of an island in indonesia.
- Java was created on the principles like Robust, Portable, Platform Independent, High Performance, Multithread, etc.
- Currently, Java is using in internet programming, mobile devices, games, e-business solutions, etc.

TIMELINE OF JAVA

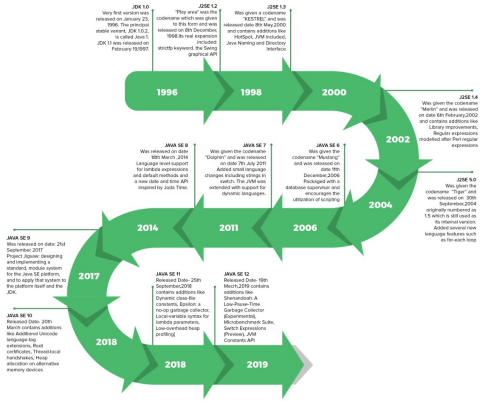


Table 1: **DIFFERENCES BETWEEN JAVA AND PYTHON**

TOPIC	JAVA	PYTHON
Compilation process	Java is both compiled and interpreted language	Python is an interpreted language
Code Length	Longer lines of code as compared to python.	3-5 times shorter than Java programs.
Syntax Complexity	Define particular block by curly brackets, end statements by semicolon.	No need of semi colons and curly braces, uses indentation
Ease of typing	Strongly typed, need to define the exact datatype of variables	Dynamic, no need to define the exact datatype of variables.
Speed of execution	Java is much faster than python in terms of speed.	Expected to run slower than Java programs
Multiple Inheritance	Multiple inheritance is partially done through interfaces	Provide both single and multiple inheritance

1.2 JAVA TERMINOLOGY

-For Video

Before learning Java, one must be familiar with these common terms of Java.

1. JAVA DEVELOPMENT KIT(JDK)

- It contains compiler, Java Runtime Environment (JRE), java debuggers, java docs, etc.
- For the program to execute in java, we need to install JDK on our computer in order to create, compile and run the java program.

2. JAVA RUNTIME ENVIRONMENT(JRE)

- JRE installation on our computers allows the java program to run, however, we cannot compile it.
- JRE includes a browser, JVM, applet support, and plugins.
- For running the java program, a computer needs JRE.

3. JAVA VIRTUAL MACHINE(JVM)

- There are three execution phases of a program. They are written, compile and run the program.
  - Writing a java program by us.
  - The compilation is done by the JAVAC compiler which is a primary Java compiler included in the Java development kit (JDK). It takes the Java program as input and generates bytecode as output.
  - In the Running phase of a program, JVM executes the bytecode generated by the compiler.
- Every Operating System has a different JVM but the output they produce after the execution of bytecode is the same across all the operating systems.

4. BYTE CODE

- The Javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. It is saved as .class file by the compiler. To view the bytecode, a disassembler like javap can be used.

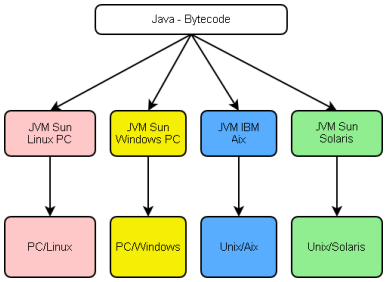


Fig:1 JVM AND OUTPUT

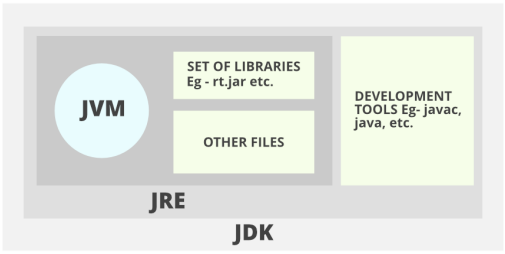


Fig:2 JDK

5.JUST INTIME COMPILER(JIT)

- In order to improve performance, JIT compilers interact with the Java Virtual Machine (JVM) at run time and compile suitable bytecode sequences into native machine code.

6. CLASS PATH

- The classpath is the file path where the java runtime and Java compiler look for .class files to load. By default, JDK provides many libraries. If you want to include external libraries they should be added to the classpath.

7. CLASS LOADER SUBSYSTEM

- It is mainly responsible for three activities like Loading, Linking and Initialization

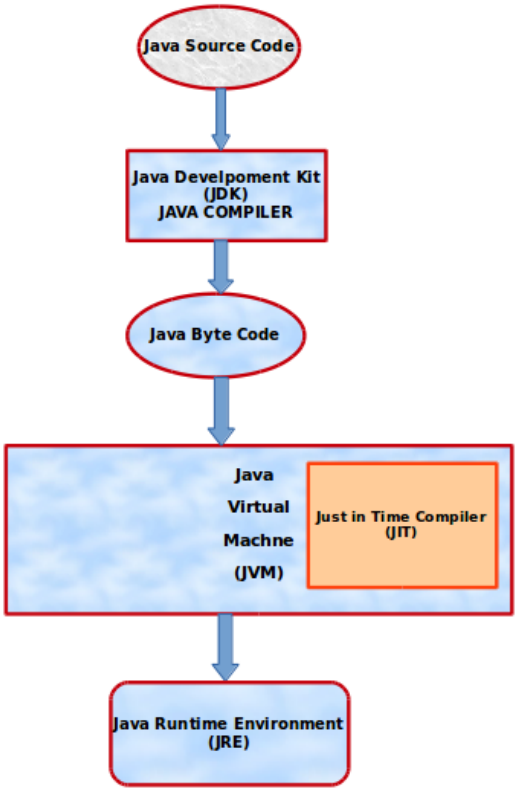
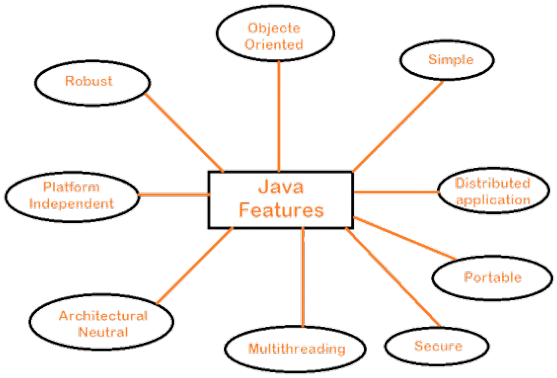


fig:3 PROCESS OF JAVA PROGRAM

1.3 FEATURES OF JAVA

-For Video

I 1. SIMPLE



- Java is one of the simple languages as it does not have complex features like pointers, operator overloading, multiple inheritances, and Explicit memory allocation.
- The Developing time as well as Execution time of java is less.
- Creating and deleting memory is easy in java.
- Java supports Multiple Inheritence through Interfaces but not supports through Classes.
- It is a rich set of API [plenty of Libraries].

2. OBJECT ORIENTED PROGRAMMING LANGUAGE

- The four main concepts of Object-Oriented programming are :
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
- Due to the presence of above principles, java is objected oriented.

3. PLATFORM INDEPENDENT

- Compiler converts source code to bytecode and then the JVM executes the bytecode generated by the compiler.
- Each operating system has a different JVM, but the output produced by all the OS is the same after the execution of the bytecode.
- That is why we call java a platform-independent language.
- But jvm is platform dependent.

4. ROBUST

- Java language is robust which means reliable.
- It puts a lot of effort into checking errors as early as possible than other programming languages.
- The main features of java that make it robust are garbage collection, Exception Handling, and memory allocation.

5. SECURE

- In java, we don't have pointers, so we cannot access out-of-bound arrays.
- That's why several security flaws like stack corruption or buffer overflow are impossible to exploit in Java.
- java programs run in an environment that is independent of the os(operating system) environment which makes java programs more secure.

6. ARCHITECTURAL NEUTRAL

- Java is architecture neutral because there are no implementation dependent features.
- For example, the size of primitive types is fixed [int data type occupies 4bytes for both 32 bit and 64 bit architecture].
- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture.

7. INTERPRETER

- JVM executes the bytecode line by line that's why java is interpreted.

8. HIGH PERFORMANCE

- Java offers high performance as it used the JIT (Just In Time) compiler.
- Bytecodes generated by the Java compiler are highly optimized, so Java Virtual Machine can execute them much faster.

9. MULTITHREADING

- Java supports multithreading.
- It allows concurrent execution of two or more parts of a program for maximum utilization of the CPU.

10.DISTRIBUTED

- Java facilitates user to create distributed applications.
- The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection.

11. PORTABLE

- Java code written on one machine can be run on another machine so it is portable.
- We can write java code once and run anywhere(WORA).

IS JAVA HUNDRED PERCENT OBJECT ORIENTED PROGRAMMING LANGUAGE ???

- Java is not 100 percent Object oriented language.
- It follows the four principles of object oriented language like abstraction, encapsulation, inheritance and polymorphism.
- But due to existence of primitive data types, java is not 100 percent object oriented.

1.4 ELEMENTS/TOKENS OF JAVA -For Video

IDENTIFIERS

- Identifiers are used to identify an element uniquely.
- In Java, an identifier can be a class name, method name or variable name.
- **Rules for defining Java Identifiers**
  1. The allowed characters for identifiers are all alphanumeric characters([A-Z],[a-z],[0-9]), $(dollar symbol) and _(underscore).
     Ex : java*123 [INVALID]
  2. Identifiers should not start with digits[0-9].
     Ex : 123java [INVALID]
  3. Java identifiers are case-sensitive.
  4. Keywords can't be used as an identifier.
- Examples for valid Identifiers.
  vedavyas$1110
  veda_vyas
  v
  v1
  $v5
- Examples for invalid identifiers.
  1variable //started with digit
  veda vyas //contains space
  five* // can't use asterisk

VARIABLES

- A variable is a name given to a memory location.
- The value stored in a variable can be changed during program execution.
- In java, all variables should be declared before use.
- We can declare a variable as follows :
  Data_type variable_name;
  EX : int age;
- We can initialize a variable as follows :
  Data_type variable_name=value;
  Examples :
  int age=19;
  float average=99.5f;
  char letter='v';
- Rules for naming a variable are same as identifiers.
- we initialize the variables whenever they are common to entire program.
  Ex : int evenprime=2

KEYWORDS

- Keywords are predefined words and Keywords define functionalities.
- The following are keywords in java.

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

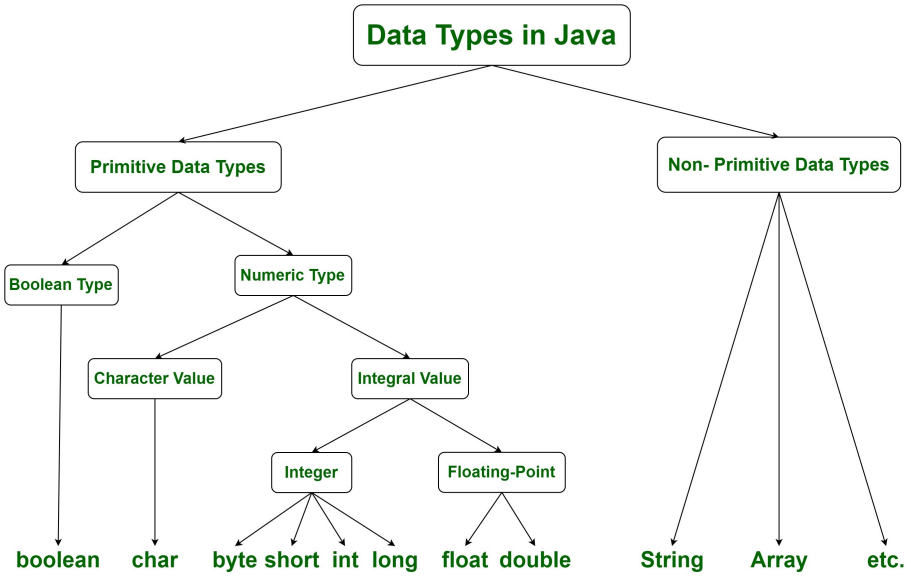
OPERATORS

1.Aritnmetic operators : +, -, *, /, %
2.Unary operators : Need only one operand and used to increment or decrement a value.[++, - -]
3.Assignment operators : +=, -=, *=, /=, %=
4.Relational operators : ==, <=, >=, !=
5.Logical operators : &&, ||, !
6.Ternary operators : condition ? if true : if false
7.Bitwise operators : &, |,<<, >>

Operators	Associativity	Type
++ --	Right to left	Unary postfix
++ -- + - ! (type)	Right to left	Unary prefix
/ * %	Left to right	Multiplicative
+ -	Left to right	Additive
< <= > >=	Left to right	Relational
== !=	Left to right	Equality
&	Left to right	Boolean Logical AND
^	Left to right	Boolean Logical Exclusive OR
|	Left to right	Boolean Logical Inclusive OR
&&	Left to right	Conditional AND
||	Left to right	Conditional OR
?:	Right to left	Conditional
= += -= *= /= %=	Right to left	Assignment

DATA TYPES

- Type of data we stored is called data type.
- Data types are two kinds :
  i. Primitive datatypes
  ii. Non Primitive datatypes



The following table shows the default values, size and range of the primitive datatypes :

Type	Default	Size	Range of values	Examples
Boolean	false	1 bit	true, false	true, false
Byte	0	8 bits/1 byte	-2^7 to 2^7-1	5,-126,0,122
Char	\u0000	16 bits/2 bytes	0 to 255	'a', 'v', '5'
Short	0	16 bits/2 bytes	-2^{15} to $2^{15}-1$	-32767,0,1,10,100,1000,10000,32767
Int	0	32 bits/4 bytes	-2^{31} to $2^{31}-1$	-2147483648,-2,-1,0,1,2,2147483648
Long	0	64 bits/8 bytes	-2^{63} to $2^{63}-1$	-2L,-1L,0L,1L,2L
Float	0.0f	32 bits/4 bytes	upto 7 decimal digits	1.23e100f , -1.23e-100f , .3f ,3.14F
Double	0.0	64 bits/8 bytes	upto 16 decimal digits	1.23456e300d , -123456e-300d , 1e1d

1.5 CLASS AND OBJECT

-For Video

LOGICAL ENTITY :

- Things which we can't touch but feel are called Logical Things.
- These are also called as General forms.

PHYSICAL ENTITY :

- Things which we can touch and feel are called Physical Things and also called as Physical forms.

GENERAL FORMS	PHYSICAL FORMS
Student	Boy,Girl
Fruit	Mango,Orange
Car	BMW,Audi
Flower	Rose,Jasmine
Mobile	oppo,MI
Vehicle	Car,Bike

CLASS :

- It is an user derived datatype which explains the logical things of existence(Real Logical Entity).
- Class is a collection of Data Members (Variables) and Methods(Functions).
- A class is also called as Blue print or Architecture.
- No memory is allocated when a class is declared.
- **Structure of class :**

Class classname{ Data Members Methods }

classname
Data Members(variables)
Methods(Functions)

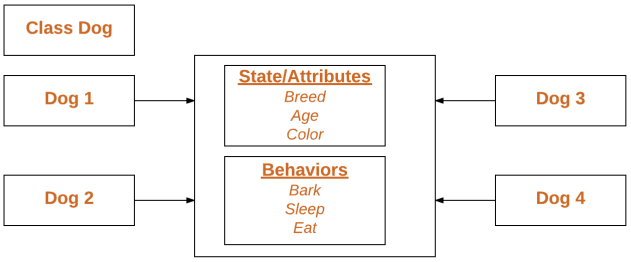
- **Rules for class name :**
  - First letter of all the words present in classname is capital.
    Ex: ClassName, FirstLetterInAllWords, JavaLanguage etc.....
- **Rules for method name in class :**
  - First letter of first word is small and first letter of subsequent words is capital.
    Ex: className, firstLetterInFirstWord, javaLanguage etc.....
  - This rule is called Hungarian notation or Camel case.
- A class in java can contain data member, method, constructor, nested class and interface.
- **Example of class :**

```
class Student
{
        public static void main(String args[])
        {
                int id;                 // data member
                String name;            // data member
                void study()            // method
                {
                        System.out.println("Studying");
                }
        }

}
```

OBJECT : -For Video

- Object is an instance of a class and a real world entity which has state and behaviour.
- Memory is allocated as soon as an object is created.
- By using a keyword "new", we create object to a class.
- When an object of a class is created, the class is said to be instantiated.
- A single class may have many number of instances(objects) as per our requirement.
- All the instances(objects) share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object.



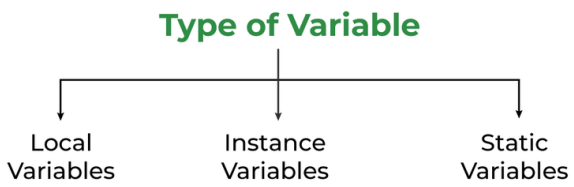
- In the above diagram, Dog1, Dog2, Dog3, Dog4 are objects to the class Dog.
- All objects get memory in Heap memory area.
- Objects are two types.
  i.Unreferenced objects
  ii.Referenced objects
- **Object Declaration :** classname objectname;
- **Object Initialization :** classname objectname=new classname();
- Example :

```
class Animal
{
        void dog()
        {
                System.out.println("Dog is Barking");
        }
}
class main{
        public static void main(String[] args)
        {
                Animal obj1= new Animal();
                Animal obj2= new Animal();
                obj1.dog();
        }
 }
```

output : Dog is Barking

1.6 TYPE OF VARIABLES AND THEIR SCOPE -For Video

The following are three types of variables.



1.LOCAL VARIABLES :

- A variable defined within a block or method or constructor is called a local variable.
- The scope of local variables is **within the block only**.
- Initialization of the local variable is mandatory before using it in the defined scope.

2.INSTANCE VARIABLES :

- These are declared inside a class and outside of a method but not prefix with any keyword.
- The scope of instance variables is **with in the class**.
- These variables are created when an object of the class is created and destroyed when the object is destroyed.
- Initialization of an instance variable is not mandatory. Its default value is dependent on the data type of variable.

3.STATIC VARIABLES :

- These variables are declared inside a class and outside of a method and prefix with **static** keyword.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Static variables cannot be declared locally inside an instance method and its scope is **with in the class**.

Example :

```
class Access
 {
        int a;
        static String name="Vedavyas";
        void age()
        {
                int age=19;
                System.out.println("AGE : "+age);
        }
 }
 class Variables
 {
        public static void main(String k[])
        {
                Access o=new Access();
                System.out.println("NAME : "+Access.name);
                System.out.println("a : "+o.a);
                o.age();
        }
 }
```

Output :

```
NAME: Vedavyas
 a: 0
 AGE: 19
```

1.7 METHOD AND TYPES OF METHODS -For Video

METHOD :

- A method is like a function i.e. used to expose the behavior of an object.
- It is a set of codes that perform a particular task.
- **Syntax of a method :**
  <access_modifier> <return_type> <method_name>(list_of_parameters)
  {
  body
  }
- **Method Signature :** It consists of the method name and a parameter list (number of parameters, type of the parameters, and order of the parameters).
- In the multi-word method name, the first letter of each word must be in uppercase except the first word.
  Ex : methodName, javaLanguage, etc.....
- **Method overriding :** In method overriding, method heading is same but body is different.
- **Method overloading :** In method overloading, only method name is same but parameter list is different.

TYPES OF METHODS :

- The methods are of two types:
  i.Void Method.
  ii.Non void Method.
- **i.Void Method :** If a method not returning anything then it is called "Void Method".
- **ii.Non Void Method :**
  - If a method returning anything then it is called "Non Void Method".
  - It returns anykind of data like 'Int','Float', 'Double', 'Object'.
- **Factory Method :** If a particular method returning object as a returntype then it is called Factory Method.
- In another classification methods are classified as Non static/Instance method and Static method.
- **Instance method :** If we can override a method then it is called Instance method/Non-static method.
- **Static method :** A method which is common to the entire program and can't override is called Static Method.
- **Rules :**
  1. One static method can call another static method directly within the class.
  2. One instance method can call another instance method directly within the class.
  3. One instance method can call static method by using classname.
  4. One static method can call instance method by creating an object.

- **Example :**

```
class Method
{
        void display()
        {
                display1();     //Rule-2
                System.out.println("Instance Method");
        }
        static void show()
        {
                System.out.println("Static method");
        }
        void display1()
        {
                Method.show();     //Rule-3
        }
        public static void main(String args[])
        {
                Method obj=new Method();
                obj.display();     //Rule-4
                show();             //Rule-1
        }
 }
```

Output :

```
Static method
Instance Method
Static method
```

1.8 ACCESS MODIFIERS -For Video

- Access modifiers help to restrict the scope of a class, constructor, variable or method.
- **Non Access modifiers :**
  1. Final
  2. Abstract
  3. Static
  4. Synchronized
  5. Native
  6. Tansient
  7. Volatile
- **Access modifiers :**
  1. Private
  2. Default
  3. Protected
  4. Public
- **1. Private :** The methods or data members declared as private are accessible only within the class in which they are declared.
- **2. Default :** The data members, classes, or methods that are not declared using any access modifiers i.e. having default access modifiers are accessible only within the same package.

- **3. Protected :** The methods or data members declared as protected are accessible within the same package or subclasses in different packages.
- **4. Public :** Classes, methods, or data members that are declared as public are accessible from everywhere in the program. There is no restriction on the scope of public data members.

	Private	Default	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package class	No	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Other package class	No	No	No	Yes
Other package subclass	No	No	Yes	Yes

1.9 STRUCTURE OF JAVA PROGRAM -For Video

Documentation section
Package declaration
Import Statements
Class definition
Main method

1. DOCUMENTATION SECTION

- In this section we can write Author name, Date, Day, Outcome of the program,etc....
- This section is not executable and it is optional section.
- We use comments like single line comments // and multiline comments /*.....*/ to write this section.

2. PACKAGE DECLARATION

- In this section, we declare the package name in which the class is placed.
- This section is also optional.
- We use the keyword package to declare the package name.

3. IMPORT STATEMENTS

- The import statement represents the class stored in the other package.
- We use the import keyword to import the class. Import is a pre-defined keyword in java which is meant for to get respective code from respective folder.
- we can use multiple import statements.
  Ex :
  import java.util.Scanner; //it imports the Scanner class only
  import java.util.*; //it imports all the class of the java.util package
  import java.lang.*//It imports all the classes within java. util package.

4. CLASS DEFINITION

- We known that class is a blue print, without the class we cannot create any Java program.
- A Java program may conation more than one class definition.
- Every Java program has at least one class that contains the main() method.

5. MAIN METHOD

- It is essential for all Java programs. Because the execution of all Java programs starts from the main() method.
- It must be inside the class. Inside the main method, we create objects and call the methods.
- It is also called as Program Driver.

PROGRAM DRIVER

- It is a main method where the java program starts execute.
- **Syntax :**
  public static void main(string args[])
  {
  body
  }
- It accepts the array of arguments of strings.
- **Why Strings as parameters in main method??**
  - It is easy to convert a string into another datatypes as it is difficult to convert other datatypes into strings that's why strings are taken as parameters.
  - We can able to access Unlimited parameters.
- **Why the main method is void??**
  - The main method does not returning anything, so it is void method.
- **Why the main method is static??**
  - At initial stage JVM looks for program, if it is non-static it requires an object.
  - There is no object of the class available at the time of starting java runtime, so it is static.
- **Why the main method is public??**
  - Main method of java should not be restricted, because it needs to execute from anywhere.

SIMPLE JAVA PROGRAM

```
import java.lang.*;
class FirstProgram
{
        public static void main(String args[ ])
        {
                System.out.println("HELLO JAVA");
        }
}
```

Output : HELLO JAVA

EXECUTION PROCESS OF JAVA PROGRAM

- After writing a java program we need to save the file with classname containing main method.
- After saving the file, **javac** compiles the program and generates **.class file** if there are no syntax errors.
- javac generates number of .classfiles is equal to number of classes available in java program.
- After that **JVM** load this file into primary memory and class loader subsystem in jvm loads the program then jvm converts the program into system understandable format.
- While Operating system understanding the format, we will get **No Main Method Exist** error if we forget main method in the program.
- If main method exists in the program, then the program starts executing from the main method.
- For compiling, we use the command **javac filename.java** and for executing **java filename**.

BUSSINESS LOGIC : A class does not containing main method is called Bussiness logic.
EXECUTION LOGIC : A class which contains main method is called Execution logic.
In a particular java program, we may have n-1 business logic classes and one execution logic class.

SINGLETON CLASS : A class which contains only one object is called Singleton class.

PRINT STREAM :

- PrintStream class is a predefined class which is responsible to perform print related operations.
- This class contains various kinds of methods like print, println etc....
- Print method is used to print the text on the screen.
- Println method is used to print the text on the screen and cursor moves to the nextline.
  Example :

```
class PrintStream          //Bussiness logic
{
        void method()
        {
                System.out.println("This is println method");
        }
 }
 class ActualMain          //Execution logic
{
        public static void main(String args[])
        {
                PrintStream p=new PrintStream();
                p.method();
        }
}
```

  Output : This is println method

1.10 CONTROL STATEMENTS -For Video

- Control statements are used to control the flow of execution of program.
- These are three types :
  1. Selection/Conditional statements
  2. Loop/Iterative statements
  3. Jump/Flow control statements

1. SELECTION/CONDITIONAL STATEMENTS :

- These are five types as following:
  i. if
  ii. if-else
  iii. nested if
  iv. else-if ladder
  v. switch
- **if :**
  - If we want to define whether the given statement is true or false then we use 'if'.
  - If the condition is true then the block of statements under it will execute.

- **Syntax :**

```
if(condition)
{
        // Block of statements
}
```

- **if else :**
  - If we want to print both positive and negative statements then we use 'if else'.
  - If the condition is true then the block of statements under it will execute and if the condition is false then block of statements under else will execute .
  - **Syntax :**

```
if(condition)
{
        // Executes this block if
        // condition is true
}
else
{
        // Executes this block if
        // condition is false
}
```

- **Nested if :**
  - Nested if statements mean an if statement inside an if statement.
  - **Syntax :**

```
if (condition1)
{
        // Executes when condition1 is true
        if (condition2)
        {
                // Executes when condition2 is true
        }
 }
```

- **if-else-if ladder :**
  - The if statements are executed from the top down.
  - As soon as one of the conditions controlling the if is true, the statement associated with that 'if' is executed, and the rest of the ladder is bypassed.
  - If none of the conditions is true, then the final else statement will be executed.
  - There can be as many as 'else if' blocks associated with one 'if' block but only one 'else' block is allowed with one 'if' block.
  - **Syntax :**

```
if (condition)
{
        // Block of statements
}
else if (condition)
{
        // Block of statements
}
```

```
.
.
.
else
{
        // Block of statements
}
```

- **Switch :**
  - The switch statement is a multiway branch statement.
  - It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
  - **Syntax :**

```
switch (expression)
{
        case value1:
                statement1;
                break;
        case value2:
                statement2;
                break;
        .
        .
        .
        case valueN:
                statementN;
                break;
        default:
                statementDefault;
}
```

2. LOOP/ITERATIVE STATEMENTS :

- These are four types in java :
  i. While
  ii. For
  iii. Do While
  iv. For each

- **while loop :**
  - While loop starts with the checking of Boolean condition. If it evaluated to true, then the loop body statements are executed.
  - It is also called Entry control loop or Pre Test loop.
  - It is indefinite loop.
  - The minimum iterations are 0.
  - **Syntax :**

```
while (boolean condition)
{
        Block of statements
}
```

- **for loop :**
  - Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line.
  - It is Definite loop.
  - **Syntax :**
    for (initialization condition; testing condition;increment/decrement)
    {
        Block of statements
    }

- **do while :**
  - do while loop is similar to while loop with only difference that it checks for condition after executing the statements.
  - It is called as Exit Control Loop or Post Test loop.
  - The minimum iterations are 1.
  - **Syntax :**
    do
    {
        Block of statements
    }
    while (condition);

- **for each loop:**
  - For-each is another array traversing technique like for loop, while loop, do-while loop introduced in Java5.
  - Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
  - In the loop body, you can use the loop variable you created rather than using an indexed array element.
  - **Syntax :**
    for (type var : array)
    {
        statements using var;
    }

3. JUMP STATEMENTS

- **Break :** It is used to terminate the program
- **Continue :** The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- **Return :** It is used to exit from a method, with or without a value.

1.11 ARRAYS -For Video

DEFINITION :

- Collection of similar type of elements/homogeneous elements is called an Array.
- Array can be collection of similar kind of primitive data types and objects.
- In Java, all arrays are dynamically allocated.
- Since arrays are objects in Java, we can find their length using the object property **arrayname.length**

DECLARATION : Data_type array_name[ ];
Example :

```
int array[ ];
char arr[ ];
```

SYNTAX FOR DECLARATION :

```
Datatype ArrayName[ ]
        (or)
Datatype[ ] ArrayName;
        (or)
Datatype [ ]ArrayName;
```

INSTANTIATION :

- When an array is declared, only a reference of an array is created.
- To create or give memory to the array, we need to create an array like this:
  ArrayName = new Datatype [size];

Example :

```
int Array[ ]; //declaring array
Array = new int[10]; // allocating memory to array
        (or)
int[ ] Array = new int[10]; // combining both statements
```

ARRAY LITERAL :

- In a situation where the size of the array and variables of the array are already known, array literals can be used. For example:
  int[ ] Array = new int[ ]{ 1,2,3,4,5,6,7,8,9,10 }; // Declaring array literal

ARRAYS OF OBJECTS :

- An array of objects is created like an array of primitive-type data items in the following way.
  Student[ ] arr = new Student[7]; //student is a user-defined class and there are 7 objects for Student array.

Example program :

```
class Array
{
    public static void main(String[ ] args)
    {
        // declares an Array of integers.
        int[ ] arr;
        // allocating memory for 5 integers.
        arr = new int[5];
        // initialize the first elements of the array
        arr[0] = 10;
```

```
                // initialize the second elements of the array
                arr[1] = 20;
                // so on...
                 arr[2] = 30;
                arr[3] = 40;
                arr[4] = 50;
                // accessing the elements of the specified array
                for (int i = 0; i < arr.length; i++)
                {
                        System.out.println("Element at index " + i + " : " + arr[i]);
                }
            }
        }
```

Output :

```
Element at index 0 : 10
Element at index 1 : 20
Element at index 2 : 30
Element at index 3 : 40
Element at index 4 : 50
```

MULTI DIMENSIONAL ARRAY :

- Multidimensional Arrays can be defined in simple words as array of arrays.
- **Syntax** :
  Datatype[1st dimension][2nd dimension][]..[Nth dimension] ArrayName = new datatype[size1][size2]....[sizeN];
- **Examples :**
  Two dimensional array:
  int[ ][ ] TwoDimArr = new int[10][20];

  Three dimensional array:
  int[ ][ ][ ] ThreeDimArr = new int[10][20][30];

- **Size of Multidimensional arrays :** The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.
  For example : The array int[ ][ ] x = new int[10][20] can store a total of (10*20) = 200 elements. Similarly, array int[ ][ ][ ] y = new int[5][10][20] can store a total of (5*10*20) = 1000 elements.
- **2-DIMENSIONAL ARRAY**
  - **DECLARATION :**
    data_type[ ][ ] array_name = new data_type[x][y];
    For example: int[ ][ ] arr = new int[10][20];
  - **INTIALIZATION :**
    array_name[row_index][column_index] = value;
    For example: arr[0][0] = 5;
    (or)
    data_type[ ][ ] array_name = valueR1C1, valueR1C2, ..., valueR2C1, valueR2C2, ... ;
    For example : int[ ][ ] arr = {{1, 2}, {3, 4}};

– **Example program :**

```
class TwoDim
{
        public static void main(String[ ] args)
        {
                int[ ][ ] arr = { { 1, 2 }, { 3, 4 } };
                for (int i = 0; i < 2; i++)
                {
                        for (int j = 0; j < 2; j++)
                        {
                                System.out.println("arr[" + i + "][" + j + "] = " + arr[i][j]);
                        }
                }
        }
}
```

Output :

```
arr[0][0] = 1
arr[0][1] = 2
arr[1][0] = 3
arr[1][1] = 4
```

JAGGED ARRAY :

- An array with different number of columns is called Jagged Array.
- **Example :**

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

ANONYMOUS ARRAY :

- An Array having no name is called Anonymous array.
- We use this to pass an array in a method.

1.12 TYPE CONVERSION AND TYPE CASTING -For Video

TYPE CASTING/EXPLICIT TYPE CONVERSION :

- In typing casting, a data type is converted into another data type by the programmer using the casting operator.
- Here, the destination data type may be smaller than the source data type when converting the data type to another data type, so it is also called **Narrowing conversion**.
- **Syntax :**
  destination_datatype = (target_datatype)variable;
  () is a casting operator.
- **Example :**

```
int x;
byte y;
y=(byte)x;
```

TYPE CONVERSION/IMPLICIT TYPE CONVERSION:

- In type conversion, a data type is automatically converted into another data type by a compiler at the compiler time.
- Here, the destination data type cannot be smaller than the source data type, so it is also called **Widening conversion**.
- It can only be applied to compatible data types.
- **Example :**
  int x=30;
  float y;
  y=x;

1.13 WRAPPER CLASSES AND SCANNER CLASSES -For Video

WRAPPER CLASSES :

- The wrapper class in Java used to convert primitive datatypes into object and object into primitive.
- These classes contain some predefined methods to convert string into any other datatype.

Fundamental Datatype	Wrapper class	Type Conversion
byte	Byte	public static byte parseByte(String val, int radix);
short	Short	public static short parseShort(String val, int radix);
int	Integer	public static int parseInt(String val, int radix);
long	Long	public static long parseLong(String val, int radix);
float	Float	public static float parseFloat(String val, int radix);
double	Double	public static double parseDouble(String val, int radix);
boolean	Boolean	public static boolean parseBoolean(String val, int radix);

- **Conversion from any datatype into string** : We use **toString** method to convert anytype into string.
- Example:
  Integer.toString(int variablename);
  Float.toString(float variablename);

- Irrespective of using datatype name we can use **valueOf()** method.
- Prototype: public static long valueOf(long b)

SCANNER CLASSES : -For Video

- Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings.
- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.
- To read numerical values of a certain data type XYZ, the function to use is nextXYZ(). For example, to read a value of type short, we can use nextShort()
- To read strings, we use next() and To read line of strings, we use nextLine().

- To read a single character, we use next().charAt(0). next() function returns the next token/word in the input as a string and charAt(0) function returns the first character in that string.
- // Java program to read data of various types using Scanner class.

```
import java.util.Scanner;
class ScannerExample
{
        public static void main(String[ ] args)
        {
                Scanner sc = new Scanner(System.in);
                String name = sc.nextLine();
                char gender = sc.next().charAt(0);
                int age = sc.nextInt();
                long mobileNo = sc.nextLong();
                double cgpa = sc.nextDouble();
                System.out.println("Name: "+name);
                System.out.println("Gender: "+gender);
                System.out.println("Age: "+age);
                System.out.println("Mobile Number: "+mobileNo);
                System.out.println("CGPA: "+cgpa);
        }
}
```

1.14 CONSTRUCTORS

-For Video

Definition :

- Constructor is a special method which is used to place programmer defined values instead of default values when an object is created.
- Whenever we want to do operations immediately once object is created then we use constructors.
- It is similar to a method but follow some rules :

Rules :

- Its Name should similar to classname.
- Constructor won't return any datatype including void also.
- It can support access modifiers except private and does not support non-access modifiers.
- Whenever an object is created then constructor will be called.

TYPES OF CONSTRUCTORS

- There are two types of constructors.
  1. Default/Parameterless constructors
  2. Parameterized Constructor

1. Default/Parameterless constructors :

- If we want multiple objects with same values then we go for Default Constructors.
- It doesn't take parameters.

Example :

```
class Constructor
{
        int a,b;
        float f;
        Constructor()
        {
                a=5;
                b=6;
                f=7.2f;
                System.out.println("a value :"+a);
                System.out.println("b value :"+b);
                System.out.println("f value :"+f);
        }
}
class Construct
{
        public static void main(String args[])
        {
                Constructor c=new Constructor();
                Constructor c1=new Constructor();
        }
}
```

Output :

```
a value :5
b value :6
f value :7.2
a value :5
b value :6
f value :7.2
```

2. Parameterized Constructors :

- If we want multiple objects with different values then we go for parameterized constructors.

Example :

```
class Parameterized
{
        int a,b;
        float f;
        Parameterized()
        {
                System.out.println("Default Constructor");
        }
        Parameterized(int a, int b)
        {
                System.out.println("Parameterized Constructor");
                System.out.println("a value :"+a);
                System.out.println("b value :"+b);
        }
        Parameterized(float f)
        {
                System.out.println("Single Parameterized Constructor");
                System.out.println("f value :"+f);
        }
}
```

```
class ParameterizedConstructor
{
        public static void main(String args[])
        {
                Parameterized p=new Parameterized();
                Parameterized p1=new Parameterized(2,3);
                Parameterized p2=new Parameterized(5.5f);
        }
}
```

Output :

```
Default Constructor
Parameterized Constructor
a value :2
b value :3
Single Parameterized Constructor
f value :5.5
```

1.15 GARBAGE COLLECTOR AND THIS KEYWORD -For Video

GARBAGE COLLECTOR :

- It is a predefined program, which is used to remove unused memory space.
- Wherever we observe unreferenced objects then we can use garbage collector and it is called by JVM automatically.
- Some of the unreferenced objects are nullyfying objects, anonymous objects and assigning one reference to another.
- We can call garbage collector through a method **gc** which is present in **System** class.
- whenever we call **System.gc( )** explicitly, then it calls **finalize** method internally.
- Structure of finalize method is **protected void finalize( );**

  Example :

```
public class GarbageCollection
{
        public static void main(String args[ ])
        {
                GarbageCollection obj=new GarbageCollection();
                obj=null;
                System.gc( );
        }
        public void finalize()
        {
                System.out.println("Object is Garbage collected");
        }
}
```

Output :

```
Object is Garbage collected
```

THIS KEYWORD : -For Video

- It is a kind of reference variable, it refers to current class object or current class methods or constructors.
  Example :

```
class ThisKeyword
{
        int a,b;
        ThisKeyword(int a, int b)
        {
                this.a=a;
                this.b=b;
        }
        void display()
        {
                System.out.println("a value :"+a);
                System.out.println("b value :"+b);
        }
}
class MainMethod
{
        public static void main(String args[ ])
        {
                ThisKeyword t=new ThisKeyword(25,35);
                t.display();
        }
}
```

  Output :

```
a value :25
b value :35
```

1.16 NESTED CLASSES AND INNER CLASSES -For Video

NESTED CLASSES

- Defining a class inside another class is called **Nested class.**
- If a nested class prefix with non-access modifier static then it is called **Static Nested classes**.
- **Structure :**

```
class A
{
        static class B
        {
                //Block of statements
        }
}
```

- Static nested classes support both instance methods and non-instance methods.
- **Example :**

```
class University
{
        int age=19;
        static String branch="cse";
        static class Branch
```

```
    {
        public static void main(String k[ ])
        {
            University u=new University();
            System.out.println("Nested class executed");
            System.out.println("Age :"+u.age);
            System.out.println("Branch :"+branch);
        }
    }
    public static void main(String k[ ])
    {
        System.out.println("Main Method Executed");
    }
}
```

Output :
Nested class Executed
Age :5
Branch :cse

- .classfiles generate after compilating the above example are University.class and University$Branch.class
- Static nested classes are not strongly associated to outer class.

INNER CLASSES **-For Video**

- If a class contains another class which doesnot prefix with any non-access modifier then it is called **Inner class** or **Non-static nested class**.
- It is strongly associated with object of outer class.
- **Ways of writing Inner classes :**
  Case-1 : Accessing Inner class from static area of same class.

```
class InnerStatic
{
    int x=5;
    static int y=10;
    class InnerClass
    {
        void method()
        {
            System.out.println("Method in inner class executed");
            System.out.println("X value :"+x+" Y value"+y);
        }
    }
    public static void main(String args[ ])
    {
        InnerStatic obj1=new InnerStatic();
        InnerStatic.InnerClass obj2=obj1.new InnerClass();        //Referenced object
        obj2.method();
    }
}
```

Output :
Method in inner class executed
X value :5 Y value10

Case-2 : Accessing inner class from instance method of same class

```
class InnerInstance
{
        int x=5;
        static int y=10;
        class InnerClass
        {
                void method()
                {
                        System.out.println("Method in inner class executed");
                        System.out.println("X value :"+x+" Y value"+y);
                }
        }
        void method2()
        {
                InnerClass obj2=new InnerClass();
                obj2.method();
        }
        public static void main(String args[ ])
        {
                InnerInstance obj1=new InnerInstance();
                obj1.method2();
        }
}
```

Output :

```
Method in inner class executed
X value :5 Y value10
```

Case-3 : Accessing innerclass from outside of outerclass.

```
class InnerClass
{
        int x=100;
        static int y=10;
        class Inner
        {
                void p()
                {
                        System.out.println("x value :"+x);
                        System.out.println("y value :"+y);
                }
        }
}
class InnerOutside
{
        public static void main(String args[])
        {
                new InnerClass().new Inner().p();
        }
}
```

Output :

```
x value :100
y value :10
```

2.1 STRINGS AND ITS METHODS -For Video

STRING :

- Generally a collection of characters is called a String.
  Example : ”cse”,”Java”,”DBMS’

c	s	e	\0

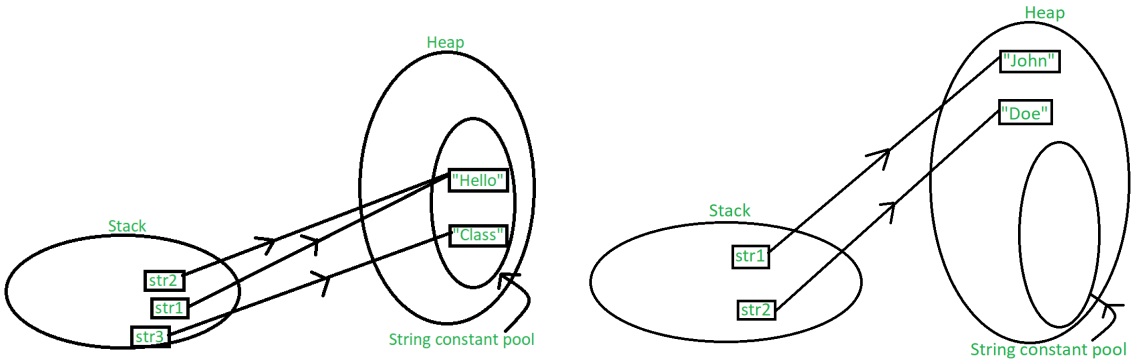
J	a	v	a	\0

D	B	M	S	\0

- In java, every string is a type of object wherever we create a string then memory will be created in **heap memory** for it.
- In java, Every string is an immutable object and modifications are possible through StringBuffer and StringBuilder classes.

WAYS OF CREATING A STRING :

- There are two ways to create a string in Java.
  1. String Literal
  2. Using new Keyword
- **String Literal :** All the constants are allocated in String Constant Pool
  Example : String str1=”Hello”;
  String str2=”Hello”;
  String str3=”class”;
- **Using new keyword** : Defining the strings using constructors.
  Example : String str1=new String(”John”);
  String str2=new String(”Doe”);



STRING COMPARISION :

- We can use ’==’ operator to compare the references of two strings.
- We can use **equals()** method which returns Boolean value to compare content of two strings but it is case-sensitive
- To ignore case-sensitivity we have a method called **equalsIgnoreCase().**
- Example : String s1=”cse”; String s2=”cse”;
  s1==s2 **true**
  s1.equals(s2); **true**

STRING METHODS :

Method	Description	ReturnType
charAt()	Returns the character at the specified index (position)	char
concat()	Appends a string to the end of another string	String
contains()	Checks whether a string contains a sequence of characters	boolean
endsWith()	Checks whether a string ends with the specified character(s)	boolean
equals()	Compares two strings. Returns true if strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
getChars()	Copies characters from a string to an array of chars	void
hashCode()	Returns the hash code of a string	int
indexOf()	Returns position of first found occurrence of specified characters in string	int
isEmpty()	Checks whether a string is empty or not	boolean
lastIndexOf()	Returns position of last found occurrence of specified characters in string	int
length()	Returns the length of a specified string	int
regionMatches()	Tests if two string regions are equal	boolean
replace()	Searches & returns a new string where specified values are replaced	String
split()	Splits a string into an array of substrings	String[]
startsWith()	Checks whether a string starts with specified characters	boolean
substring()	Returns a new string which is the substring of a specified string	String
toCharArray()	Converts this string to a new character array	char
toLowerCase()	Converts a string to lower case letters	String
toString()	Returns the value of a String object	String
toUpperCase()	Converts a string to upper case letters	String
trim()	Removes whitespace from both ends of a string	String
valueOf()	Returns the string representation of the specified value	String

Note : We have lot of methods available in java profile. Just enter **javap java.lang.String** in command prompt.

Sample Program with some methods :

```
class StringMethods
{
        public static void main(String args[])
        {
                String s="cse";
                String s1="hello";
                String s2="cse";
                System.out.println("String constant Pool Checking : "+(s==s1));
                System.out.println("String constant Pool Checking : "+(s==s2));
                String s3=new String("JaiSriRam");
                String s4=new String("JaiSriRam");
                System.out.println("Heap Checking : "+(s3==s4));
                String s5="java programming";
                System.out.println("Second character in cse : "+s.charAt(1));
                char c[]=new char[10];
                s5.getChars(5,12,c,2);
                for(int i=0;i<c.length;i++)
                {
                        System.out.println("Index-"+i+" : "+c[i]);
                }
```

```
        String s6="hello java, hello python, hello C";
        System.out.println("Position of hello from begining in string s6 : "+s6.indexOf("hello"));
        System.out.println("Position of hello from begining in string s6 after 0 : "+s6.indexOf("hello",1));
        System.out.println("Position of hello from last in string s6 : "+s6.lastIndexOf("hello"));
        System.out.println("substring of string s6[from 12 to 24] : "+s6.substring(12,24));
        System.out.println("Length of the string s6 : "+s6.length());
        System.out.println("Concatenation to string s1 : "+s.concat(" branch"));
        System.out.println("Region Matches : "+s1.regionMatches(true,0,s6,12,5));
        System.out.println("Replace['C' with 'c' in string s6] : "+s6.replace('C','c'));
        System.out.println("Startswith : "+s5.startsWith("java"));
        System.out.println("endswith : "+s5.endsWith("ing"));
        String s7=new String(" StringTrimMethod ");
        System.out.println("Before using Trim Method : "+s7);
        System.out.println("After using Trim Method : "+s7.trim());
    }
}
```

Output :

```
String constant Pool Checking : false
String constant Pool Checking : true
Heap Checking : false
Second character in cse : s
Index-0 :
Index-1 :
Index-2 : p
Index-3 : r
Index-4 : o
Index-5 : g
Index-6 : r
Index-7 : a
Index-8 : m
Index-9 :
Position of hello from begining in string s6 : 0
Position of hello from begining in string s6 after 0 : 12
Position of hello from last in string s6 : 26
substring of string s6[from 12 to 24] : hello python
Length of the string s6 : 33
Concatenation to string s1 : cse branch
Region Matches : true
Replace['C' with 'c' in string s6] : hello java, hello python, hello c
Startswith : true
endswith : true
Before using Trim Method :   StringTrimMethod
After using Trim Method   : StringTrimMethod
```

Why strings are immutable in java??

- Because java uses the concept of string literal. Suppose there are 5 reference variables, all refer to one object like "India". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why string objects are immutable in java.

2.2 STRING-BUFFER -For Video

StringBuffer :

- StringBuffer is a class in Java that represents a mutable sequence of characters.
- It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time.
- The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

Some important features and methods of StringBuffer class :

1. StringBuffer objects are mutable, meaning that you can change the contents of the buffer without creating a new object.
2. By default, it is having 16 characters of memory space, so no need to reallocate the memory.
3. The initial capacity of a StringBuffer can be specified when it is created, or it can be set later with the **ensureCapacity()** method.
4. The **append()** method is used to add characters, strings, or other objects to the end of the buffer.
5. The **insert()** method is used to insert characters, strings, or other objects at a specified position in the buffer.
6. The **delete()** method is used to remove characters from the buffer.
7. The **reverse()** method is used to reverse the order of the characters in the buffer.
8. The **equals()** method does not compare content but checks references of buffers.

Note :

- If you need to perform multiple modifications to a string, or if you need to access a string from multiple threads, using StringBuffer can be more efficient and safer than using regular String objects.

Important Constructors of StringBuffer class :

1. **StringBuffer()** : creates an empty string buffer with an initial capacity of 16.
2. **StringBuffer(String str)** : creates a string buffer with the specified string.
3. **StringBuffer(int capacity)** : creates an empty string buffer with the specified capacity as length.

Sample Program :

```
class StringBufferPractice
{
        public static void main(String args[])
        {
                StringBuffer sb=new StringBuffer("java");
                StringBuffer sb1=new StringBuffer("java");
                System.out.println(sb.length());
                System.out.println( sb.capacity());
                System.out.println(sb==sb1);
                System.out.println("Content : "+sb.equals(sb1));
                System.out.println("Content :"+sb.toString().equals(sb1.toString()));
                sb.insert(2,"Like");
                System.out.println("Insert : "+sb);
                sb.delete(2,6);
```

```
            System.out.println("Delete : "+sb);
            sb.replace(2,3,"y");
            System.out.println("Replace : "+sb);
            sb1.reverse();
            System.out.println("Reverse : "+sb1);
            sb.replace(2,3,"v");
            System.out.println("Replace : "+sb);
            sb.append("Sir");
            System.out.println("Append : "+sb);
            System.out.println("Index : "+sb.indexOf("Sir"));
            sb.append("java");
            System.out.println("Append : "+sb);
            System.out.println("Last Index : "+sb.lastIndexOf("java"));
        }
}
```

Output :

```
4
20
false
Content : false
Content : true
Insert : jaLikeva
Delete : java
Replace : jayaReverse : avaj
Replace : java
Append : javaSirIndex : 4
Append : javaSirjava
Last Index : 7
```

2.3 STRING-BUILDER -For Video

StringBuilder :

- StringBuilder in Java represents a mutable sequence of characters.
- Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternative to String Class.
- The function of StringBuilder is very much similar to the StringBuffer class, as both of them provide an alternative to String Class by making a mutable sequence of characters.
- However, the StringBuilder class differs from the StringBuffer class on the basis of synchronization.
- The StringBuilder class provides no guarantee of synchronization whereas the StringBuffer class does.
- StringBuilder is not thread-safe, so it should not be used in a multi-threaded environment.

Constructors of StringBuilder class :

1. **StringBuilder()** : Constructs a string builder with no characters in it and an initial capacity of 16 characters.
2. **StringBuilder(int capacity)** : Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.

3. **StringBuilder(CharSequence seq)** : Constructs a string builder that contains the same characters as the specified CharSequence.
4. **StringBuilder(String str)** : Constructs a string builder initialized to the contents of the specified string.

Conversion between types of strings in Java :

- Sometimes there is a need for converting a string object of different classes like String, StringBuffer, StringBuilder to one another.
  1. From String to StringBuffer and StringBuilder
  2. From StringBuffer and StringBuilder to String
  3. From StringBuffer to StringBuilder or vice-versa

Case-1 : From String to StringBuffer and StringBuilder

- We can directly pass the String class object to StringBuffer and StringBuilder class constructors.
- As the String class is immutable in java, so for editing a string, we can perform the same by converting it to StringBuffer or StringBuilder class objects.
- **Example :**

```
class StringToBufferAndBuilder
{
        public static void main(String[] args)
        {
                String str = "Computer";
                StringBuffer sbr = new StringBuffer(str);
                sbr.reverse();
                System.out.println(sbr);
                StringBuilder sbl = new StringBuilder(str);
                sbl.append("ScienceEngineering");
                System.out.println(sbl);
        }
}
```

Output :

```
retupmoC
ComputerScienceEngineering
```

Case 2 : From StringBuffer and StringBuilder to String

- This conversion can be performed using **toString()** method which is overridden in both StringBuffer and StringBuilder classes.
- **Example :**

```
class BufferAndBuilderToTheString
{
        public static void main(String[] args)
        {
                StringBuffer sbr = new StringBuffer("Java");
                StringBuilder sbdr = new StringBuilder("Python");
                String str = sbr.toString();
                System.out.println("StringBuffer object to String : "+str);
                String str1 = sbdr.toString();
```

```
            System.out.println("StringBuilder object to String : "+str1);
            sbr.append("Programming");
            System.out.println(sbr);
            System.out.println(str);
        }
}
```

Case 3 : From StringBuffer to StringBuilder or vice-versa

- In this case, We first convert the StringBuffer/StringBuilder object to String using **toString()** method and then from String to StringBuilder/StringBuffer using constructors.
- **Example :**

```
class Example3
{
        public static void main(String[] args)
        {
            StringBuffer sbr = new StringBuffer("RGUKT");
            String str = sbr.toString();
            StringBuilder sbl = new StringBuilder(str);
            System.out.println(sbl);
        }
}
```

Output : RGUKT

DIFFERENCE BETWEEN STRINGS, STRINGBUFFER AND STRINGBUILDER :

Strings	StringBuffer	StringBuilder
Immutable	Mutable	Mutable
Content is fixed	Content changes frequently	Content changes frequently
-	StringBuffer is synchronized.	StringBuilder is asynchronized.
-	It is thread safe	It is not thread safe
-	Multiple threads can't call methods at a time.	Multiple threads can call methods at a time
-	StringBuffer is slower than StringBuilder.	StringBuilder is faster than StringBuffer.

2.4 STRING TOKENIZER -For Video

StringTokenizer class :

- It is used to break a string into tokens.
- It takes input as a large string and converts into small parts by using given **delimeters**.
- To use String Tokenizer class we have to specify an input string and a string that contains delimiters.
- Delimiters are the characters that separate tokens. Each character in the delimiter string is considered as a valid delimiter.
- Default delimiters are whitespaces, new line, space, and tab.

Methods Of StringTokenizer Class :

Method	ActionPerformed
countTokens()	Returns the total number of tokens present
hasMoreToken()	Tests if tokens are present for the StringTokenizer's string
nextElement()	Returns an Object rather than String
hasMoreElements()	Returns the same value as hasMoreToken
nextToken()	Returns the next token from the given StringTokenizer.

Program :

```
import java.util.StringTokenizer;
class StringTokenizer
{
        public static void main(String args[])
        {
                StringTokenizer st1 = new StringTokenizer("I Like Java");
                System.out.println("No of tokens : "+st1.countTokens());
                while (st1.hasMoreTokens())
                {
                        System.out.println(st1.nextToken());
                }
                StringTokenizer st2 = new StringTokenizer("I:Like:RGUKT", ":");
                System.out.println("No of tokens : "+st2.countTokens());
                while (st2.hasMoreTokens())
                {
                        System.out.println(st2.nextToken());
                }
                StringTokenizer st3 = new StringTokenizer("I:Like:RGUKT", " :", true);
                System.out.println("No of tokens : "+st3.countTokens());
                while (st3.hasMoreTokens())
                {
                        System.out.println(st3.nextToken());
                }
        }
}
```

Output :

```
No of tokens : 3
I
Like
Java
No of tokens : 3
I
Like
RGUKT
No of tokens : 5
I
:
Like
:
RGUKT
```

2.5 RANDOM CLASSES -For Video

- Random class is used to generate random values.
- This class provides various method calls to generate different random data types such as float, double, int.
- It is present in java.util.* package. We can simply import it as **java.util.Random**.
  Examples :
  Random r=new Random();
  r.nextBoolean(); //generates either true or false
  r.nextDouble(); //generates a value from 0.0 to 1.0
  r.nextFloat(); //generates a value from 0.0f to 1.0f
  r.nextInt(int); //generates a seed value
  r.nextInt(10); //generates a value from 0 to 10

- **Program :**

```
import java.util.Random;
class RandomPractice
{
        public static void main(String args[])
        {
                Random r=new Random();
                System.out.println("Boolean value : "+r.nextBoolean());
                System.out.println("Double value : "+r.nextDouble());
                System.out.println("Float value : "+r.nextFloat());
                System.out.println("Integer value : "+r.nextInt());
                System.out.println("Integer Seek value : "+r.nextInt(10));
        }
}
```

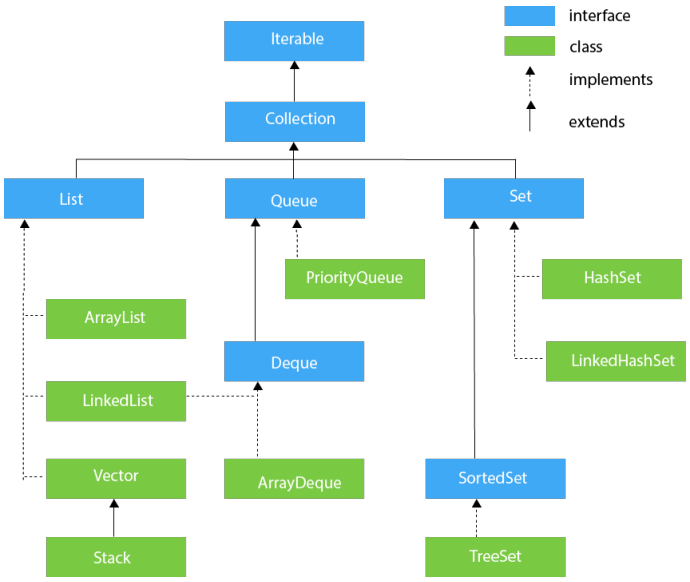
Output :
Boolean value : true
Double value : 0.5871664409028182
Float value : 0.8111359
Integer value : 1572686262
Integer Seek value : 5
*****All values are generated randomly*****

2.6 ARRAY LIST

- ArrayList is a dynamic array and we do not have to specify the size while creating it, the size of the array automatically increases when we dynamically add and remove items.
- ArrayList is a class implemented using the List interface.

Collection Frame :

- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.
- Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).



Important Points :

1. ArrayList inherits AbstractList class and implements the List interface.
2. Java ArrayList class can contain duplicate elements.
3. ArrayList class maintains insertion order.
4. ArrayList class is non synchronized.
5. ArrayList allows random access because array works at the index basis.
6. In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.
7. ArrayList allows us to randomly access the list.
8. ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases.

Constructors :

- **ArrayList() :** This constructor is used to build an empty array list.
- **ArrayList(Collection c) :** This constructor is used to build an array list initialized with the elements from collection c.
- **ArrayList(int capacity) :** This constructor is used to build an array list with initial capacity being specified.

Some Methods of ArrayList :

1. **add(int index, Object element) :** This method is used to insert a specific element at a specific position index in a list.
2. **add(Object o) :** This method is used to append a specific element to the end of a list.
3. **addAll(Collection C) :** This method is used to append all the elements from a specific collection to the end of the mentioned list, in such an order that the values are returned by the specified collection's iterator.

4. **addAll(int index, Collection C) :** Used to insert all of the elements starting at the specified position from a specific collection into the mentioned list.
5. **clear() :** This method is used to remove all the elements from any list.
6. **clone() :** This method is used to return a shallow copy of an ArrayList in Java.
7. **indexOf(Object O) :** The index the first occurrence of a specific element is either returned or -1 in case the element is not in the list.
8. **isEmpty() :** Returns true if this list contains no elements.
9. **lastIndexOf(Object O) :** The index of the last occurrence of a specific element is either returned or -1 in case the element is not in the list.
10. **trimToSize() :** This method is used to trim the capacity of the instance of the ArrayList to the list's current size.

2.7 LINKED LIST

- Linked List are linear data structures where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part.
- The elements are linked using pointers and addresses. Each element is known as a node.
- Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays.
- It also has few disadvantages like the nodes cannot be accessed directly instead we need to start from the head and follow through the link to reach to a node we wish to access.
- LinkedList class, manipulation is fast because no shifting needs to be occurred.

Constructors :

- **LinkedList() :** Used to create an empty linked list.
- **LinkedList(Collection C) :** Used to create a ordered list which contains all the elements of a specified collection, as returned by the collection's iterator.

Some Methods of Linked list:

1. **size() :** It returns the number of elements in this list.
2. **clear() :** It removes all of the elements from this list.
3. **clone() :** It returns a shallow copy of this LinkedList.
4. **contains(Object o) :** It returns true if this list contains the specified element.
5. **add(E e) :** This method Appends the specified element to the end of this list.
6. **add(int index, E element) :** This method Inserts the specified element at the specified position in this list.
7. **get(int index) :** This method returns the element at the specified position in this list.
8. **getFirst() :** This method returns the first element in this list.
9. **getLast() :** This method returns the last element in this list.
10. **indexOf(Object o) :** It returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
11. **lastIndexOf(Object o) :** This method returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
12. **remove() :** It retrieves and removes the head (first element) of this list.

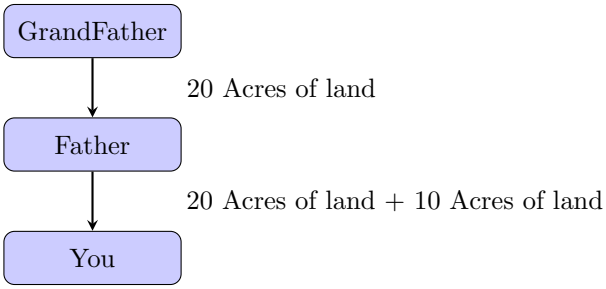
UNIT-3 OBJECT ORIENTED PRINCIPLES

3.1 INHERITANCE AND ITS TYPES

-For Video

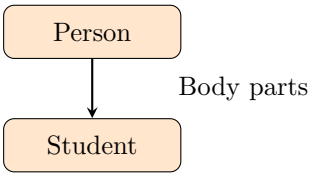
Inheritance : Generally, Inheritance means the process of acquiring properties from one thing to another.

Example-1 :



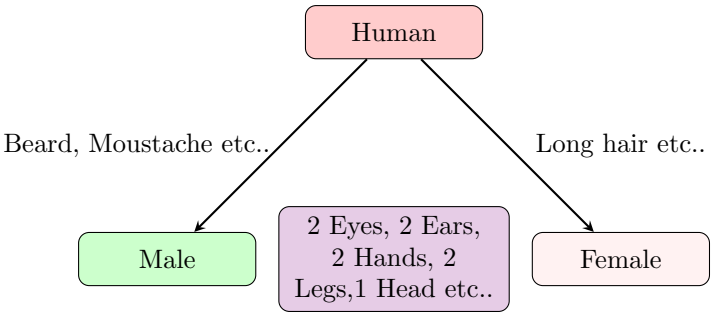
- In the above example, Grandfather gave 20 acres to father.
- Father with his own effort earned another 10 acres of land.
- Totally 30 acres of land is given to you by your Father.

Example-2 :



- A student contains another additional works like Homeworks and Exams.
- Every student is a person.
- But every person need not be a student.

Example-3 :



- Every human including male and female contains body parts like eyes , nose ,ears, hands ,legs etc..
- But male contains extra things like beard, moustache etc.
- Female also having additional things like long hair etc.

Note Points :

- From all the above examples we can conclude that Inheritance is the process of getting features.
- As said in example-2, we can't inherit in reverse manner.
- After inheritance, along with inherited features, we can have additional features also.

INHERITANCE

- The process of getting features[Data members and Methods] from one class to another class is called Inheritance.
- Inheritance is an important pillar of OOP(Object-Oriented Programming).
- Inheritance is also called as **Reusability** or **Derivation** or **Subclassing** or **Extendable classes**.
- The data members and methods from parent class are available logically in child class. But they won't occupy any amount of memory space.

TERMINOLOGY IN INHERITANCE

1. Parent/Super/Base class :

- The class which is giving features to some other class is called Parent/Super/Base class.

2. Child/Sub/Derived class :

- The class which is taking features from some other class is called Child/Sub/Derived class.
- It contains features of parent class as well as features of its own class.
- Thus it contains more features than base class.

NEED OF INHERITANCE :

- Wherever we want to reuse the components(Data members and methods) then we go for inheritance.

ADVANTAGES OF INHERITANCE :

1. Application development time is less.
2. Memory space is also taking less.
3. Execution time is also less.
4. Perfomance will be improved.

SYNTAX OF INHERITANCE :

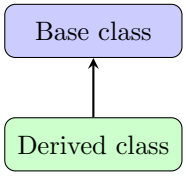
- The **extends** keyword is used for inheritance in Java.
- Using the extends keyword indicates you are derived from an existing class.
  class <classname> extends <baseclassname>
  {
  data members and methods;
  }

TYPES OF INHERITANCE :

- There are five types of inheritances.
  1. Single Inheritance
  2. Multilevel Inheritance
  3. Hierarchical Inheritance
  4. Multiple Inheritance
  5. Hybrid Inheritance

1. SINGLE INHERITANCE :

- In single inheritance, there exists only one base class and one derived class.



- **Example Program :**

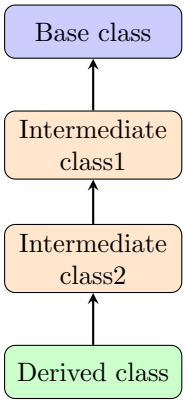
```
class BC
{
        int a=5;
        void method1()
        {
                System.out.println("Method1 from Base class");
        }
}
class DC extends BC
{
        int b=6;
        void method2()
        {
                System.out.println("Method2 from Derived class");
        }
}
class SingleInheritance
{
        public static void main(String args[ ])
        {
                DC dc=new DC();
                System.out.println("a value in base class : "+dc.a);
                dc.method1();
                System.out.println("b value in Derived class : "+dc.b);
                dc.method2();
        }
}
```

Output :

```
a value in base class : 5
Method1 from Base class
b value in Derived class : 6
Method2 from Derived class
```

2. MULTILEVEL INHERITANCE :

- In multilevel inheritance, there exists one base class, one derived class and multiple intermediate base classes.



- **Example :**

```
class BC
{
        int a=5;
        void method1()
        {
                System.out.println("Method1 from Base class");
        }
}
class IBC extends BC
{
        int b=6;
        void method2()
        {
                System.out.println("Method2 from Intermediate Base class");
        }
}
class DC extends IBC
{
        int c=7;
        void method3()
        {
                System.out.println("Method3 from Derived class");
        }
}
class MultiLevelInheritance
{
        public static void main(String args[ ])
        {
                DC dc=new DC();
                System.out.println("a value : "+dc.a);
                dc.method1();
                System.out.println("b value : "+dc.b);
```

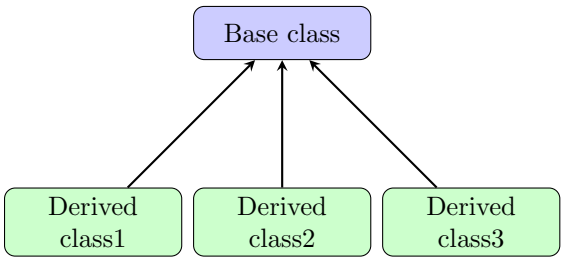
```
        dc.method2();
        System.out.println("c value : "+dc.c);
        dc.method3();
    }
}
```

Output :

a value : 5
Method1 from Base class
b value : 6
Method2 from Intermediate Base class
c value : 7
Method3 from Derived class

3. HIERARCHIAL INHERITANCE :

- In Hierarchical Inheritance, there exists one base class and multiple derived classes.



- **Example :**

```
class BC
{
    int a=5;
    void method1()
    {
        System.out.println("Method1 from Base class");
    }
}
class DC1 extends BC
{
    int b=6;
    void method2()
    {
        System.out.println("Method2 from Derived class1");
    }
}
class DC2 extends BC
{
    int c=7;
    void method3()
    {
        System.out.println("Method3 from Derived class2");
    }
}
class HierarchiealInheritance
```

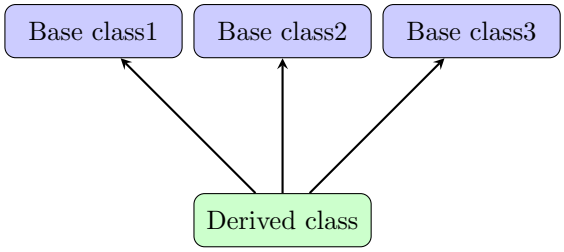
```
{
        public static void main(String args[ ])
        {
                DC1 dc=new DC1();
                dc.method1();
                System.out.println("b value : "+dc.b);
                DC2 d=new DC2();
                System.out.println("c value : "+dc.a);
                d.method3();
        }
}
```

Output :

Method1 from Base class
b value : 6
c value : 5
Method3 from Derived class2

4. MULTIPLE INHERITANCE :

- In multiple inheritance, there exists multiple base classes and one derived class.



- In java, multiple inheritance **doesnot supports through classes** but supports through interfaces.
- **Example :**

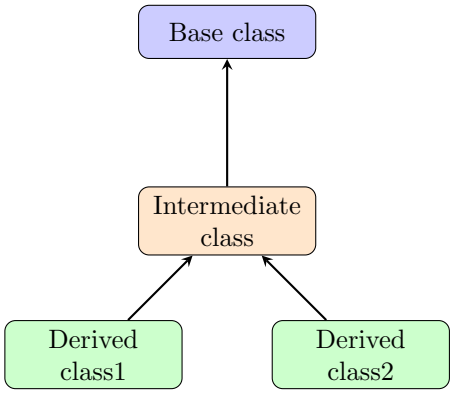
```
class A
{
        int a=10;
}
class B
{
        int a=11;
}
class C extends A, B
{
        void m()
        {
                System.out.println("a value : "+a);
        }
}
class MultipleInheritance
{
        public static void main(String args[ ])
        {
```

```
        C obj=new C();
        obj.m();
    }
}
```

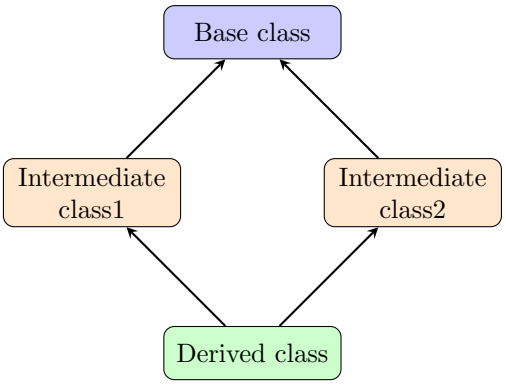
Output : error

5. HYBRID INHERITANCE :

- It is a combination of all available inheritances.
- If it contains multiple inheritance then it does not supports in java.



Valid



Invalid[Diamond Problem]

Example :

```
class BC
{
    int a=1;
}
class IBC extends BC
{
    int b=2;
}
class DC1 extends IBC
{
    int c=3;
}
class DC2 extends IBC
{
    int d=4;
}
class HybridInheritance
{
    public static void main(String args[ ])
    {
        DC1 dc=new DC1();
        System.out.println("b value : "+dc.b);
        DC2 d=new DC2();
        System.out.println("a value : "+dc.a);
    }
}
```

Output :
b value : 2
a value : 1

Note Points :

1. It is recommended to create object for **BottomMostClass** which contains all features.
2. A superclass can have any number of subclasses. But a subclass can have only one superclass.
3. If we don't want inherit some features from base class then make them **private**.
4. Whenever we **override** logical thing then it will give priority for local variables and local methods.
5. **Constructors** doesnot participate in inheritance process, because it is violating the naming principle.
6. All **static** methods and variables are not participate in inheritance process.
7. All **final** classes also not participate in inheritance process.
8. For every class, the topmost super class is **Object class** and every class is implicitly a subclass of the Object class. Thus by default, every class participates in inheritance process.

3.2 TYPES OF RELATIONSHIPS IN JAVA -For Video

- In java, there are 3 types of relationships.
  1. is-A Relation
  2. has-A Relation
  3. Uses-A Relation

1. is-A Relation :

- Whenever we inherits one class features to another class then it is said to be in is-A relation.
- We know that every class participates in inheritance process with Object class then they have is-A relation.
- Final classes doesnot have is-A relation because they don't participate in inheritance.
- Here memory will be created logically.

2. has-A Relation :

- When object of one class is created inside another class then it is in has-A relation.
- In has-A relation memory will be created physically.
  Example : PrintStream class
- The advantage of has-A relation is it can eliminates Diamond problem.

3. Uses-A Relation :

- When object of one class is created inside a method of another class then it is in Uses-A relation.
- In Uses-A relation also memory will be created physically.
  Example : Business class and execution classes.
- Generally we creates object of a business class in main method of execution class. So it is an example of Uses-A relation.

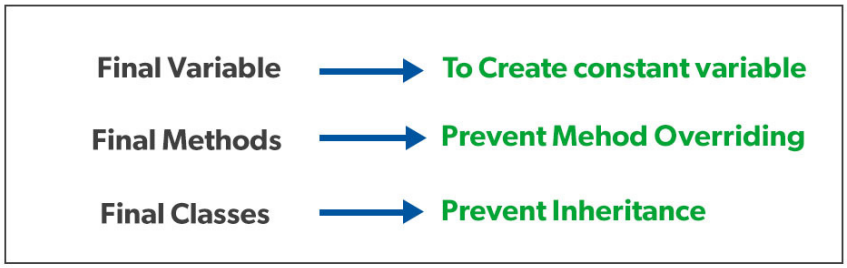
3.3 FINAL KEYWORD AND USAGE OF FINAL AT VARIOUS LEVELS -For Video

CONSTANTS IN JAVA :

- Generally, constant is a variable whose value cannot be changed.
- We use final keyword to make a constant.

FINAL :

- final is a non-access modifier applicable only to a variable, a method, or a class.



FINAL AT DIFFERENT LEVELS :

1. At Variable level
2. At Method level
3. At Class level

1. final at variable level :

- We can't modify a variable after made it as a final variable or constant.
- Declaration : final int a ;
- **Initialization :** final int a=2;
- **Note :** If we make final at declaration time then it allows to change the value for one time only.
  Example :
  final int a;
  a=2
  a=3 //error

2. final at method level :

- Once we made a method as final, then we can't override this method.
- We can use a final method without modifying it.

3. final at class level :

- If we don't want to override features of base class then make it as final[restriction to entire class].
- Final classes are not participate in inheritance process.

- **Example :**

```
final class C1
{
        int a=5;
        private int b=6;
        final void display()
        {
                System.out.println("a and b values : "+a+","+b);
        }
}
class C2 extends C1          //cannot inherit from final C1
{
        void display()              //display() in C2 cannot override display() in C1
        {
                System.out.println("Over riding");
        }
}
class FinalKeyword
{
        public static void main(String args[ ])
        {
                C2 obj=new C2();
                obj.display();
        }
}
```

3.4 STATIC AND DYNAMIC BINDING, OBJECT TYPE-CASTING

Binding : -For Video

- The Association between two things is called Binding.
- It is a relation between called method and calling method.

Types of Binding :

1. Static Binding.
2. Dynamic Binding.

1. Static Binding :

- It is a relation between calling method and called method during compile time.
- In static binding, compiler will decide which method it has to call during compile time.
- Compiler knows that final methods cannot be override.
- Through private and static also methods cannot be override.

2. Dynamic Binding : -For Video

- If compiler decides which method it has to call, at runtime then it is called Dynamic Binding.
- It is a relation between calling method and called method at run time.
  Example : Method Overriding
- Here compiler doesnot decide which method to be called at compile time.

Example program :

```
class Bc
{
        final void m1()
        {
                System.out.println("m1 final method called...");
        }
        static void m2()
        {
                System.out.println("m2 static method called...");
        }
        private void m3()
        {
                System.out.println("m3 private method called...");
        }
}
class Dc extends Bc
{
        void m4()
        {
                System.out.println("m4 normal method called...");
        }
}
class Binding
{
        public static void main(String args[ ])
        {
                Dc d=new Dc();
                d.m1();         //Static Binding
                m2();          //Static Binding
                d.m3();         //Static Binding
                d.m4();         //Dynamic Binding
        }
}
```

OBJECT TYPE-CASTING **-For Video**

1. **Normal object type-casting :**
   - It is assigning same class object to same class another object.
2. **Implicit object type-casting :**
   - Assigning sub class object to super class object is called implicit object type-casting.
   - Here there is no loss of the data.
3. **Explicit object type-casting :**
   - It is assigning super class object to sub class object.
   - Here there is a chance of loss of data.

Example :

```
class A
{
        void m()
        {
                System.out.println("m method in class A");
        }
        void m1()
        {
                System.out.println("m1 method in class A");
        }
}
class B extends A
{
        void m1()
        {
                System.out.println("m1 method in class B");
        }
}
class ObjectTypeCast
{
        public static void main(String args[])
        {
                A a=new A();
                A a1=new A();
                B b=new B();
                a=a1;          //object type casting
                a.m1();
                a=b;          //implicit type casting
                a.m1();
                //b=a;          //explicit and data loss
                b=(B)a;          //explicit to avoid data loss
                b.m();
        }
}
```

Output :

```
m1 method in class A
m1 method in class B
m method in class A
```

3.5 SUPER KEYWORD AND IT'S USAGE -For Video

Super Keyword :

- The super keyword in java is a reference variable that is used to refer to parent class objects.
- Whenever we are inheriting features of base class to derived class there is a possibility that base class and derived class features are same then compiler gets ambiguity.
- To overcome this we use Super keyword.

Usage of super keyword at various levels :

1. At Variable level.
2. At Method level.
3. At Constructor level.

1. Super at Variable level

- It occurs when a derived class and base class has the same data members. In that case, there is a possibility of ambiguity for the JVM.
- **Example :**

```
class A
{
        int i;
}
class B extends A
{
        int i;
        void assign(int x, int y)
        {
                this.i=x;
                super.i=y;
        }
        void display()
        {
                System.out.println("i value in Super class : "+super.i);
                System.out.println("i value in Sub class : "+this.i);
        }
}
class SuperAtVariableLevel
{
        public static void main(String args[ ])
        {
                B b=new B();
                b.assign(5,6);
                b.display();
        }
}
```

Output :

```
i value in Super class : 6
i value in Sub class : 5
```

2. Super at Method Level :

- This is used when we want to call the parent class method.
- So whenever a parent and child class have the same-named methods then to resolve ambiguity we use the super keyword.
- It is not applicable for multilevel inheritance.

- **Example :**

```
class A
{
        void display()
        {
                System.out.println("Super class called.....");
        }
}
class B extends A
{
        void display()
        {
                super.display();
                System.out.println("Sub class called.....");
        }
}
class SuperAtMethodLevel
{
        public static void main(String k[])
        {
                B b=new B();
                b.display();
        }
}
```

Output :

```
Super class called.....
Sub class called.....
```

Super at Constructor Level :

- In order to establish communication between the class constructors then we use super keyword at constructor level.
- super can call both parametric as well as non-parametric constructors depending upon the situation.
- Call to super() must be the first statement in the Derived Class constructor.
- super should not use for static methods.
- **Example :**

```
class Bc
{
        Bc()
        {
                System.out.println("Bc constructor called.....");
        }
        Bc(int a, int b)
        {
                System.out.println("Bc parameterized constructor called.....");
        }
}
```

```
class Dc extends Bc
{
        Dc()
        {
                super();
                System.out.println("Dc constructor called.....");
        }
        Dc(int a,int b)
        {
                super(5,4);
                System.out.println("Dc parameterized constructor called.....");
        }
}
class SuperAtConstructorLevel
{
        public static void main(String k[ ])
        {
                Bc b=new Bc();
                Bc b1=new Bc(2,3);
        }
}
```

Output :

Bc constructor called.....
Bc parameterized constructor called.....

3.6 POLYMORPHISM

-For Video

- Poly means many and morphism means forms.
- We can define polymorphism as the ability of a message to be displayed in more than one form.

Reallife Illustration :

-> A Boy behave like a Student in a School

-> A Boy behave like a Customer in Market or Shopping Mall

-> A Boy behave like a Passenger in a Bus

-> A Boy behave like a Son in Home

- A person possesses different behavior in different situations. This is called polymorphism.

POLYMORPHISM IN JAVA

- Polymorphism is considered one of the important features of Object-Oriented Programming.
- Polymorphism allows us to perform a single action in different ways.

TYPES OF POLYMORPHISM

- Based on type of methods we are using, polymorphism is classified as
  1. Static Polymorphism.
  2. Dynamic Polymorphism.

1. Static Polymorphism :

- Here compiler decides which method should be invoked.
- This type of polymorphism is achieved by **method overloading.**
- It is also called as **Compile time Polymorphism** or **Static Method Dispatch.**
- Example

```
class A
{
        static int Multiply(int a, int b)
        {
                return a * b;
        }
        static double Multiply(double a, double b)
        {
                return a * b;
        }
}
class StaticMethodDispatch
{
        public static void main(String[ ] args)
        {
                System.out.println(A.Multiply(3,5));
                System.out.println(A.Multiply(10.5,5.3));
        }
}
```

Output :

```
15
55.65
```

2. Dynamic Polymorphism :

- It is a process in which a call to an overridden method is decided at runtime.
- This type of polymorphism is achieved by **Method Overriding.**
- It is also called as **Runtime Polymorphism** or **Dynamic Method Dispatch.**
- If we call an overriden method under super class reference then sub class method will be called and it will be decided at run time.

- **Example :**

```
class SuperClass
{
        void Hello()
        {
                System.out.println("Hello method called from SuperClass");
        }
}
class SubClass1 extends SuperClass
{
        void Hello()
        {
                System.out.println("Hello method called from SubClass1");
        }
}
class SubClass2 extends SuperClass
{
        void Hello()
        {
                System.out.println("Hello method called from SubClass2");
        }
}
class DynamicMethodDispatch
{
        public static void main(String args[ ])
        {
                SuperClass s;
                s=new SubClass1();
                s.Hello();
                s=new SubClass2();
                s.Hello();
        }
}
```

Output :

```
Hello method called from SubClass1
Hello method called from SubClass2
```

Advantages of Polymorphism

1. Increases code reusability by allowing objects of different classes to be treated as objects of a common class.
2. Improves readability and maintainability of code by reducing the amount of code that needs to be written and maintained.
3. Supports dynamic binding, enabling the correct method to be called at runtime, based on the actual class of the object.
4. Enables objects to be treated as a single type, making it easier to write generic code that can handle objects of different types.

Differences between compile-time polymorphism and run-time polymorphism :

Compile Time Polymorphism	Run time Polymorphism
Here, the call is resolved by the compiler.	Here, the call is not resolved by the compiler.
It is also known as Static binding, Early binding.	It is also known as Dynamic binding, Late binding
It provides fast execution.	It provides slow execution
It is less flexible as all things execute at compile time.	It is more flexible as all things execute at run time.
Inheritance is not involved.	Inheritance is involved.

3.7 ABSTRACTION, ABSTRACT METHODS AND CLASSES.

ABSTRACTION :

- Abstraction is one of the pillars of Object Oriented Programming.
- The process of identifying only the required characteristics of an object ignoring the irrelevant details is called **Abstraction.**.
- Simply, it is a process of hiding implementation details or unnecessary details or internal details.
- **Real-Life Example :**

A man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

ABSTRACT METHODS :

- An abstract method is a method that does not have an implementation.
- It is declared using the abstract keyword and ends with a semicolon instead of a method body.
- Whenever we want a method without body then we use abstract methods.
- Abstract method is common to all and can be overridden.
- **Syntax : abstract returntype method-name(parameter-list);**
- A method-defined abstract must always be redefined in the subclass, thus making overriding compulsory or making the subclass itself abstract.
- The normal methods are called as **Concrete methods.**
- Any concrete class(i.e. class without abstract keyword) that extends an abstract class must override all the abstract methods of the class.

ABSTRACT CLASSES : **-For Video**

- An abstract class is a class that is declared with an abstract keyword.
- It is a class that can not be initiated by itself, it needs to be subclassed by another class to use its properties.

- **Important points :**
  1. An instance(object) of an abstract class can not be created.
  2. Constructors are allowed. An abstract class can have parameterized constructors and the default constructor is always present in an abstract class.
  3. If a class contains at least one abstract method then compulsory should declare that as abstract.
  4. We can have an abstract class without any abstract method.
  5. An abstract class may or may not have all abstract methods. Some of them can be concrete methods.
  6. Every abstract class must have to participate in inheritance.

- **Example :**

```
abstract class Animal
{
        abstract void sound();
}
class Dog extends Animal
{
        void sound()
        {
                System.out.println("Dog Barks");
        }
}
class Cat extends Animal
{
        void sound()
        {
                System.out.println("Cat sounds like Meow...");
        }
}
class AbstractClass
{
        public static void main(String args[])
        {
                Animal a;
                a=new Dog();
                a.sound();
                a=new Cat();
                a.sound();
        }
}
```

Output :

```
Dog Barks
Cat sounds like Meow...
```

Advantages of Abstract classes :

1. It will reduce the complexity.
2. It avoids code duplicates.
3. Helps to improve the security of apps.
4. Modularity and Reusability.

3.8 ENCAPSULATION

DATA HIDING :

- It is a way of restricting the access of data members by hiding the implementation details and by using access modifier **private**.

ENCAPSULATION :

- Encapsulation is a fundamental concept in object-oriented programming.
- In java, the wrapping up of similar kind of data is called as Encapsulation.
- Encapsulation is a way of hiding the implementation details of a class from outside access and only displaying a public interface that can be used to interact with the class.
- **Data Hiding + Abstraction = Encapsulation**.
- Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.
- It is more defined with the setter and getter method.
- **Example :**

```
class Age
{
        private int age;
        public void setAge(int age) {
                this.age = age;
        }
        public int getAge() {
                return age;
        }
}
class Encapsulation
{
        public static void main(String[ ] args) {
                Age a= new Age();
                a.setAge(19);
                System.out.println("The age of the person is : "+ a.getAge());
        }
}
```

Output :

The age of the person is : 19

Differences between abstraction and encapsulation :

Abstraction	Encapsulation
It is the method of hiding the unwanted information.	It is a method to hide the data in a single entity or unit
It can be implemented by abstract class and interfaces	It can be implemented by using access modifiers
Abstraction provides access to specific part of data.	Encapsulation hides data, user cannot access data directly.
Here, problems are solved at the interface level.	Here, problems are solved at implementation level.

3.9 INTERFACES -For Video

INTERFACE :

- Interface is an user defined datatype and it is a collection of **public static final initialized data members** and **public abstract methods**.
- It is used to achieve 100 % abstraction and multiple inheritance in Java.

Syntax :

```
interface <interfacename>
{
        public static final initialized data members
        public abstract methods
}
```

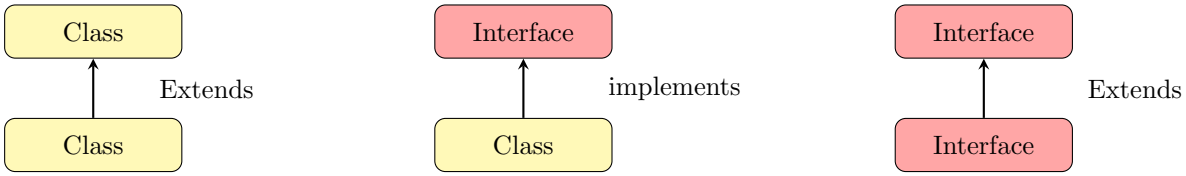
Important points :

1. Like class an interface also a collection of data members and purely abstract methods.
2. It is no need to write abstract keyword before method because by default it is abstract.
3. It is not possible to create object for interfaces as it contains all abstract methods. But we can create references.
4. Since java does not support multiple inheritances in the case of class, by using an interface it can achieve multiple inheritances.
5. Any class can extend only 1 class but any class can implement infinite number of interface.
6. An interface can extend to another interface or more than one interface.
7. A class that implements the interface must implement all the methods in the interface.
8. Inside the Interface not possible to declare instance variables because by default variables are public static final.
9. Inside the Interface constructors are not allowed.
10. Inside the interface main method is not allowed.
11. Inside the interface static ,final,private methods declaration are not recommended.
12. An interface can be inherited by a class by using the keyword 'implements' and it can be inherited by an interface using the keyword 'extends'.

Differences between a Class and an Interface :

Class	Interface
The keyword used to create a class is "class"	The keyword used to create an interface is "interface".
A class can be instantiated [objects can be created].	An Interface cannot be instantiated [objects can't be created].
Classes do not support multiple inheritance.	The interface supports multiple inheritance.
It can contain constructors.	It cannot contain constructors.
It cannot contain abstract methods.	It contains abstract methods only.
Variables & methods declared by any access specifier.	variables and methods are declared as public.
Variables in a class can be static, final, or neither.	All variables are static and final

Relationship between classes and interfaces :



Example-1 :

```
interface In
{
        final int a = 5;
        void display();
        }
class TestClass implements In
{
        public void display()
        {
                System.out.println("Hello");
        }
}
class Example1
{
        public static void main(String[ ] args) {
                TestClass t = new TestClass();
                t.display();
                System.out.println(In.a);
        }
}
```

Output :

```
Hello
5
```

Example-2 :

```
interface Left
{
        int a=55;
}
interface Right
{
        int a=56;
}
class InterfaceTest implements Left,Right
{
        public static void main(String k[ ])
        {
                System.out.println("Value of a in Left Interface : "+Left.a);
                System.out.println("Value of a in Right Interface : "+Right.a);
        }
}
```

Output :

Value of a in Left Interface : 55
Value of a in Right Interface : 56

Example-3:

```
interface vehicle
{
        void seatcapacity();
        void ac();
}
class Lorry implements vehicle
{
        public void seatcapacity()
        {
                System.out.println("Seat capacity in Lorry is 3");
        }
        public void ac()
        {
                System.out.println("There is no AC in Lorry ");
        }
}
class Car implements vehicle
{
        public void seatcapacity()
        {
                System.out.println("Seat capacity in Car is 6");
        }
        public void ac()
        {
                System.out.println("In Car, AC facility is available");
        }
}
class Interfaces
{
        public static void main(String args[ ])
        {
                vehicle v;
                v=new Car();
                v.seatcapacity();
                v.ac();
                v=new Lorry();
                v.seatcapacity();
                v.ac();
        }
}
```

Output :

Seat capacity in Car is 6
In Car, AC facility is available
Seat capacity in Lorry is 3
There is no AC in Lorry

4.1 EXCEPTIONS AND ERRORS -For Video

Exceptions :

- Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException etc.
- It is used to handle the runtime errors so that the regular flow of the application can be preserved.
- Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions.
- When an exception occurs within a method, JVM creates an object. This object is called the exception object.
- It contains information about the exception, such as the name, description and the state of the program when the exception occurred.

Reasons for Exceptions :

- Some of the reasons for exceptions are
  1. Invalid user input
  2. Out of disk memory
  3. Code errors
  4. Opening an unavailable file
  5. Dividing a value by zero
  6. Calling invalid methods

Errors :

- Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, stack overflow errors, infinite recursion, etc.
- Errors are usually beyond the control of the programmer, and we should not try to handle errors.

Differences between Error and Exception :

Error	Exception
It is not caused by programmers.	It is caused by programmers.
It is due to lack of system resources	It is due to wrong logics written by programmer.
Errors are irrecoverable	Exceptions are recoverable by writing correct logics.
JVM doesnot creates any object for errors.	JVM creates Exception objects.
Errors are recoverable by system admins only.	Through exception keywords we can handle exceptions.

Exception Handling :

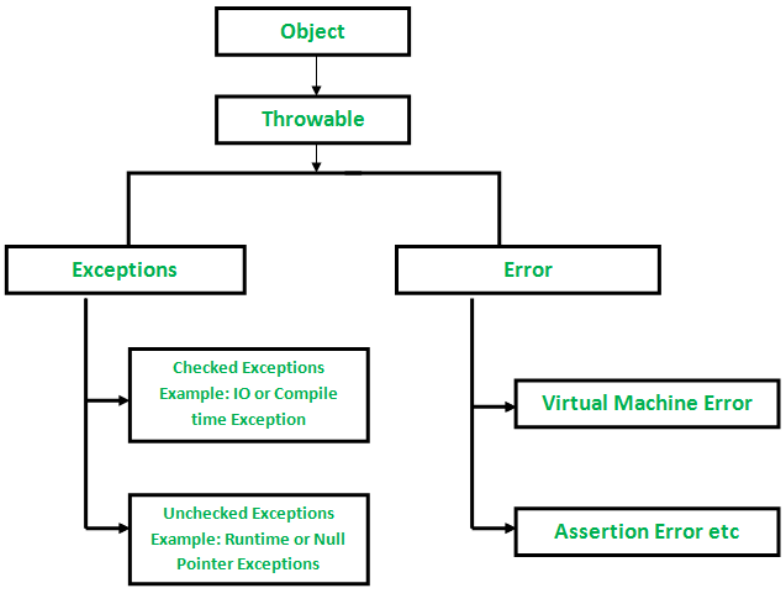
- Generally the system errors are not understandable by the users.
- It is a process of converting system error messages into user understandable format.

4.2 EXCEPTION HIERARCHY AND ITS FLOW

-For Video

EXCEPTION HIERARCHY :

- All exception and error types are subclasses of class Throwable, which is the base class of the hierarchy.
- One branch is headed by Exception. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception.
- Another branch, Error is used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.



FLOW OF EXCEPTION HANDLING :

- Whenver we give invalid inputs to a program the JVM can't understand them then raises an exception and forwards to JRE.
- JRE forwards them to java.lang.Throwable class then throwable class checks and returns it back to JRE.
- Then JRE forwards it to Java API to classify the exception either it belongs to an error or an exception and returns it back to JRE.
- JRE returns it to JVM, then JVM creates an object for appropriate exception class and generates a system error message.
- As the system error messages cannot understandable by user, we need to handle them(exception handling).
- This is the flow of exception handling.

4.3 TYPES OF EXCEPTIONS

- Exceptions can be categorized in two ways:
  1. Built-in Exceptions
  2. User-Defined Exceptions

1. Built-in Exceptions -For Video

- These are developed by sun micro systems and also called as Predefined Exceptions.
- Built-in exceptions are available in Java libraries.
- Built-in Exceptions are again classified into
  1. Asynchronus Exceptions
  2. Synchronus Exceptions
     (a) Checked Exceptions
     (b) Unchecked Exceptions

i. Asynchronus Exceptions :

- These are always going to deal with Hardware problems like Memory Stack Over Flow Exception.
- These exceptions having a superclass called **java.lang.Error**

ii. Synchronus Exceptions :

- These are always going to deal with programmatic errors(due to mistake in logic).
- These exceptions having a superclass called **java.lang.Exception**

a. Checked Exceptions :

- These are the exceptions that are checked at compile time.
- These exceptions occur frequently like classNotFound Exception etc.
- In checked exceptions, there are two types: fully checked and partially checked exceptions.
- A fully checked exception is a checked exception where all its child classes are also checked, like IOException, and InterruptedException.
- A partially checked exception is a checked exception where some of its child classes are unchecked, like an Exception.

b. Unchecked Exceptions :

- These are the exceptions that are not checked at compile time.
- These are Runtime errors like Division By Zero Exception etc..
- These exceptions are usually caused by programming errors, such as attempting to access an index out of bounds in an array or attempting to divide by zero.
- Unchecked exceptions include all subclasses of the RuntimeException class, as well as the Error class and its subclasses.
- That means, In Java, exceptions under Error and RuntimeException classes are unchecked exceptions, everything else under throwable is checked.

4.4 LIST OF BUILT-IN EXCEPTIONS -For Video

- Let us see the list of important built-in exceptions in Java.

1. **ArithmeticException :** It is thrown when an exceptional condition has occurred in an arithmetic operation(Ex : Division By Zero).
2. **ArrayIndexOutOfBoundsException :** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
3. **ClassNotFoundException :** This Exception is raised when we try to access a class whose definition is not found
4. **FileNotFoundException :** This Exception is raised when a file is not accessible or does not open.
5. **IOException :** It is thrown when an input-output operation failed or interrupted
6. **InterruptedException :** It is thrown when a thread is waiting or doing some processing, and it is interrupted.
7. **NoSuchFieldException :** It is thrown when a class does not contain the field (or variable) specified
8. **NoSuchMethodException :** It is thrown when accessing a method that is not found.
9. **NullPointerException :** This exception is raised when referring to the members of a null object. Null represents nothing
10. **NumberFormatException :** This exception is raised when a method could not convert a string into a numeric format.
11. **RuntimeException :** This represents an exception that occurs during runtime.
12. **StringIndexOutOfBoundsException :** It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string
13. **IllegalArgumentException :** This exception will throw the error or error statement when the method receives an argument which is not accurately fit to the given relation or condition. It comes under the unchecked exception.
14. **IllegalStateException :** This exception will throw an error or error message when the method is not accessed for the particular operation in the application. It comes under the unchecked exception.

4.5 EXCEPTION HANDLING BLOCKS AND KEYWORDS

- We have blocks & Keywords used for exception handling are **try, catch, finally, throw** and **throws.**

1. try : -For Video

- The try block contains a set of statements where an exception can occur.
- If we are thinking that some kind of statements in a particular program will raise some errors then we can write those statements in try block.
- Simply, we will write all the **risky code** in try block.
- **Syntax :**

```
try
{
        // statement(s) that might cause exception
}
```

2. catch :

- The catch block is used to handle the uncertain condition of a try block.
- A try block is always followed by a catch block, which handles the exception that occurs in the associated try block.
- Whenever compiler tries to compile the try block, if there is an error then JVM creates specific object and produces error message at runtime.
- Then control goes to catch block which contains all the handling code.
- **Syntax :**

```
catch(Reference)
{
        // statement(s) that handle an exception
}
```

Example program :

```
import java.util.Scanner;
class TryAndCatch
{
        public static void main(String args[ ])
        {
                Scanner sc=new Scanner(System.in);
                try
                {
                        System.out.print("Enter first number : ");
                        int a=sc.nextInt();
                        System.out.print("Enter second number : ");
                        int b=sc.nextInt();
                        System.out.println("Division of two numbers is "+(a/b));
                }
                catch(ArithmeticException ae)
                {
                        System.out.println("Denominator cannot be zero");
                }
        }
}
```

Output :

```
Enter first number : 5
Enter second number : 0
Denominator cannot be zero
```

Note points :

1. Whenever we are writing a try block, the try block assosciates with atleast one catch block. If need we can use n-catch blocks.
2. For a single try block we can write many catch blocks.
3. Even though we write multiple catch blocks, only one catch block will occurs at a time.
4. There will be no intermediate statements between try and catch blocks.

5. Whenever we are trying to handle the exceptions in catch block, if there is a parent to child relationship occurs then must write child to parent exceptions.

Nested try blocks :

- We can write try block inside a try block but each try block must associates with one catch block.
- **Syntax** :

```
try
{
        try
        {
                // statement(s) that might cause exception
        }
        catch(Reference)
        {
                // statement(s) that handle an exception
        }
}
catch(Reference)
{
        // statement(s) that handle an exception
}
```

3. finally :

- It is an independent and optional block.
- Once we write this block, it will executes whether exception occurs or not.
- It is executed after the catch block. We use it to put some common code when there are multiple catch blocks.
- **Syntax :**

```
finally
{
        //statements to be executed
}
```

- We can write only one finally block.

4. throw : -For Video

- It is used to throw an exception object (reference) to the catch block explicity.
- The throw keyword is used to transfer control from the try block to the catch block.
- The throw keyword will be used while working with user defined exceptions.

5. throws :

- The throws keyword is used to handover responsibilities of exception handler to the caller.
- We can use throws keyword while working with checked exceptions.
- Because of the throws keyword, we are prevent from abnormal termination.

Example :

```
import java.util.Scanner;
class ThrowsExample extends RuntimeException
{
        static void calculate(int x, int y) throws ArithmeticException
        {
                System.out.println("Division of two numbers is "+(x/y));
                throw new ThrowsExample();
        }
        public static void main(String k[ ])
        {
                Scanner sc=new Scanner(System.in);
                System.out.print("Enter first number : ");
                int a=sc.nextInt();
                System.out.print("Enter second number : ");
                int b=sc.nextInt();
                calculate(a,b);
        }
}
```

Output :

```
Enter first number : 10
Enter second number : 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Differences between throw and throws :

throw	throws
It is used to throw an explicit exception	It is used to handover the exception
It can be written inside a method.	It can be written along with method signature.
We can't throw multiple exceptions.	We can throw multiple exceptions to caller.

4.6 FLOW CONTROL IN TRY-CATCH-FINALLY

1. Control flow in try-catch-finally clause
   - Case 1: Exception occurs in try block and handled in catch block
   - Case 2: Exception occurs in try-block is not handled in catch block
   - Case 3: Exception doesn't occur in try-block
2. try-finally clause
   - Case 1: Exception occurs in try block
   - Case 2: Exception doesn't occur in try-block

Control flow in try-catch-finally clause

1. Exception occurs in try block and handled in catch block :

- If a statement in try block raised an exception, then the rest of the try block doesn't execute and control passes to the corresponding catch block.
- After executing the catch block, the control will be transferred to finally block(if present) and then the rest of the program will be executed.

2. Exception occurred in try-block is not handled in catch block :

- In this case, the default handling mechanism is followed.
- If finally block is present, it will be executed followed by the default handling mechanism.
- That means finally block will be executed whether try block is handled or not.

3. Exception doesn't occur in try-block :

- In this case catch block never runs as they are only meant to be run when an exception occurs. finally block(if present) will be executed followed by rest of the program.

Control flow in try-finally

- In this case, no matter whether an exception occurs in try-block or not finally will always be executed. But control flow will depend on whether an exception has occurred in the try block or not.

1. Exception occurs in try block :

- If an exception has occurred in the try block then the control flow will be finally block followed by the default exception handling mechanism.

2. Exception doesn't occur in try-block

- If an exception does not occur in the try block then the control flow will be finally block followed by the rest of the program.

4.7 METHODS TO PRINT THE EXCEPTION INFORMATION

- There are 3 ways to find the exception type.
  1. java.lang.Exception
  2. printStackTrace( )
  3. getMessage( )

1. java.lang.Exception :

- It displays the name of exception.
- It also shows the nature of message.
- Example :

```
class MethodsToFindExceptionType
{
        public static void main(String k[])
        {
                try
                {
                        System.out.println("Division of two numbers is "+(10/0));
                }
                catch(Exception ie)
                {
                        System.out.println("The exception is "+ie);
                }
        }
}
```

Output :

The exception is java.lang.ArithmeticException: / by zero

2. printStackTrace( ):

- This method prints exception information in the format of
  1. Name of the exception
  2. Description/Nature of the exception
  3. Code line number

- Example :

```
class MethodsToFindExceptionType
{
        public static void main(String k[])
        {
                try
                {
                        System.out.println("Division of two numbers is "+(10/0));
                }
                catch(Exception ie)
                {
                        ie.printStackTrace();
                }
        }
}
```

Output :

```
java.lang.ArithmeticException: / by zero
at MethodsToFindException.main(MethodsToFindException.java:9)
```

3. getMessage( ):

- This method prints only the description(nature) of the exception.

- Example :

```
class MethodsToFindExceptionType
{
        public static void main(String k[])
        {
                try
                {
                        System.out.println("Division of two numbers is "+(10/0));
                }
                catch(Exception ie)
                {
                        System.out.println("The exception is "+ie.getMessage());
                }
        }
}
```

Output :

```
The exception is / by zero
```

4.8 USER DEFINED EXCEPTIONS

-For Video

- User defined exceptions are defined by programmers.
- As predefined exceptions are classes, JVM create objects and passes them to catch block.
- It won't happen in case of user defined exceptions. Since JVM doesnot identify the user defined exceptions.
- Whenever we are creating user defined exceptions then we need to extend any one of the predefined exception.
- **Example program :**

```
import java.util.Scanner;
public class UserDefinedException extends Exception
{
        UserDefinedException(String s)
        {
                super(s);
        }
        public static void main(String k[ ])
        {
                Scanner sc=new Scanner(System.in);
                try
                {
                        String rno=k[0];
                        System.out.print("Enter your age : ");
                        int age=sc.nextInt();
                        if(age<19)
                        {
                                throw new UserDefinedException("Please enter more than 19");
                        }
                        System.out.println("Your entered age : "+age);
                        if(rno.length()!=7)
                        {
                                throw new UserDefinedException("Enter rollno correctly");
                        }
                        System.out.println("Your entered rollno : "+rno);
                }
                catch(UserDefinedException u)
                {
                        System.out.println(u.getMessage());
                }
        }
}
```

Output :

```
java UserDefinedException N20
Enter your age : 19
Your entered age : 19
Enter rollno correctly
```

FILE HANDLING

4.9 STREAMS AND TYPE OF STREAMS -For Video

Volatile Programs :

- The programs which are stored in primary memory are called Volatile (or) Non-persistent programs.

Non-Volatile Programs :

- The programs which requires secondary storage devices to store the data are called Non-Volatile (or) persistent programs.

File :

- It is a collection of related information.
- It is used to store the data permanently.

Disadvantages of Files :

- It is not secured i.e, unauthorized people can also able to access the data.
- If there is a huge data[tera bytes of data] then it is difficulty to manage.
- To overcome this we will use Data Base Management System to store the data.

How to work with Files :

- We will give data to java applications using keyboard, network sockets and files.
- We will get data from java applications on to monitor, printer,files etc..

Stream :

- It is a kind of channel which is able to allow continuous flow of data.
- It is also defined as flow of data(in the form of bits/bytes) between primary memory and secondary memory.
- Simply, it is used to move the data from one place to another place.
- In files concept, we have input stream and output stream.

Types of streams :

- Streams are two types.
  1. Byte oriented stream
  2. Character oriented stream

1. Byte Oriented Stream :

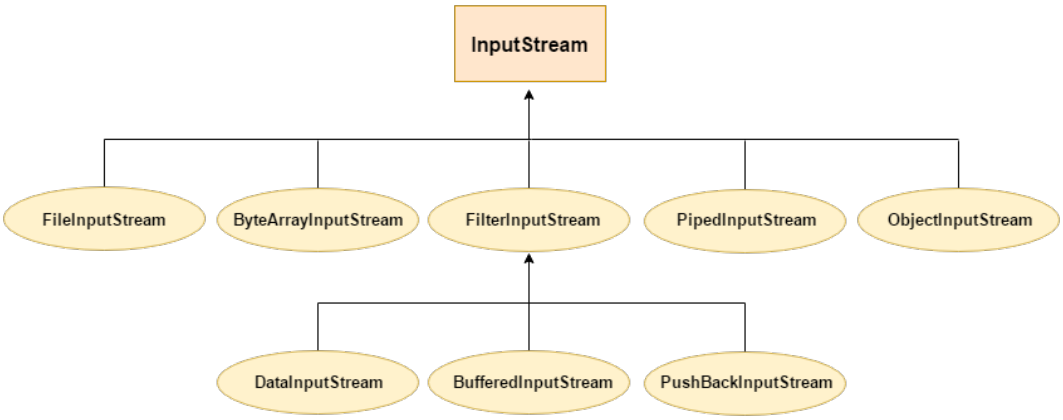
- It is a flow of data in the **form of bytes** from input devices to java applications and java applications to output devices.
- In Byte oriented stream, we can able to handle any kind of data like 'characters', 'images', 'videos', 'audios', etc...
- In Byte oriented stream, we have two type of classes
  a. InputStreamClass b. OutputStreamClass

2. Character Oriented Stream :

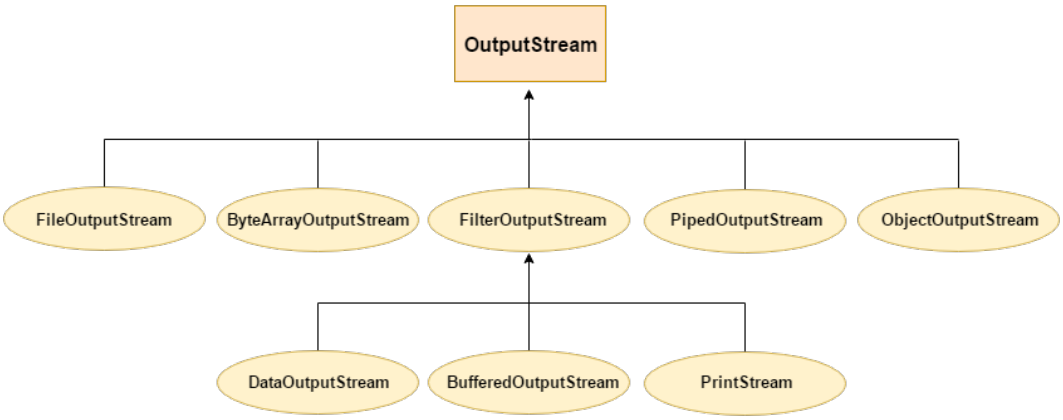
- It is a flow of data in the **form of characters/text** from input devices to java applications and java applications to output devices.
- In Character oriented stream, we can able to handle the data in the form of 'characters' and 'text' only.
- To work with character oriented stream there are predefined classes like
  a. File Reader b. File Writer
- All predefined classes of streams are available in **java.io package**

4.10 HIERARCHY OF STREAMS -For Video

Hierarchy of Byte Oriented Stream :
1. Input Stream Hierarchy :



2. Output Stream Hierarchy :



Note points :

1. In byte oriented streams, all the class names are ends with the word 'stream'[either in input or output streams].

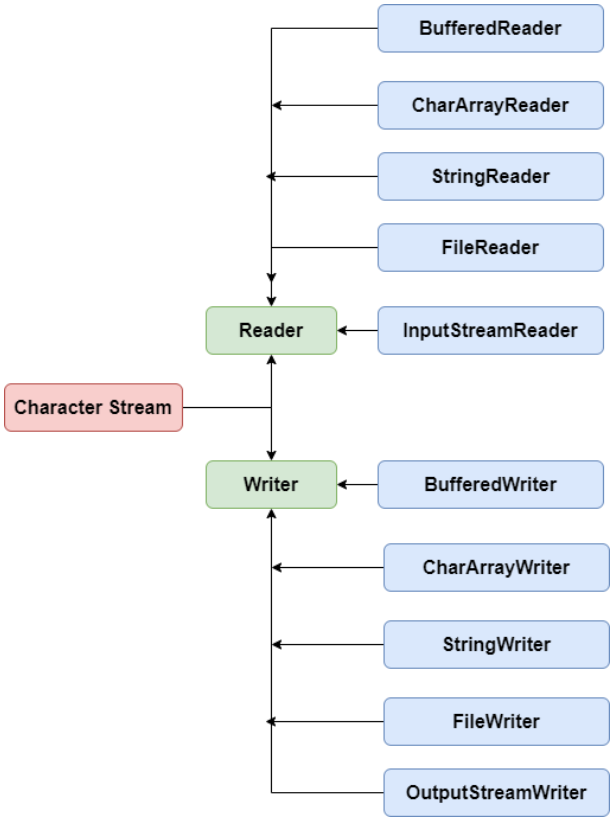
2. In byte oriented streams, each and every data occupies 1 byte of space.

3. In character oriented streams, all the class names are ends with the word 'Reader' in input streams.

4. In character oriented streams, all the class names are ends with the word 'Writer' in output streams.

5. In character oriented streams, each and every data occupies 2 bytes of space.

Hierarchy of Character Oriented Stream :



4.11 FILE CLASS

-For Video

TYPES OF FILES

- Files are of two types
  1. Sequential Files
  2. Random Access Files

Sequential Files :

- We can able to access the data sequentially.
- Java contains 'File class' (available in java.io) to work while using sequential files.

File class :

- File class is Java's representation of a file or directory pathname.
- File class contains several methods for working with the pathname, deleting and renaming files, creating new directories, listing the contents of a directory, and determining several common attributes of files and directories.
- Generally for any class we should create an object likewise we should create the File class object by passing the filename or directory name to it.
- Instances of the File class are immutable i.e once created, the abstract pathname represented by a File object will never change.

How to Create a File Object ?

- A File object is created by passing a string which represents the name of a file, a String, or another File object.
- We can create a File object in three ways as follows
  File f = new File(String Name/File Name);
  or
  File f = new File(String Subdirectory Name, String Name);
  or
  File f = new File(File Object, String Name);

Methods of File Class :

Method	Description	Return Type
createNewFile()	Creates a new, empty file named by this abstract pathname.	boolean
delete()	Deletes the file or directory denoted by this abstract pathname.	boolean
equals(Object obj)	Tests this abstract pathname for equality with the given object.	boolean
exists()	Tests whether file or directory denoted by this abstract pathname exists.	boolean
getAbsolutePath()	Returns the absolute pathname string of this abstract pathname.	String
list()	Returns an array of files and directories in the directory.	String[ ]
getFreeSpace()	Returns the number of unallocated bytes in the partition.	long
getName()	Returns the name of file or directory denoted by this abstract pathname.	String
getPath()	Converts this abstract pathname into a pathname string.	String
setReadOnly()	Marks the file or directory named so that only read operations are allowed.	boolean
isDirectory()	Tests whether the file denoted by this pathname is a directory.	boolean
isFile()	Tests whether the file denoted by this abstract pathname is a normal file.	boolean
length()	Returns the length of the file denoted by this abstract pathname.	long
listFiles()	Returns an array of abstract pathnames denoting files in directory.	File[ ]
mkdir()	Creates the directory named by this abstract pathname.	boolean
renameTo(File destination)	Renames the file denoted by this abstract pathname.	boolean
toString()	Returns the pathname string of this abstract pathname.	String
toURI()	Constructs a file URI that represents this abstract pathname.	URI

Example for creating a file using 1st way :

```
import java.io.File;
class FileCreation1
{
        public static void main(String args[ ]) {
                try {
                        File f=new File("File.java");
                        System.out.println("File Exists : "+f.exists());
                        System.out.println("New File Created : "+f.createNewFile());
                        System.out.println("File Exists : "+f.exists());
                        System.out.println("New File Created : "+f.createNewFile());
                }
                catch(Exception e) {
                        System.out.println("There is a problem in code");
```

```
            }
        }
}
```
Output :
File Exists : false
New File Created : true
File Exists : true
New File Created : false

Example for creating a file using 2nd way :
```
import java.io.File;
class FileCreation2
{
        public static void main(String args[ ])
        {
            try
            {
                File f=new File("D://JAVA PROGRAMS//in","File2.java");
                System.out.println("File Exists : "+f.exists());
                System.out.println("New File Created : "+f.createNewFile());
                System.out.println("File Exists : "+f.exists());
                System.out.println("New File Created : "+f.createNewFile());
            }
            catch(Exception e)
            {
                System.out.println("There is a problem in code");
            }
        }
}
```
Output :
File Exists : false
New File Created : true
File Exists : true
New File Created : false

Example for creating directory :
```
import java.io.File;
class FileCreation3
{
        public static void main(String args[]) throws Exception {
            File f=new File("D://JAVA PROGRAMS//in");
            f.mkdir();
            File f1=new File(f,"IIIT.java");
            System.out.println("Directory created successfully");
            f1.createNewFile();
        }
}
```
Output :
Directory created successfully

4.12 INPUT STREAM CLASS AND OUTPUT STREAM CLASS

Input Stream Class : -For Video

- It is an abstract class developed by java developers.
- It is a superclass of various Input Stream Classes.
- It is used to read the data from the files.
- It consists three types of methods.
  1. read( )
  2. available( )
  3. close( )

1. read( )

- It is used to read the data byte by byte.
- Whenever the read( ) method returns '-1' then it indicates that we reached to end of the file.

2. read(byte[ ])

- It is used to read the array of data byte by byte.

3. available( )

- It returns the total number of bytes that can be read from the current Input Stream Class.

4. close( )

- It is used to close the current Input stream class.

Output Stream Class :

- It is also an abstract class developed by java developers.
- It is a superclass of various output Stream Classes.
- It is used to write the data into the files.

Methods of Output Stream Class :

Method	Description
write(int)	It is used to write data byte by byte
write(byte)	It is used to write an array of data
flush( )	It is used to flushes the output stream class
close( )	It is used to close current output stream class

4.13 FILE-INPUT-STREAM AND FILE-OUTPUT-STREAM

File Input Stream : -For Video

- It carries data from source file to java application.
- **Syntax :**
  FileInputStream fis = new FileInputStream("FileName");
  or
  File f = new File("FileName");
  FileInputStream fis = new FileInputStream(f);
- If the file not exists then FileNotFoundException raises.

- We can use Input Stream Class methods here.

Example using 1st way :

```
import java.io.*;
public class InputFile
{
        public static void main(String args[]) throws Exception
        {
                FileInputStream fis=new FileInputStream("abc.txt");
                int i=0;
                while((i=fis.read())!=-1)
                {
                        System.out.print((char)i);
                }
                fis.close();
        }
}
```

Output :

Hello, We are learning File Handling.

Example using 2nd way :

```
import java.io.*;
public class InputFile
{
        public static void main(String args[]) throws Exception
        {
                FileInputStream fis=new FileInputStream("abc.txt");
                byte[] b=new byte[fis.available()];
                fis.read(b);
                String data=new String(b);
                System.out.println(data);
                fis.close();
        }
}
```

Output :

Hello, We are learning File Handling.

File Output Stream :

- It is used to carry data from java applications to output devices.
- **Ways to create :**
  FileOutputStream fos = new FileOutputStream("FileName");
  or
  File f = new File("FileName");
  FileOutputStream fos = new FileOutputStream(f);
- If the file not exists then new file will be created.
- We can use Output Stream Class methods here.

Example :

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

```
public class OutputFile
{
        public static void main(String args[ ]) throws Exception
        {
                FileOutputStream fos=new FileOutputStream("java.txt");
                fos.write('v');
                String s="James Gosling is the Father of java ";
                byte[] j=s.getBytes();
                fos.write(j);
                System.out.println("Successfully written into the file");
        }
}
```

Output :

Successfully written into the file

4.14 FILE READER AND FILE WRITER

[-For Video]

File Reader :

- It belongs to character oriented stream class.
- It is used to read characters or text from the file.
- Structure :
  FileReader fr=new FileReader("FileName");
  or
  File f=new File("FileName");
  FileReader fr=new FileReader(f);
- If the file not exists then Exception raises.

File Writer :

- It belongs to character oriented stream class.
- It is used to write characters or text into the file.
- Structure :
  FileWriter fw=new FileWriter("FileName");
  or
  File f=new File("FileName");
  FileWriter fw=new FileWriter(f);
- If the file not exists then new file will be created.

Example-1 :

```
import java.io.*;
class FileReaderAndWriter
{
        public static void main(String args[]) throws Exception
        {
                FileWriter fw=new FileWriter("Write.txt");
                fw.write("This is an example program of File Writer and File Reader");
                fw.flush();
```

```
                System.out.println("Successfully written into the file");
                FileReader fr=new FileReader("Write.txt");
                int i=0;
                while((i=fr.read())!=-1)
                {
                        System.out.print((char)i);
                }
        }
}
```

Output :

Successfully written into the file
This is an example program of File Writer and File Reader

Example-2 :

```
import java.io.*;
class FileWriterEx
{
        public static void main(String args[ ]) throws Exception
        {
                FileWriter fw=new FileWriter("Write.txt");
                char[] ch='r','g','u','k','t';
                String s=" Vedavyas ";
                fw.write("Hello cse! ");
                fw.write(s);
                fw.write(ch);
                fw.write(ch,1,4);
                fw.flush();
                System.out.println("Successfully Written into the file");
                FileReader fr=new FileReader("Write.txt");
                int i=0;
                while((i=fr.read())!=-1)
                {
                        System.out.print((char)i);
                }
                fw.close();
                fr.close();
        }
}
```

Output :

Successfully Written into the file
Hello cse! Vedavyas rguktgukt

ReadLine( ) Method :

- It is used to read the character data line by line.

- It doesnot supported by FileReader. It is supported in BufferedReader.

4.15 BUFFERED-READER AND BUFFERED-WRITER -For Video

BufferedReader :

- It belongs to character oriented stream class.
- It is used to read characters or text from the file.
- Both BufferedReader and BufferedWriter cannot communicate with the file directly.
- **Structure** :
  FileReader fr=new FileReader("FileName");
  BufferedReader br=new BufferedReader(Reader Object)
  or
  BufferedReader br=new BufferedReader(new FileReader("FileName"));
- If contains methods like read(), read(ch[ ]), close(),and readLine()
  Example :

```
import java.io.*;
class BFile
{
        public static void main(String args[ ]) throws Exception
        {
                BufferedReader br=new BufferedReader(new FileReader("abc.txt"));
                String line=br.readLine();
                while(line!=null)
                {
                        System.out.println(line);
                        line=br.readLine();
                }
        br.close();
        }
}
```

  Output :

```
This is Line no-1.
This is Line no-2.
This is Line no-3.
This is Line no-4.
```

BufferedWriter :

- It belongs to character oriented stream class.
- It is used to write characters or text into the file line by line.
- **Structure** :
  FileWriter fw=new FileWriter("FileName");
  BufferedWriter bw=new BufferedWriter(Writer Object)
  or
  BufferedWriter bw=new BufferedWriter(new FileWriter("FileName"));
- It contains methods like write, write(ch[ ]), close(), flush()and newLine()
- **newLine( ) :** It is used to insert a new line into the file.

Example :

```
import java.io.*;
class BFile
{
        public static void main(String args[ ]) throws Exception
        {
                BufferedWriter bw=new BufferedWriter(new FileWriter("abc.txt"));
                bw.write(100);
                bw.newLine();
                bw.write("100");
                bw.newLine();
                bw.write("RGUKT Nuzvid");
                bw.flush();
                System.out.println("Successfully written into the file");
                BufferedReader br=new BufferedReader(new FileReader("abc.txt"));
                String line=br.readLine();
                while(line!=null)
                {
                        System.out.println(line);
                        line=br.readLine();
                }
                bw.close();
                br.close();
        }
}
```

Output :

```
Successfully written into the file
d
100
RGUKT Nuzvid
```

Draw backs of BufferedWriter :

- It only accepts integers along characters and converts into respective ascii values.
- It doesnot write float or double values.
  Example :
  bw.write(100); //d
  bw.write(10.5); //compile time error

4.16 PRINT WRITER -For Video

- It is used to write data into file and all the methods of FileWriter will support here.
- **Structure :**
  PrintWriter pw=new PrintWriter("FileName");
  or
  FileWriter f=new FileWriter("FileName");
  PrintWriter pw=new PrintWriter(f);
  or
  PrintWriter pw=new PrintWriter(Writer Object);

Advantages :

- It can accept any type of data.
- No need to use newLine() method instead we have println() method to insert a new line.

Example :

```
import java.io.*;
class PFile
{
        public static void main(String args[ ]) throws Exception
        {
                PrintWriter pw=new PrintWriter(new FileWriter("abc.txt"));
                pw.println(100);
                pw.println(22.34f);
                pw.println(true);
                pw.println("RGUKT Nuzvid");
                pw.flush();
                System.out.println("Successfully written into the file");
                BufferedReader br=new BufferedReader(new FileReader("abc.txt"));
                String line=br.readLine();
                while(line!=null)
                {
                        System.out.println(line);
                        line=br.readLine();
                }
                pw.close();
                br.close();
        }
}
```

Output :

```
Successfully written into the file
100
22.34
true
RGUKT Nuzvid
```

4.17 INPUT-STREAM-READER & OUTPUT-STREAM-WRITER

InputStreamReader : -For Video

- It is an extension of Reader class and it is used to convert data in bytes to data in characters.
- It is going to read data in bytes and decodes into characters.
- It supports methods like read(), read(char[ ] array), read(char[ ] array, int StartingIndex, int length)

Constructors of InputStreamReader :

1. InputStreamReader(object);
2. InputStreamReader(System.in);
3. InputStreamReader(object, CharSet.forName());

Example :

```
import java.io.*;
class ISReader
{
        public static void main(String k[]) throws Exception
        {
                char []arr=new char[100];
                FileInputStream fis=new FileInputStream("abc.txt");
                InputStreamReader isr= new InputStreamReader(fis);
                isr.read(arr);
                System.out.print(arr);
                fis.close();
        }
}
```

Output :

```
Java is the one of the best language.
10
1.5
```

OutputStreamWriter :

- It is an extension of Writer class.
- It is used to convert character data in to byte format.
- It supports methods like write(), write(char[ ] array), write(String data), flush() and getEncoding()

Example :

```
import java.io.*;
import java.nio.charset.*;
class OpSWriter
{
        public static void main(String k[]) throws Exception
        {
                FileOutputStream fos=new FileOutputStream("hello.txt");
                OutputStreamWriter osw =new OutputStreamWriter(fos);
                osw.write(20);
                osw.flush();
                char arr[]=new char[100];
                FileInputStream fis=new FileInputStream("hello.txt");
                InputStreamReader isr1= new InputStreamReader(fis);
                InputStreamReader isr2= new InputStreamReader(fis, Charset.forName("UTF-8"));
                System.out.println("Data in the stream.....");
                System.out.println(isr1.read());
                System.out.println("Without encoding : "+isr1.getEncoding());
                System.out.println("With encoding : "+isr2.getEncoding());
        }
}
```

Output :

```
Data in the stream.....
20
Without encoding : Cp1252
With encoding : UTF8
```

4.18 DYNAMIC INPUT APPROACHES

-For Video

- We can read data dynamically by using the following
  1. BufferedReader class
  2. Scanner class
  3. Console

1. BufferedReader class :

- As we already seen about the BufferedReader Class, here we can use it to read the data as follows.
- **Structure :**
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
- Here we used InputStreamReader class to convert the data in bytes to characters.
- We also included the Reader Object (System.in)
  Example :

```
import java.io.*;
class BufferedReaderEx
{
        public static void main(String args[]) throws Exception
        {
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Enter the name : ");
                String name=br.readLine();
                System.out.print("Enter the age : ");
                String age=br.readLine();
                int k=Integer.parseInt(age);
                System.out.println("Name : "+name);
                System.out.println("Age : "+k);
        }
}
```

Output :

```
Enter the name : Vedavyas
Enter the age : 19
Name : Vedavyas
Age : 19
```

Drawbacks of BufferedReader class :

- It takes data in string format whenever we give data as integers, float , double..
- So we need to use wrapper classes to convert strings into different datatypes.

2. Scanner class

- Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings.
- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.
- To read numerical values of a certain data type XYZ, the function to use is nextXYZ(). For example, to read a value of type short, we can use nextShort()

- We need not to use wrapper classes specifically here.

Drawbacks :

- Every time we need to write System.out.print();
- Every time we need to write nextXYZ methods to store the data.

3. Console :

- It is useful to encrypt the input data like passwords, pins etc..
- It contains methods readLine() and readPassword();
- Whatever the data we passed in those constructors will visible directly without using print and println methods.
  Example :

```
import java.io.*;
class ConsoleEx
{
        public static void main(String args[]) throws Exception
        {
                Console c=System.console();
                String name=c.readLine("Enter the Username : ");
                char []pwd=c.readPassword("Enter password : ");
                String password=new String(pwd);
                if(name.equals("Mahesh")&&password.equals("mpc"))
                {
                        System.out.println("Login Successfull");
                }
                else
                {
                        System.out.println("Login not Successfull");
                }
        }
}
```

Output :

```
Enter the Username : Mahesh
Enter password :
Login Successfull
```

Note : Here the entered password is not visible in the output due to readPassword() method.

4.19 DATA-INPUT-STREAM AND DATA-OUTPUT-STREAM

Data Input Stream **-For Video**

- It is an extended class from InputStream class.
- As we seen that FileInputStream class can't read double data. To overcome this we use DataInputStream class.
- **Structure :**
  DataInputStream dis=new DataInputStream(new FileInputStream("FileName"));
- Here we can use methods like read(), read(char[] array), close()

Constructors of DataInputStream class :

- DataInputStream dis=new DataInputStream(new FileInputStream("FileName"));
  dis.readUTF(); =>For strings
  dis.readInt();
  dis.read();
  dis.readChar();
  dis.readBoolean();

Data Output Stream

- It is an extended class from OutputStream class.
- As we seen that FileOutputStream class can't write double data. To overcome this we use DataOutputStream class.
- **Structure :**
  DataOutputStream dos=new DataOutputStream(new FileOutputStream("FileName"));
- Here we can use methods like write(), write(char[] array), close()

Constructors of DataOutputStream class :

- DataOutputStream dos=new DataOutputStream(new FileOutputStream("FileName"));
  dos.writeUTF(); =>For strings
  dos.writeInt();
  dos.write();
  dos.writeChar();
  dos.writeBoolean();

Example :

```
import java.io.*;
class DataInputOutputStreams
{
        public static void main(String args[ ]) throws Exception
        {
                FileOutputStream fos=new FileOutputStream("f1.txt");
                DataOutputStream dos=new DataOutputStream(fos);
                dos.writeUTF("RGUKT");
                dos.writeInt(6);
                dos.writeBoolean(true);
                FileInputStream fis=new FileInputStream("f1.txt");
                DataInputStream dis=new DataInputStream(fis);
                String s=dis.readUTF();
                int n=dis.readInt();
                Boolean j=dis.readBoolean();
                System.out.println(s);
                System.out.println(n);
                System.out.println(j);
        }
}
```

Output :

```
RGUKT
6
true
```

Draw Back of DataInputStream and DataOutputStream :

- In which order you are writing the data into a file, in the same order only you need to read the data.

4.20 BYTE-ARRAY-INPUT AND OUTPUT-STREAMS

ByteArrayInputStream : **-For Video**

- It is a kind of Byte Oriented Stream.
- It is used to read byte array as input stream.
- It is an extension of Input Stream class.
- **Structure :**
  ByteArrayInputStream bais=new ByteArrayInputStream(byte [ ]array);
- It contains methods like read(), read(bytes), read(byte[] array, int StartingValue, int length), close(), available(), and skip()

- **Example :**

```
import java.io.*;
class ByteArrayInputStreamEx
{
        public static void main(String k[ ]) throws Exception
        {
                byte[ ] array={1,2,3,4,5,6,7,8,9,10};
                ByteArrayInputStream bais=new ByteArrayInputStream(array);
                for(int i=0;i<array.length;i++)
                {
                        int data=bais.read();
                        bais.skip(2);
                        if(data==-1)
                        {
                                break;
                        }
                        System.out.println(data);
                }
        }
}
```

Output :

```
1
4
7
10
```

ByteArrayOutputStream :

- Classes like FileOutputStream class, DataOutputStream class etc.. are to write the data into a single file at a time.
- To write the same data into multiple files at a time, we use ByteArrayOutputStream class.
- We use **writeTo()** method to write the data into multiple files at a time

Example :

```
import java.io.*;
class ByteArrayOutputStreamEx
{
        public static void main(String k[]) throws Exception
        {
                byte[] array={1,2,3,4,5,6,7,8,9,10};
                FileOutputStream f1=new FileOutputStream("File1.txt");
                FileOutputStream f2=new FileOutputStream("File2.txt");
                FileOutputStream f3=new FileOutputStream("File3.txt");
                ByteArrayOutputStream baos=new ByteArrayOutputStream();
                baos.write(array);
                baos.writeTo(f1);
                baos.writeTo(f2);
                baos.writeTo(f3);
                System.out.println("Successfully written into multiple files");
                f1.close();
                f2.close();
                f3.close();
        }
}
```

Output :

Successfully written into multiple files

*****

UNIT-5. PACKAGES AND MULTI-THREADING

5.1 PACKAGES AND ITS TYPES -For Video

PACKAGE :

- Package is defined as **system defined physical folder.**
- Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.
- It contains files which are arranged in hierarchial order to access the data efficiently.
- Packages are used for :
  1. Preventing naming conflicts.
     Example : There can be two classes with name Student in two packages, rgukt.nuzvid.cse.Student and rgukt.nuzvid.ece.Student
  2. Searching/locating and usage of classes, interfaces, enumerations and annotations easier.
  3. Providing controlled access: protected and default have package level access control.
  4. By following the folder approach, we can overcome visibility mechanism.
- Package names and directory structure are closely related.
  Example : if a package name is rgukt.cse.student then there are three directories, rgukt,cse and student such that student is present in cse and cse is present inside rgukt.

PACKAGE NAMING COVENTIONS :

- Packages are named in reverse order of domain names, i.e., www.rguktnuzvid.ac.in
  For example, in a college, the recommended convention is college.tech.cse, college.tech.ee, etc.

ADDING A CLASS TO A PACKAGE :

- We can add more classes to a created package by using package name at the top of the program and saving it in the package directory.
- We need a new java file to define a public class, otherwise we can add the new class to an existing .java file and recompile it.

SUBPACKAGES :

- Packages that are inside another package are called the subpackages.
- These are not imported by default, they have to imported explicitly.

TYPES OF PACKAGES :

- Packages are two types. They are
  1. Pre-defined packages.
  2. User defined packages.

5.2 PRE-DEFINED PACKAGES -For Video

- These are developed by sun micro systems.
- These are also called as **Built-in packages** or **Core packages**.
- There are approximately 14 predefined packages are available.
- Let us see some of the commonly used built-in packages :

i) java.lang package:

- It is a basic package which is used to convert string data to fundamental datatypes, displaying result on console, having garbage collector etc...
- This package is automatically imported.

ii) java.util package:

- It contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
- We can use it to read the data through keyboard.
- We can also use it to implement/develop quality and reliable applications.

iii) java.io package:

- It contains classes for supporting input / output operations.
- It is used to implement file handling applications.

iv) java.applet package:

- It contains classes for creating Applets.
- It is used to develop browser oriented applications(which are runs through browsers).

v) java.awt package:

- AWT means Abstract Window Toolkit.
- This package is used to implement GUI(Graphical user interface) components.

vi) java.awt.event :

- This is a subpackage of awt package.
- It is used to add functionality to the GUI components.
  Example : click button => makes action.

vii) java.sql package :

- It contains classes for supporting networking operations.

viii) java.sql package :

- This package is used to work with data base.
- That means to retrieve the data from data base and to update the data into data base etc..

Important points :

1. Every class is part of some package.
2. All classes/interfaces in a file are part of the same package. Multiple files can specify the same package name.
3. If package name is specified, the file must be in a subdirectory called name (i.e., the directory name must match the package name).
4. We can access public classes in another package using: **packagename.classname**

5.3 CREATING A PACKAGE

- First we should choose a name for the package we are going to create and include.
- We can create a package by using the keyword **package**.
- Further inclusion of classes, interfaces, annotation types, etc that is required in the package can be made in the package.
  For example, the below single statement creates a package name called "FirstPackage".
- To create a class inside a package
  1. First declare the package name as the first statement of our program.
  2. Then include a class as a part of the package
- **Example :**

```
package FirstPackage;
class Welcome
{
        public static void main(String[ ] args)
        {
                System.out.println("First package imported successfully");
        }
}
```

- **procedure to compile and run the above program :**
  1. **Command :** javac Welcome.java
     => The above command will give us Welcome.class File.
  2. **Command :** javac -d . Welcome.java
     => This command will create a new folder called FirstPackage.
  3. **Command :** java FirstPackage.Welcome
     => This command will generates the output.

5.4 USER DEFINED PACKAGES -For Video

- These are the packages that are defined by the user.
- **Single level packages :**
  Example :

```
package rguktnzd;
class SingleLevelPackage
{
        public static void main(String args[])
        {
                System.out.println("Single Level Package");
        }
}
```

  Compile : javac -d . SingleLevelPackage.java;
  Run : java rguktnzd.SingleLevelPackage;

- **Multilevel packages :**
  Example :

```
package in.ac.rgukt.nuzvid;
public class MultilevelPackage
{
        public static void main(String args[ ])
        {
                System.out.println("MultiLevel Package created successfully");
        }
}
```
Compile : javac -d . MultilevelPackage.java
Run : java in.ac.rgukt.nuzvid.MultilevelPackage

- **Multiple packages :**
  Example :

```
package multiplepackage1;
public class MultiplePackage1
{
        int a=535;
        public void hi()
        {
                System.out.println("Hello");
        }
        public void num()
        {
                System.out.println("a value is "+a);
        }
}

package multiplepackage2;
public class MultiplePackage2
{
        int b=1000;
        public void hello()
        {
                System.out.println("Hi");
        }
        public void num()
        {
                System.out.println("b value is "+b);
        }
}

package multiplepackage3;
import multiplepackage1.MultiplePackage1;
import multiplepackage2.MultiplePackage2;
public class MultiplePackage3
{
        public static void main(String k[ ])
        {
                MultiplePackage1 m1=new MultiplePackage1();
```

```
                m1.hi();
                m1.num();
                MultiplePackage2 m2=new MultiplePackage2();
                m2.hello();
                m2.num();
                }
}
```

Compile : javac -d . MultiplePackage3.java
Run : java multiplepackage3.MultiplePackage3
Output :

```
Hello
a value is 535
Hi
b value is 1000
```

5.5 ACCESS CONTROL PROTECTION -For Video

- Let us see the accessibility of private, protected, public and default in packages.

	Private	Default	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package class	No	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Other package class	No	No	No	Yes
Other package subclass	No	No	Yes	Yes

Example :
C1 class in package accessmodifiers1

```
package accessmodifiers1;
public class C1
{
        private int a=1;
        int b=2;
        protected int c=3;
        public int d=4;
}
```

C2 class in package accessmodifiers1

```
package accessmodifiers1;
public class C2
{
        public static void main(String args[ ])
        {
                C1 obj=new C1();
                System.out.println(obj.a);      //a has private access in C1
                System.out.println(obj.b);      // 2
                System.out.println(obj.c);      // 3
                System.out.println(obj.d);      // 4
        }
}
```

C3 class in package accessmodifiers2

```
package accessmodifiers2;
import accessmodifiers1.C1;
public class C3
{
        public static void main(String args[ ])
        {
                C1 obj=new C1();
                System.out.println(obj.a);          // a has private access in C1
                System.out.println(obj.b);          // b is not public in C1
                System.out.println(obj.c);          // c has protected access in C1
                System.out.println(obj.d);          // 4
        }
}
```

C4 class in package accessmodifiers2

```
package accessmodifiers2;
import accessmodifiers1.C1;
public class C4 extends C1
{
        public static void main(String args[ ])
{
                C4 obj=new C4();
                System.out.println(obj.a);          //a has private access in C1
                System.out.println(obj.b);          // b is not public in C1
                System.out.println(obj.c);          // 3
                System.out.println(obj.d);          // 4
        }
}
```

MUTLI THREADING

5.6 CONCEPTS OF MULTI THREADING -For Video

Uni-Programming :

- The process of executing only one program at a time is called Uni-Programming.
- Here one CPU can perform one work at a time.

Multi-Programming :

- The process of executing multiple programs at a time is called Multi-Programming.
- Here one CPU can perform many tasks simultaneously.

Multi Tasking :

- Multi-tasking is a logical extension of multiprogramming.
- Multitasking is the ability of an OS to execute more than one task simultaneously on a CPU machine. These multiple tasks share common resources (like CPU and memory).
- With a single CPU only one task can be running at one point of time, so CPU uses context switching to perform multitasking.
- Context switch (Context means state) is the process of storing and restoring the state of a process so that execution can be resumed from the same point at a later time.
- We can implement multi tasking in two ways :
  1. Multi-Processing
  2. Multi-Threading

1. Multi Processing :

- Multiprocessing is a type of multitasking based upon processes i.e. context switching is done in-between processes.
- Simply, the processing of multiple tasks or multiple programs is called Multi processing.
- **For Example :** Typing in notepad, Listening music and downloading file from internet at the same time. Here in example we can clearly see that all applications are independent.
- Multiprocessing is the type of multitasking which is handled at operating system level.
- **Process :** A process is simply a normal program that can be like audio player, editor etc which is going to perform some action.
- A process has a self-contained execution environment i.e. allocates separate memory area.
- Context switch time is more in case of processes because switch is done between different memory areas.
- When a single process is performing some operation in webserver, if a new request came then webserver going to create a child process.

DrawBacks of Multi-Processing :

1. For each and every process it having its own address in the memory.
2. The context switching between the processes is very high or it takes high time for the communication between the processes.
3. For all the processes the file description is same. Even though the data and code is same for all processes, we need to maintain different copies.

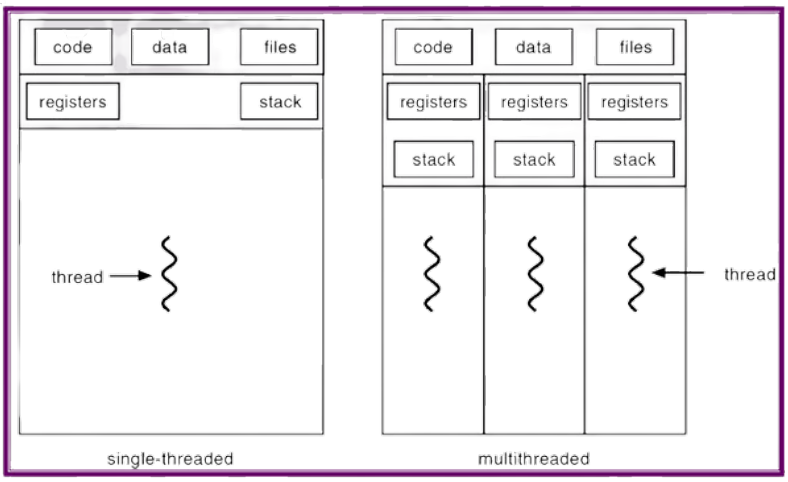
2. Multi Threading :

- Multithreading is a type of multitasking based upon threads i.e. context switching is done in-between threads.
- In case of multithreading, multiple independent tasks are executed simultaneously.
- These independent tasks are the part of same application.
- Multithreading is the type of multitasking which is handled at program level.
- Here no need to create memory space for each thread. It is going to share same address space for all threads.
- Every thread having same code and data.

Thread :

- A thread is a light weight process .
- Thread uses process's execution environment i.e. memory area.
- Context switch time is less in case of threads because switch is done within the same process's memory area.
- A thread can't be exist without process, it exist within the process.

Structure of a Thread :



Need of Multi-Threading :

1. To reduce the CPU ideal time.
2. To increase the performance of the system.
3. To execute multiple tasks simultaneously.
4. To execute the programs independently.

5.7 DIFFERENCES BETWEEN PROCESS AND THREAD

PROCESS :

1. Process is considered as heavy weight component.
2. One process can have multiple threads.
3. Process means any program is in execution.
4. The process takes more time to terminate.
5. It takes more time for creation.
6. It also takes more time for context switching.
7. The process is less efficient in terms of communication.
8. Multiprogramming holds the concepts of multi-process.
9. The process is isolated.
10. Process switching uses an interface in an operating system.
11. If one process is blocked then it will not affect the execution of other processes
12. The process has its own Process Control Block, Stack, and Address Space.
13. Changes to the parent process do not affect child processes
14. A system call is involved in it.
15. The process does not share data with each other.

THREAD :

1. Thread is considered as lightweight component.
2. One thread can't have multiple process.
3. Thread means a segment of a process.
4. The thread takes less time to terminate.
5. It takes less time for creation.
6. It takes less time for context switching.
7. Thread is more efficient in terms of communication.
8. A single process consists of multiple threads.
9. Threads share memory.
10. Thread switching does not require calling an operating system.
11. If a user-level thread is blocked, then all other user-level threads are blocked.
12. Thread has Parents' PCB, its own Thread Control Block, Stack and common Address space.
13. Any changes to the main thread may affect the behavior of the other threads of the process.
14. No system call is involved, it is created using APIs.

5.8 THREAD CLASS

-For Video

- Thread is a line of execution within a program. Each program can have multiple associated threads.
- Each thread has a priority which is used by the thread scheduler to determine which thread must run first.
- Java provides a thread class that has various method calls in order to manage the behavior of threads by providing constructors and methods to perform operations on threads.
- Thread class is present in **java.lang** package
- A method frequently used (to start execution) in thread class known as the **start()** method. This method implicitly calls **run()** method which is also a method of thread class and begins executing the body of the run() method.
- As run() method is unimplemented method, we must override the run() method.

Constructors of Thread class :

1. Thread();
2. Thread(Runnable);
3. Thread(Runnable, java.security.AccessControlContext);
4. Thread(ThreadGroup, Runnable);
5. Thread(String);
6. Thread(ThreadGroup, String);
7. Thread(Runnable, String);
8. Thread(ThreadGroup, Runnable, String);
9. Thread(ThreadGroup, Runnable, String, long);
10. Thread(ThreadGroup, Runnable, java.lang.String, long, boolean);

Methods of Thread class :

Method	Action Performed
activeCount()	Returns no of active threads in current thread's thread group and its subgroups
checkAccess()	Determines if the currently running thread has permission to modify this thread
currentThread()	Returns a reference to the currently executing thread object
enumerate(Thread[ ] tarray)	Copies the threads into the specified array
getId()	Returns the identifier of this Thread
getName()	Returns the current thread's name
getPriority()	Returns the current thread's priority
getState()	Returns the state of current thread
getThreadGroup()	Returns the thread group to which this thread belongs
interrupt()	Interrupts the current thread
isAlive()	Tests if the current thread is alive
isDaemon()	Tests if the current thread is a daemon thread
join()	Waits for the current thread to die
join(long millisec)	Waits at most millis milliseconds for this thread to die
run()	It is an unimplemented method
setDaemon(boolean on)	Marks this thread as either a daemon thread or a user thread
setName(String name)	Changes the name of this thread to be equal to the argument name.
setPriority(int newPriority)	Changes the priority of this thread
sleep(long millis)	Causes the currently executing thread to pause for specified time.
start()	Causes a thread to begin execution; the JVM calls the run method of this thread
yield()	Pauses the current execution & gives chance to other threads for execution

5.9 RUNNABLE INTERFACE

-For Video

- Runnable is an interface that is to be implemented by a class whose instances are intended to be executed by a thread.
- Runnable is available in **java.lang** package and it contains only one abstract method i.e, **public abstract void run()** method.
- As run() method is abstract, we need to implement it and override to create a thread.
- For creating a thread we need to use start() method. But there is no start() method in Runnable interface.
- So we need to create an object for Thread class to use start() method.
- As we know that start() method implicitly calls run() method, but we need to use run() method which is present in Runnable interface
- So we need to use Thread class constructor i.e, **Thread(Runnable)**.

5.10 CREATING A THREAD

- In java, we can create a thread by using the following two ways
  1. By extending the Thread Class.
  2. By implementing the Runnable Interface.

Note :

- By default, every java program having a thread called **Main Thread**.

1. By extending the Thread Class

- **Example :**

```
class Hello extends Thread
{
        public void run()
        {
                for(int i=0;i<3;i++)
                {
                        System.out.println("Child Thread");
                }
        }
}
class CreatingThreadByExtending
{
        public static void main(String args[ ])
        {
                Hello h=new Hello();
                h.start();
                for(int i=0;i<3;i++)
                {
                        System.out.println("Main Thread");
                }
        }
}
```

Output :
Main Thread
Main Thread
Child Thread
Child Thread
Main Thread
Child Thread
Note : We can't expect the exact output. It changes everytime.

2. By implementing the Runnable Interface

- Steps to create a new thread using Runnable :
  1. Create a Runnable implementer and implement the run() method.
  2. Instantiate the Thread class and pass the implementer to the Thread, Thread has a constructor which accepts Runnable instances.
  3. Invoke start() of Thread instance, start internally calls run() of the implementer. Invoking start() creates a new Thread that executes the code written in run(). Calling run() directly doesn't create and start a new Thread, it will run in the same thread. To start a new line of execution, call start() on the thread.

- **Example :**

```
class Hello implements Runnable
{
        public void run()
        {
                for(int i=0;i<3;i++)
                {
                        System.out.println("Child Thread");
                }
        }
}
class CreatingThreadByImplementing
{
        public static void main(String args[ ])
        {
                Hello h=new Hello();
                Thread t=new Thread(h);
                t.start();
                for(int i=0;i<3;i++)
                {
                        System.out.println("Main Thread");
                }
        }
}
```

Output :
Main Thread
Main Thread
Child Thread
Child Thread
Main Thread
Child Thread
Note : We can't expect the exact output. It changes everytime.

- **Note :** The second way is the best way to create a thread, because if we want to inherit a class along Thread class or Runnable interface, it is not possible through 1st way.

5.11 CREATING MULTIPLE THREADS -For Video

1. By Extending Thread class :

```
class A extends Thread
{
        public synchronized void run()
        {
                System.out.println("Thread is created with the Name : "+Thread.currentThread().getName());
        }
}
class CreatingMultipleThreads1
{
        public static void main(String args[ ])
        {
                A a1=new A();
                A a2=new A();
                A a3=new A();
                A a4=new A();
                A a5=new A();
                a1.start();
                a2.start();
                a3.start();
                a4.start();
                a5.start();
        }
}
```

Output :

```
Thread is created with the Name : Thread-1
Thread is created with the Name : Thread-4
Thread is created with the Name : Thread-2
Thread is created with the Name : Thread-0
Thread is created with the Name : Thread-3
```

Note : Output varies everytime.

2. By implementing Runnable Interface :

```
class A implements Runnable
{
        public void run()
        {
                System.out.println("Thread is created with the Name : "+Thread.currentThread().getName());
        }
}
class CreatingMultipleThreads2
{
        public static void main(String args[ ])
        {
                A a=new A();
```

```
            Thread t1=new Thread(a);
            Thread t2=new Thread(a);
            Thread t3=new Thread(a);
            Thread t4=new Thread(a);
            Thread t5=new Thread(a);
            t1.start();
            t2.start();
            t3.start();
            t4.start();
            t5.start();
        }
}
```

Output :

```
Thread is created with the Name : Thread-2
Thread is created with the Name : Thread-0
Thread is created with the Name : Thread-1
Thread is created with the Name : Thread-4
Thread is created with the Name : Thread-3
```

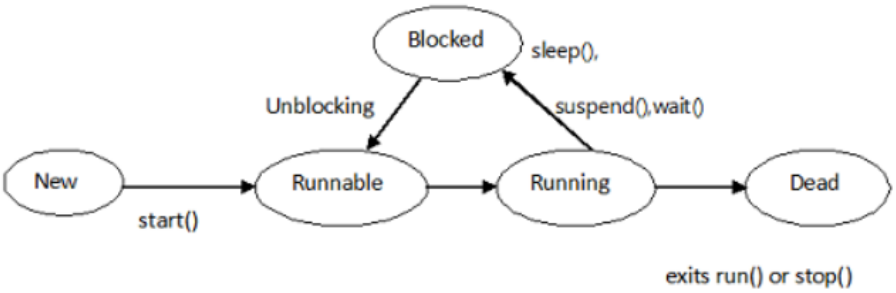
Note : Output varies everytime.

5.12 LIFE CYCLE OF A THREAD

-For Video

Stages of a thread :

1. New
2. Runnable
3. Running
4. Blocked(Non-Runnable)
5. Dead



1. **New**: A new thread is created but not working. A thread after creation and before invocation of start() method will be in new state.
2. **Runnable :** A thread after invocation of start() method will be in runnable state. A thread in runnable state will be available for thread scheduler.
3. **Running :** A thread in execution after thread scheduler select it, it will be in running state.

4. **Blocked :** A thread which is alive but not in runnable or running state will be in blocked state. A thread can be in blocked state because of suspend(), sleep(), wait() methods or implicitly by JVM to perform I/O operations.

5. **Dead :** A thread after exiting from run() method will be in dead state. We can use stop() method to forcefully killed a thread.

5.13 PROGRAMS ON THREAD METHODS -For Video

Example program for getName and setName Methods :

```
class Hello extends Thread
{
        public void run()
        {
                for(int i=0;i<5;i++)
                {
                        System.out.println("Child Thread");
                }
        }
}
class ThreadMethods
{
        public static void main(String args[ ])
        {
               System.out.println("Name of present executing thread : "+Thread.currentThread().getName());
                Hello h=new Hello();
                System.out.println("Name of present executing thread : "+h.getName());
                Thread.currentThread().setName("Vedavyas");
               System.out.println("Name of present executing thread : "+Thread.currentThread().getName());
                System.out.println(10/0);
                }
}
```

Example :

```
Name of present executing thread : main
Name of present executing thread : Thread-0
Name of present executing thread : Vedavyas
Exception in thread "Vedavyas" java.lang.ArithmeticException: / by zero
```

Program on yield Method :

```
class Hello extends Thread
{
        public void run()
        {
                for(int i=0;i<3;i++)
                {
                        System.out.println("Child Thread");
                        Thread.yield();
                }
        }
}
```

```
class YieldMethod
{
	public static void main(String args[ ])
	{
		Hello h=new Hello();
		h.start();
		for(int i=0;i<3;i++)
		{
			System.out.println("Main Thread");
		}
	}
}
```

Output :

```
Main Thread
Main Thread
Child Thread
Main Thread
Child Thread
Child Thread
```

Note : Must ends with child class.

Program on join Method :

```
class Hello extends Thread
{
	public void run()
	{
		for(int i=0;i<3;i++)
		{
			System.out.println("Child Thread");
		}
	}
}
class JoinMethod
{
	public static void main(String args[ ]) throws Exception
	{
		Hello h=new Hello();
		h.start();
		h.join();
		for(int i=0;i<3;i++)
		{
			System.out.println("Main Thread");
		}
	}
}
```

Output :

```
Child Thread
Child Thread
Child Thread
Main Thread
```

Main Thread
Main Thread

Program on sleep Method :

```
class Hello extends Thread
{
        public void run()
        {
                for(int i=0;i<5;i++)
                {
                        System.out.println("Child Thread");
                        try
                        {
                                Thread.sleep(3000);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }
        }
}
class SleepMethod
{
        public static void main(String args[]) throws Exception
        {
                Hello h=new Hello();
                h.start();
                for(int i=0;i<5;i++)
                {
                        System.out.println("Main Thread");
                }
        }
}
```

Output :

```
Main Thread
Main Thread
Main Thread
Main Thread
Main Thread
Child Thread
Child Thread
Child Thread
Child Thread
Child Thread
```

5.14 THREAD PRIORITIES

-For Video

- Java works within a multithreading environment in which thread scheduler assigns the processor to a thread based on the priority of thread.
- Whenever we create a thread in Java, it always has some priority assigned to it.
- Priority can either be given by JVM while creating the thread or it can be given by the programmer explicitly.
- Priorities in threads is a concept where each thread is having a priority which in layman's language one can say every object is having priority here which is represented by numbers ranging from 1 to 10.
  => The default priority is set to 5 as excepted.
  => Minimum priority is set to 1.
  => Maximum priority is set to 10.
- Here 3 constants are defined in it namely as follows:
  1. **public static int NORM_PRIORITY**
  2. **public static int MIN_PRIORITY**
  3. **public static int MAX_PRIORITY**
- Let us do discuss how to get and set priority of a thread in java.
  1. **public final int getPriority() :** It returns the priority of the thread.
  2. **public final void setPriority(int newPriority) :** It changes the priority of thread to the value newPriority. This method throws IllegalArgumentException if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.
- **Example :**

```
class Hello extends Thread
{
        public void run()
        {
                System.out.println("Child Thread");
        }
}
class ThreadPriority
{
        public static void main(String args[ ])
        {
                System.out.println("Priority of present main thread : "+Thread.currentThread().getPriority());
                Hello h=new Hello();
                System.out.println("Priority of present child thread : "+h.getPriority());
                h.setPriority(6);
                System.out.println("Priority of present child thread : "+h.getPriority());
        }
}
```

Output :

```
Priority of present main thread : 5          //(By default)
Priority of present child thread : 5         // (Inherited from parent thread)
Priority of present child thread : 6
```

5.15 DAEMON THREAD

-For Video

- Daemon thread is a low-priority thread that runs in the background to perform tasks such as garbage collection.
- Daemon thread is also a service provider thread that provides services to the user thread.
- When all the user threads die, JVM terminates this thread automatically.
- Simply, it provides services to user threads for background supporting tasks. It has no role in life other than to serve user threads.

Properties of Daemon Thread :

- They can not prevent the JVM from exiting when all the user threads finish their execution.
- JVM terminates itself when all user threads finish their execution.
- If JVM finds a running daemon thread, it terminates the thread and, after that, shutdown it. JVM does not care whether the Daemon thread is running or not.

Default Nature of Daemon Thread :

- By default, the main thread is always non-daemon but for all the remaining threads, daemon nature will be inherited from parent to child. That is, if the parent is Daemon, the child is also a Daemon and if the parent is a non-daemon, then the child is also a non-daemon.

Methods of Daemon Thread :

1. **void setDaemon(boolean status) :** This method marks the current thread as a daemon thread or user thread.
2. **boolean isDaemon() :** It is used to check that current thread is a daemon. It returns true if the thread is Daemon. Else, it returns false.

Example :

```
class Hello extends Thread
{
        public void run()
        {
                for(int i=0;i<5;i++)
                {
                        System.out.println("Child Thread");
                }
        }
}
class DaemonThread
{
        public static void main(String args[ ]) throws Exception
        {
                System.out.println(Thread.currentThread().isDaemon());          //false
                Hello h=new Hello();
                System.out.println(h.isDaemon());       //false
                h.setDaemon(true);
                System.out.println(h.isDaemon());       //true
        }
}
```

5.16 SYNCHRONIZATION -For Video

- It is used to make sure by some synchronization method that only one thread can access the resource at a given point in time.
- This synchronization is implemented in Java with a concept called monitors or locks.
- Only one thread can own a monitor at a given time.
- When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.
- We cannot apply **synchronized** keyword with the variables and classes.
- Java programming language provide two synchronization idioms:
  1. Methods synchronization
  2. Statement(s) synchronization (Block synchronization)

1. Methods synchronization

- If a particular thread 't1' wants to execute a synchronized method on the given object then first it has to acquire the lock of that object.
- Once it acquired the lock of object then we can able to allow to execute synchronized method.
- **Example :**

```
class Display
{
        public synchronized void greet(String name)
        {
                for(int i=0;i<3;i++)
                {
                        System.out.print("Hello...");
                        try
                        {
                                Thread.sleep(2000);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                        System.out.println(name);
                }
        }
}
class Assign extends Thread
{
        Display d;
        String name;
        Assign(Display d, String name)
        {
                this.d=d;
                this.name=name;
```

```
        }
        public void run()
        {
                d.greet(name);
        }
}
class SynchronizationExample
{
        public static void main(String args[ ]) throws Exception
        {
                Display d=new Display(); //object for Display class
                Assign a1= new Assign(d,"Vedavyas"); //1st child thread instantiated
                Assign a2= new Assign(d,"Mahesh"); //2nd child thread instantiated
                a1.start(); //1st child thread created, it contains 2 threads(main, c1)
                a2.start(); //2nd child thread created, it contains 3 threads(main, c1, c2)
        }
}
```

Types of locks :

- When thread enters into synchronized instance method or block, it acquires Object level lock and when it enters into synchronized static method or block it acquires class level lock.
- **Locking Current Object :**

```
synchronized(this)
{
        //Block of statements
}
```

- **Class level Lock :**

```
synchronized(ClassName.class)
{
        //Block of statements
}
```

- **Lock of particular Object :**

```
synchronized(object)
{
        //Block of statements
}
```

2. Block Synchronization

- If we only need to execute some subsequent lines of code but not all lines of code within a method, then we should synchronize only block of the code within which required instructions are exists.
- lets suppose there is a method that contains 1000 lines of code but there are only 50 lines (one after one) of code which contain critical section of code i.e. these lines can modify (change) the Object's state. So we only need to synchronize these 50 lines of code method to avoid any modification in state of the Object and to ensure that other threads can execute rest of the lines within the same method without any interruption.

- **Example**

```
class Display
{
        public void greet(String name)
        {
                synchronized(Display.class)
                {
                        System.out.print("Hello...");
                        try
                        {
                                Thread.sleep(2000);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                        System.out.println(name);
                }
        }
}
class Assign extends Thread
{
        Display d;
        String name;
        Assign(Display d, String name)
        {
                this.d=d;
                this.name=name;
        }
        public void run()
        {
                d.greet(name);
        }
}
class SynchronizedBlocks
{
        public static void main(String args[ ]) throws Exception
        {
                Display d=new Display();
                Assign a1= new Assign(d,"Vedavyas");
                Assign a2= new Assign(d,"Mahesh");
                a1.start();
                a2.start();
        }
}
```

Output :

```
Hello...Vedavyas
Hello...Mahesh
```

5.17 INTERTHREAD COMMUNICATION -For Video

- It is defined as allowing synchronized threads to communicate with each other.
- It is also defined as the cooperation between various threads in the critical section.
- **Critical Section :** It is a block of code that accesses a shared resource and can be executed by only one thread at a time.
- **Methods :**
  1. **notify( )** : It is used to notify only one waiting thread.
  2. **notifyAll( )** : It is used to notify the all waiting threads.
  3. **wait( )** : It is used to release the lock and wait for another thread until it get information.
- **Prototypes of above methods :**
  1. public final void wait( )
  2. public final native void wait(long millisec)
  3. public final void wait(long millisec, int nanosec)
  4. public final ntive void notify( )
  5. public final native void notifyAll( )
- **Note :** wait( ) method throws InterruptedException.
- **Example :**

```
class Hello extends Thread {
        int total=0;
        public void run( )
        {
                synchronized(this)
                {
                        for(int i=0;i<=10;i++)
                        {
                                total=total+i;
                        }
                        this.notify( );
                }
        }
        // 1 thousand lines of code
}
class InterThreadCommunication {
        public static void main(String args[ ]) throws Exception {
                Hello h=new Hello( );
                h.start( );
                synchronized(h)
                {
                        h.wait( );
                        System.out.println("The Sum is "+h.total);
                }
        }
}
```

Output : The Sum is 55

5.18 DEAD LOCKS

-For Video

- When a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Here both threads are waiting for each other to release the lock, the condition is called deadlock.
- In Deadlock condition two or more threads will be blocked forever and waiting for each other.
- Deadlock condition will arises due to developer mistakes or less resources.
- Deadlock condition is a complex condition which occurs only in case of multiple threads.
- Deadlock condition can break our code at run time and can destroy business logic.

Figure - 1

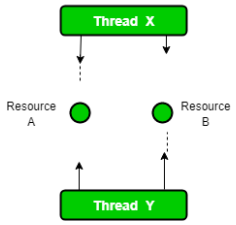
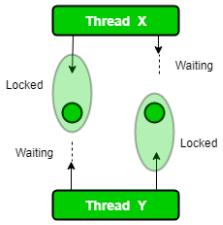


Figure - 2



- **Example :**

```
class A
{
        public synchronized void afirst(B b)
        {
                System.out.println("Thread starts execution of afirst() in A class");
                try
                {
                        Thread.sleep(3000);
                }
                catch(Exception ee)
                {
                        System.out.println(ee);
                }
                b.last();
                System.out.println("Thread1 trying to call last() of B class");
        }
        public synchronized void last()
        {
                System.out.println("Last method of A class");
        }
}
class B
{
        public synchronized void bfirst(A a)
        {
                System.out.println("Thread starts execution of bfirst() in B class");
                try
```

```
                {
                        Thread.sleep(3000);
                }
                catch(Exception ee)
                {
                        System.out.println(ee);
                }
                a.last();
                System.out.println("Thread1 trying to call last() of A class");
        }
        public synchronized void last()
        {
                System.out.println("Last method of B class");
        }
}
class DeadLocks extends Thread
{
        A a=new A();
        B b=new B();
        public void vv()
        {
                this.start();
                a.afirst(b);
        }
        public void run()
        {
                b.bfirst(a);
        }
        public static void main(String args[ ]) throws Exception
        {
                DeadLocks d=new DeadLocks();
                d.vv();
        }
}
```

Output :

```
Thread starts execution of afirst() in A class
Thread starts execution of bfirst() in B class
```

5.19 THREAD GROUPS

[-For Video]

- Grouping of similar kind of threads is called a ThreadGroup.
- It is a predefined class in java.lang package
- In java, default thread is **main** and default thread group is **system**.
- We can able to apply various kinds of methods for a thread group.

Constructors :

1. **public ThreadGroup(String name) :** Constructs a new thread group. The parent of this new group is the thread group of the currently running thread.
2. **public ThreadGroup(ThreadGroup parent, String name) :** Creates a new thread group. The parent of this new group is the specified thread group.

Methods :

1. **getThreadGroup() :** It is used to get the reference of the current executable threadgroup.
2. **getName() :** It is used to get the name of the current threadgroup.
3. **getParent() :** It is used to get reference of the current executable threadgroup parent.
4. **list() :** It is used to display what are the threads available in the threadgroup.
5. **activeCount() :** It is used to show no of activ threads present in the threadgroup.
6. **enumerate() :** It is used to copy all the threads which are present in threadgroup.
7. **isDaemon() :** It is used to check whether the thread is daemon or not.
8. **setDaemon() :** It is used to a thread set as Daemon.
9. **interrupt() :** It is used to interrupt the threadgroup.
10. **destroy() :** It is used to destroy the thread group and any child groups on which it is called.

Example :

```
class Cse implements Runnable
{
        public synchronized void run()
        {
                for(int i=0;i<1;i++)
                {
                        System.out.println("Thread is created with the Name : "+Thread.currentThread().getName());
                }
                try
                {
                        Thread.sleep(2000);
                }
                catch(Exception e)
                {
                        System.out.println("Interruted Exception raised");
                }
        }
}
class TGExample
{
        public static void main(String args[ ]) throws Exception
        {
                System.out.println("Default Parent ThreadGroup : "+Thread.currentThread().getThreadGroup()
.getParent().getName());
                Cse a=new Cse();
                ThreadGroup tg1=new ThreadGroup("FirstTG");
```

```
                System.out.println("ThreadGroup1 is created with the Name : "+tg1.getName());
                System.out.println("Parent of ThreadGroup1 : "+tg1.getParent().getName());
                ThreadGroup tg2=new ThreadGroup("SecondTG");
                System.out.println("ThreadGroup2 is created with the Name : "+tg2.getName());
                System.out.println("Parent of ThreadGroup2 : "+tg1.getParent().getName());
                Thread t1=new Thread(tg1,a,"T1");
                Thread t2=new Thread(tg2,a,"T2");
                Thread t3=new Thread(tg1,a,"T3");
                Thread t4=new Thread(tg2,a,"T4");
                Thread t5=new Thread(tg1,a,"T5");
                t1.start();
                t2.start();
                t3.start();
                t4.start();
                t5.start();
                int count1=tg1.activeCount();
                System.out.println("No of threads ThreadGroup1 : "+count1); tg1.list();
                int count2=tg2.activeCount();
                System.out.println("No of threads ThreadGroup2 : "+count2); tg2.list();
                Thread[ ] ts=new Thread[3];
                tg1.enumerate(ts);
                for(int i=0;i<ts.length;i++)
                {
                        System.out.println((i+1)+"-Thread Name : "+ts[i].getName());
                }
        }
}
```

Output :

```
Default Parent ThreadGroup : system
ThreadGroup1 is created with the Name : FirstTG
Parent of ThreadGroup1 : main
ThreadGroup2 is created with the Name : SecondTG
Parent of ThreadGroup2 : main
Thread is created with the Name : T1
No of threads ThreadGroup1 : 3
java.lang.ThreadGroup[name=FirstTG,maxpri=10]
Thread[T1,5,FirstTG]
Thread[T3,5,FirstTG]
Thread[T5,5,FirstTG]
No of threads ThreadGroup2 : 2
java.lang.ThreadGroup[name=SecondTG,maxpri=10]
Thread[T2,5,SecondTG]
Thread[T4,5,SecondTG]
1-Thread Name : T1
2-Thread Name : T3
3-Thread Name : T5
Thread is created with the Name : T5
Thread is created with the Name : T4
Thread is created with the Name : T2
Thread is created with the Name : T3
```

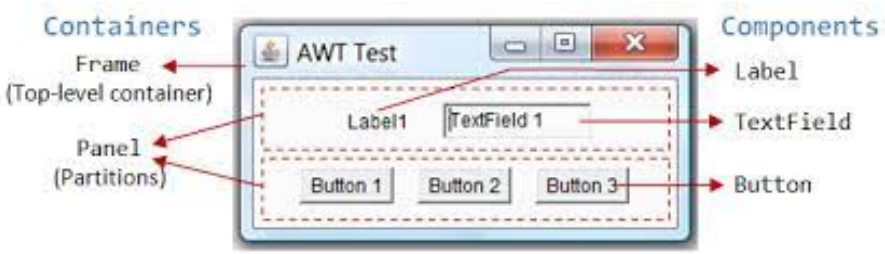
6.1 GRAPHICAL USER INTERFACE -For Video

- GUI is the collection of its various components like buttons, checkboxes, textfields, labels, radiobuttons etc...
- GUI allows the user to interact with the system.
- With the help of GUI, visualization will be more for the user.
- GUI is two types :
  1. Standalone GUI
  2. Distributed GUI
- In java, to implement any GUI application we can use 3 types of packages
  i. AWT [Abstract window toolkit]
  ii. Applets
  iii. Swings
- AWT and Swings are used to develop standalone or desktop GUIs
- Applet is used to develop distributed GUI.

NOTE:

- java.applet package has been deprecated in Java 9 and later versions,as applets are no longer widely used on the web.

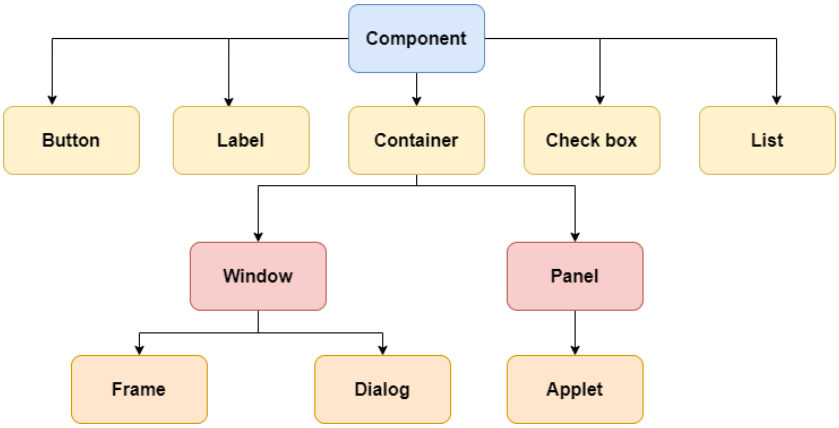
STRUCTURE OF GUI :



6.2 ABSTRACT WINDOW TOOLKIT (AWT) -For Video

- AWT (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.
- AWT is a framework which provides good environment to implement various kind of GUI applications.
- AWT components are platform-dependent i.e. components are displayed according to the view of operating system.
- AWT will have different look and feel for the different platforms like Windows, MAC OS, and Unix.
- In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

AWT HIERARCHY :



COMPONENT :

- A Component is simply an user interface object.
- Every component have some properties, methods and events.
- All the elements like the button, text fields, scroll bars, etc. are called components.
- In Java AWT, there are classes for each component as shown in above diagram.
- In order to place every component in a particular position on a screen, we need to add them to a container.
- **Methods :**
  1. **public void add(Component c) :** Inserts a component on this component.
  2. **public void setSize(int width,int height) :** Sets the size (width and height) of the component.
  3. **public void setLayout(LayoutManager m) :** Defines the layout manager for the component.
  4. **public void setVisible(boolean status) :** Changes the visibility of the component, by default false.
  5. **public void setTitle(String Title) :** Defines the title to appear at the top of the window.
  6. **public void setBackground(Color.colorname) :** Used to set the background color.
  7. **public void setForeground(Color.colorname) :** Used to set the foreground colour i.e the colour in which text is shown.
  8. **public void paint(Graphics g) :** It is used to display the data on the screen. This paint( ) method will internally calls **drawString( )** method.
  9. **public abstract void drawString(String str, int x, int y):** Used to draw the specified Shape.

CONTAINER :

- The Container is a component in AWT that can contain another components like buttons, textfields, labels etc.
- A Container is a class which is going to add 'N' number of GUI components to the screen.
- It is simply a displaying area.
- Every container is associated with layout mechanics of all the components.
- A container itself is a component (see the above diagram), therefore we can add a container inside container.

TYPES OF CONTAINERS :

- There are four types of containers in Java AWT :
  1. Window
  2. Panel
  3. Dialog
  4. Frame

1. WINDOW :

- The window is the container that have no borders and menu bars.
- It is used to place various kind of components in it.
- We must use frame, dialog or another window for creating a window.
- We need to create an instance of Window class to create this container.

2. PANEL :

- The Panel is the container that doesn't contain title bar, border or menu bar.
- It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

3. DIALOG :

- The Dialog represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class.
- Frame and Dialog both inherits Window class. Frame has maximize and minimize buttons but Dialog doesn't have those.

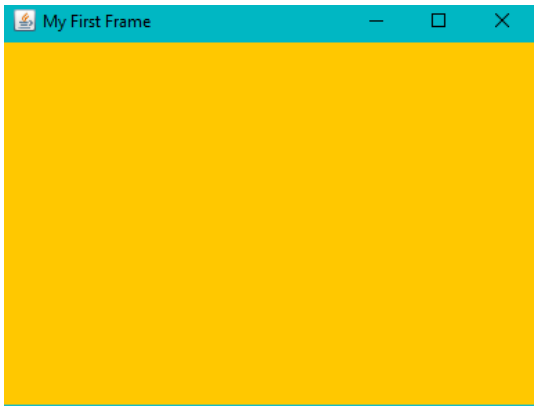
4. FRAME : **-For Video**

- The Frame is the container that contain title bar and border and can have menu bars.
- It can have other components like button, text field, scrollbar etc.
- Frame is most widely used container while developing an AWT application.
- A Frame is a collection of panels and it is a subclass of Window.
- To create a frame we need to **import java.awt.Frame**
- **Constructors :**
  1. public Frame();
  2. public Frame(String title);
- **Creation of Frames :**
  We can create Frames in two ways :
  1. By creating instance of Frame class
  2. By extending the Frame class.
- **1. By creating instance of Frame class :**

```
import java.awt.*;
class FrameByCreatingInstanceOfFrameClass
{
        public static void main(String args[ ])
        {
                Frame f=new Frame();
                f.setTitle("My First Frame");
                f.setVisible(true);
```

```
        f.setSize(400,300);
        f.setBackground(Color.orange);
    }
}
```

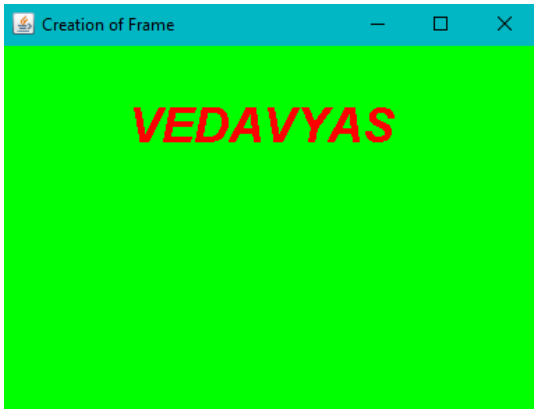
Output :



- **2. By extending the Frame class :**

```
import java.awt.*;
class MyFrame extends Frame
{
    MyFrame()
        {
        setVisible(true);
        setSize(400,300);
        setTitle("Creation of Frame");
        setBackground(Color.green);
    }
    public void paint(Graphics g)
    {
        Font f=new Font("ARIAL", Font.BOLD+Font.ITALIC,35); g.setFont(f);
        setForeground(Color.red);
        g.drawString("VEDAVYAS",100,100);
    }
}
class FrameByExtendingFrameClass
{
    public static void main(String args[ ])
    {
        MyFrame m=new MyFrame();
    }
}
```

Output :



6.3 EVENT HANDLING -For Video

- An **event** can be defined as changing the state of an object or behavior by performing actions.
- Actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc.
- By default, all the active components don't perform any type of actions.
- The **java.awt.event** package can be used to provide various event classes to perform actions.
- Whenever we click on the active components then object will be created for respective active component which contains **Name** and **Reference of the component.**
- Events are generated from the sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, windows, etc.
- **Listeners** are used for handling the events generated from the source. Each of these listeners are interfaces.
- To perform Event Handling, we need to register the source with the listener.
- Different Classes provide different registration methods.

Event Class	Listener Interface	Description
ActionEvent	ActionListener	Performing actions like button click, selecting an item from menu.
AdjustmentEvent	AdjustmentListener	This event is emitted by an Adjustable object like Scrollbar.
ComponentEvent	ComponentListener	Involving in moving the component,changing size or its visibility.
ContainerEvent	ContainerListener	Event of adding a component to a container (or) removed from it.
ItemEvent	ItemListener	Event that indicates whether an item was selected or not.
KeyEvent	KeyListener	Event that occurs due to a sequence of keypresses on keyboard.
MouseEvent	MouseListener	Events that occur due to user interaction with the mouse .
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	It indicates whether a window has changed its status or not.

- Different Listener interfaces consists of different methods which are specified below.

Listener Interface	Methods
ActionListener	actionPerformed()
AdjustmentListener	adjustmentValueChanged()
ComponentListener	componentResized() componentShown() componentMoved() componentHidden()
ContainerListener	componentAdded() componentRemoved()
ItemListener	itemStateChanged()
KeyListener	keyTyped() keyPressed() keyReleased()
MouseListener	mousePressed() mouseClicked() mouseEntered() mouseExited() mouseReleased()
TextListener	textChanged()
WindowListener	windowActivated() windowDeactivated() windowIconified() windowDeiconified() windowOpened() windowClosed() windowClosing()

- **Flow of Event Handling :**
  1. User Interaction with a component is required to generate an event.
  2. The object of the respective event class is created automatically after event generation, and it holds all information of the event source.
  3. The newly created object is passed to the methods of the registered listener.
  4. The method executes and returns the result.
- **Steps to write a program :**
  1. Create a class with the extension of Frame class.
  2. Select a required Listener and implement it by overriding the abstract methods.
     Note : We can combine both step-1 and step-2.
  3. Attach Listener class to GUI component by using respective addListener method.

Example - WindowListener

```
import java.awt.*;
import java.awt.event.*;
class Window1 implements WindowListener
{
        public void windowOpened(WindowEvent we)
        {
                System.out.println("Window Opened");
        }
        public void windowActivated(WindowEvent we)
        {
```

```
            System.out.println("Window Activated");
        }
        public void windowDeactivated(WindowEvent we)
        {
            System.out.println("Window Deactivated");
        }
        public void windowIconified(WindowEvent we)
        {
            System.out.println("Normal state to minimize state");
        }
        public void windowDeiconified(WindowEvent we)
        {
            System.out.println("From minimize state to normal state");
        }
        public void windowClosing(WindowEvent we)
        {
            System.out.println("Window closed");
            System.exit(0);
        }
        public void windowClosed(WindowEvent we)
        {
            System.out.println("Window closed from window closed method");
        }
}
class EventExample extends Frame
{
        EventExample()
        {
            setTitle("EventExample");
            setVisible(true);
            setSize(500,500);
            setBackground(Color.green);
            addWindowListener(new Window1());
        }
}
class EventHandlingExample
{
        public static void main(String args[ ])
        {
            EventExample obj=new EventExample();
        }
}
```

6.4 ADAPTER CLASSES -For Video

- It is a kind of class which contains Null body definition for all the methods which are inheriting from the appropriate Listener.
- If a Listener having more than 1 method and we didn't require override all the methods then we can go for **Adapter classes.**
- For each and every Listener there exists separate adapter class.
- Let us see some of them.

Predefined Interface	Adapter class
ActiveListener	WindowAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMonitorAdapter
WindowListener	WindowAdapter

Example :

```
import java.awt.*;
import java.awt.event.*;
class EventEx extends Frame
{
        EventEx()
        {
                setTitle("Frame for AdapterClass Example");
                setVisible(true);
                setSize(400,400);
                setBackground(Color.yellow);
                addWindowListener(new WindowAdapter()
                {
                        public void windowIconified(WindowEvent we)
                        {
                                System.out.println("Normal size to minimize");
                        }
                        public void windowClosing(WindowEvent we)
                        {
                                System.out.println("Window closed");
                                System.exit(0);
                        }
                });
        }
}
class AdapterClassExample
{
        public static void main(String args[ ])
        {
                EventEx obj=new EventEx();
        }
}
```

6.5 AWT COMPONENTS -For Video

- There are different components available in the Java AWT package for developing user interfaces.
- Some of them are as follows :
  1. Label
  2. Button
  3. TextField
  4. TextArea
  5. Choice
  6. List
  7. CheckBox
  8. RadioButtons
  9. ScrollBar

1. Label :

- It is a predefined class used to create labels [Text Information].
- Label is a passive component. It never perform any type of action.
- With the help of constructors, we can create a label.
- With the help of Data members, we can able to provide various kind of properties.
- **Constructors :**
  1. Label( ) : Creates a blank label [Not visible].
  2. Label(String)
  3. Label(String, Alignment)
- **Methods :**
  1. setText(String)
  2. getText( );
  3. setAlignment( );
  4. getAlignment( );
  5. setBounds( );
- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponents extends Frame
{
        AWTComponents()
        {
                setTitle("Label Example");
                setSize(500,400);
                setVisible(true);
                setBackground(Color.cyan);
                Label l=new Label();
                l.setText("Enter Name : ");
                l.setAlignment(Label.LEFT);
                l.setBounds(50,50,100,100);
                add(l);
                Label l1=new Label("Enter ID : ", Label.CENTER);
```

```
                add(l1);
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                }
                );
        }
}
class LabelExample
{
        public static void main(String args[ ])
        {
                AWTComponents obj=new AWTComponents();
        }
}
```

Output :

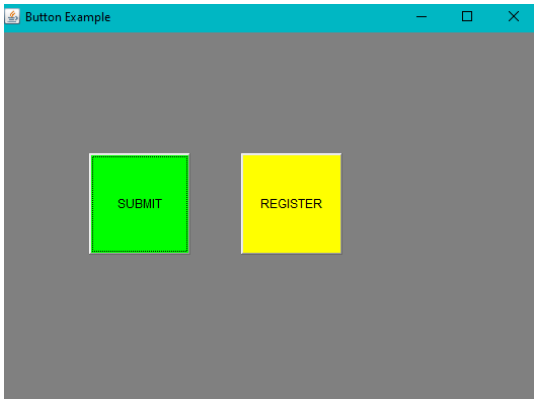
2. Button :

- A button is basically a control component with a label that generates an event when clicked.
- The Button class is used to create a labelled button that has platform independent implementation.
- **Constructors :**
  1. Button()
  2. Button(String Text)
- **Some of Methods :**
  1. setLabel(String)
  2. getLabel()
  3. setText(String)
  4. getText()

- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponents extends Frame {
        AWTComponents() {
                setTitle("Button Example");
                setSize(500,400);
                setVisible(true);
                setBackground(Color.gray);
                Button b1=new Button();
                b1.setLabel("SUBMIT");
                b1.setBounds(100,150,100,100);
                b1.setBackground(Color.green);
                Button b2=new Button("REGISTER");
                b2.setBounds(250,150,100,100);
                b2.setBackground(Color.red);
                Label l=new Label();
                add(b1);
                add(b2);
                add(l);
                addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent we) {
                                System.exit(0);
                        }
                }
                );
        }
}
class ButtonExample
{
        public static void main(String args[ ]) {
                AWTComponents obj=new AWTComponents();
        }
}
```

Output :



3. TextField :

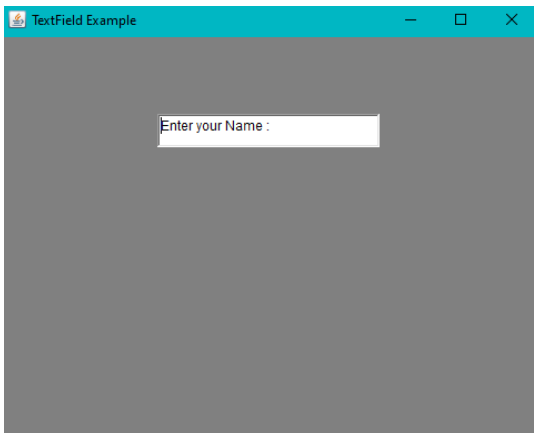
- TextField class is a text component that allows to enter a single line text and edit it.
- It inherits TextComponent class, which further inherits Component class.
- **Constructors :**
  1. **TextField() :** It constructs a new text field component.
  2. **TextField(String text) :** It constructs a new text field initialized with the given string text to be displayed.
  3. **TextField(int columns) :** It constructs a new textfield with given number of columns.
  4. **TextField(String text, int columns) :** It constructs a new text field with the given text and given number of columns (width).
- **Some of methods :**
  1. getEditable()
  2. setEditable()
  3. getText()
  4. setText()
  5. **setEchoChar() :** It sets the echo character for text field. [Passwords]

- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponents extends Frame
{
        AWTComponents()
        {
                setTitle("TextField Example");
                setSize(500,400);
                setVisible(true);
                setBackground(Color.gray);
                setLayout(null);
                TextField t1=new TextField("Enter your Name : ");
                t1.setBounds(150, 100, 200, 30);
                add(t1);
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                }
                );
        }
}
class TextFieldExample
{
        public static void main(String args[ ])
```

```
        {
                AWTComponents obj=new AWTComponents();
        }
}
```

Output :



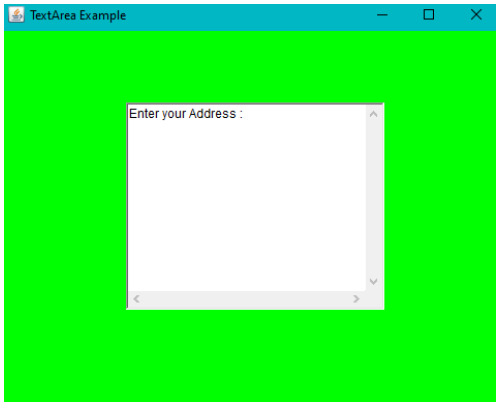
4. TextArea :

- It is used to enter the large amount of text.
- In TextField we can enter the text in single line only whereas the text area allows us to type as much text as we want.
- When the text in the text area becomes larger than the viewable area, the scroll bar appears automatically which helps us to scroll the text up and down, or right and left.
- **Fields of TextArea Class :** The fields of java.awt.TextArea class are as follows:
  1. **static int SCROLLBARS_BOTH :** It creates and displays both horizontal and vertical scrollbars.
  2. **static int SCROLLBARS_HORIZONTAL_ONLY :** It creates and displays only the horizontal scrollbar.
  3. **static int SCROLLBARS_VERTICAL_ONLY :** It creates and displays only the vertical scrollbar.
  4. **static int SCROLLBARS_NONE :** It doesn't create or display any scrollbar in the text area.
- **Constructors :**
  1. **TextArea() :** It constructs a new text area component.
  2. **TextArea(String text) :** It constructs a new text area initialized with the given string text to be displayed.
  3. **TextArea(int rows, int columns) :** It constructs a new text area with specified number of rows and columns and empty string as text.
  4. **TextArea(String text, int rows, int columns) :** It constructs a new text area with given number of columns and rows along string.
  5. **TextArea (String text, int row, int column, int scrollbars) :** It constructs a new text area with specified text in text area and specified number of rows and columns and visibility.

- **Some of methods :**
  1. **void setColumns(int columns) :** It sets the number of columns for this text area.
  2. **void setRows(int rows) :** It sets the number of rows for this text area.
  3. **int getRows() :** It returns the number of rows of text area.
  4. **int getColumns():** It returns the number of columns of text area.
- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponents extends Frame
{
        AWTComponents() {
                setTitle("TextArea Example");
                setSize(500,400);
                setVisible(true);
                setBackground(Color.green);
                setLayout(null);
                TextArea t1=new TextArea("Enter your Address : ");
                t1.setBounds(130, 100, 250,200);
                add(t1);
                addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent we) {
                                System.exit(0);
                        }
                }
                );
        }
}
class TextAreaExample
{
        public static void main(String args[ ]) {
                AWTComponents obj=new AWTComponents();
        }
}
```

Output :

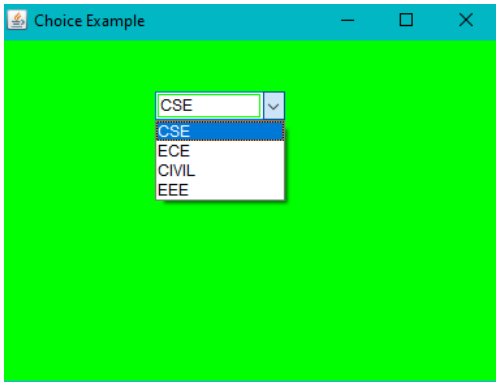


5. Choice :

- It creates a drop down list, where the list contains all the data in the form of strings.
- Out of all, we can select only one item at a time.
- **Constructor :** public Choice();
- **Some of methods :**
  1. **add (String ItemName):** It adds an item to the menu.
  2. **remove (String ItemName) :** It removes the item from the menu.
  3. **insert(String ItemName, int position) :** It adds an item to the menu at specified position.
  4. **getItemCount():** It returns the number of items present in the menu.
  5. **getItem(int position) :** It return items present at the specified position.
- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponents extends Frame
{
        AWTComponents()
        {
                setTitle("Choice Example");
                setSize(400,300);
                setVisible(true);
                setBackground(Color.green);
                setLayout(null);
                Choice c=new Choice();
                c.add("CSE");
                c.add("ECE");
                c.add("CIVIL");
                c.add("EEE");
                c.setBounds(130,70,100,100);
                add(c);
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                }
                );
        }
}
class ChoiceExample
{
        public static void main(String args[ ]) {
                AWTComponents obj=new AWTComponents();
        }
}
```

Output :



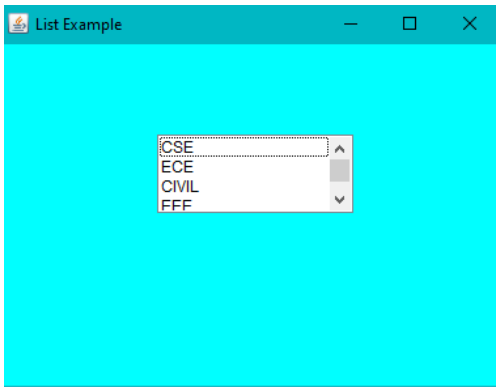
6. List :

- It also provides a drop downlist by scrolling.
- In list, we can able to select multiple items at a time. So it is also called as **MultiChoice**.
- By default it displays four rows only.
- **Constructor :**
  1. public List()
  2. List(int rows)
  3. List(int rows, boolean) : True for multiselection.
- **Some of methods :**
  1. **add (String ItemName):** It adds an item to the menu.
  2. **remove (String ItemName) :** It removes the item from the menu.
  3. **public String[] getItems:** It gives items present in the list.
  4. **getItemCount():** It returns the number of items present in the menu.
  5. **removeAll :** Used to remove the items.
- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponents extends Frame
{
        AWTComponents() {
                setTitle("List Example");
                setSize(400,300);
                setVisible(true);
                setBackground(Color.cyan);
                setLayout(null);
                List c=new List();
                c.add("CSE");
                c.add("ECE");
                c.add("CIVIL");
                c.add("EEE");
                c.setBounds(130,100,150,60);
```

```
                add(c);
                addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent we) {
                                System.exit(0);
                        }
                }
                );
        }
}
class ListExample
{
        public static void main(String args[ ]) {
                AWTComponents obj=new AWTComponents();
        }
}
```

Output :



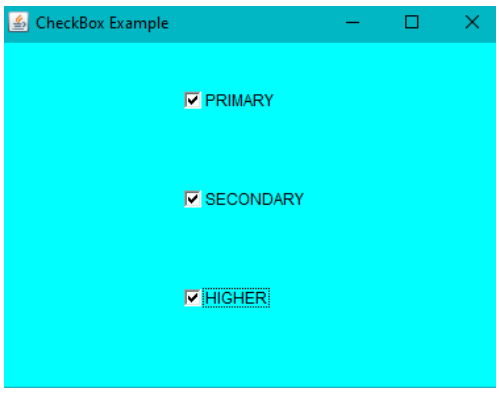
7. CheckBox:

- It is used to get the checked data and to check the multiple data.
- Checkbox control is used to turn an option on(true) or off(false).
- There is label for each checkbox representing what the checkbox does.
- The state of a checkbox can be changed by clicking on it.
- **Constructor :**
  1. Checkbox()
  2. Checkbox(String label)
  3. Checkbox(String label, boolean) : Creates a check box with the specified label and sets the specified state.
- **Some of methods :**
  1. **setLabel() :** Sets this check box's label to be the string argument.
  2. **getLabel() :** Gets the label of this check box.
  3. **setState(boolean) :** Sets the state of this check box to the specified state.
  4. **getState( ):** Determines whether this check box is in the on or off state.

- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponents extends Frame
{
        AWTComponents() {
                setTitle("CheckBox Example");
                setSize(400,300);
                setVisible(true);
                setBackground(Color.cyan);
                setLayout(null);
                Checkbox ch1=new Checkbox("PRIMARY");
                Checkbox ch2=new Checkbox("SECONDARY");
                Checkbox ch3=new Checkbox("HIGHER");
                ch1.setBounds(150,50,100,50);
                ch2.setBounds(150,100,100,100);
                ch3.setBounds(150,150,100,150);
                add(ch1);
                add(ch2);
                add(ch3);
                addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent we) {
                                System.exit(0);
                        }
                }
                );
        }
}
class CheckBoxExample
{
        public static void main(String args[]) {
                AWTComponents obj=new AWTComponents();
        }
}
```

Output :

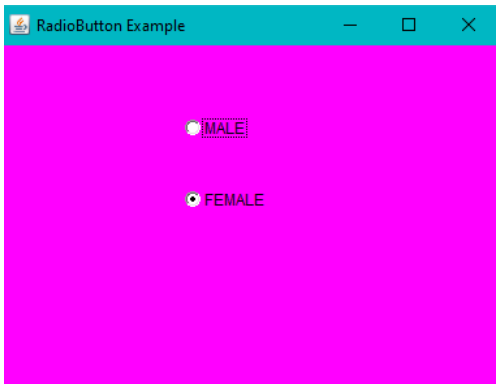


8. RadioButton :

- RadioButton class is used to create a radio button.
- It is used to choose one option from multiple options.
- We need to use checkboxGroup() to create a radiobutton.
- **Constructor :** checkbox(String label, boolean state, checkboxGroup)
- **Some of Methods :**
  void setText(String s) : It is used to set specified text on button.
  String getText() : It is used to return the text of the button.
- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponents extends Frame
{
        AWTComponents()
        {
                setTitle("RadioButton Example");
                setSize(400,300);
                setVisible(true);
                setBackground(Color.magenta);
                setLayout(null);
                CheckboxGroup Gender=new CheckboxGroup();
                Checkbox ch1=new Checkbox("MALE",true,Gender);
                Checkbox ch2=new Checkbox("FEMALE",true,Gender);
                ch1.setBounds(150,70,100,50);
                ch2.setBounds(150,100,100,100);
                add(ch1);
                add(ch2);
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                }
                );
        }
}
class RadioButtonExample
{
        public static void main(String args[ ])
        {
                AWTComponents obj=new AWTComponents();
        }
}
```

Output :



9. ScrollBar :

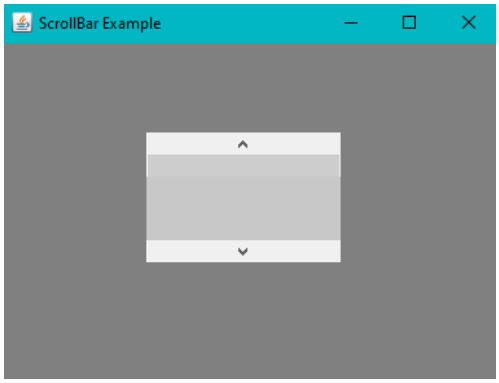
- It is used to add horizontal and vertical scrollbar.
- Scrollbar is a GUI component allows us to see invisible number of rows and columns.
- **Constructor :**
  1. **Scrollbar() :** Constructs a new vertical scroll bar.
  2. **Scrollbar(int orientation) :** Constructs a new scroll bar with the specified orientation.
  3. **Scrollbar(int orientation, int value, int visible, int minimum, int maximum) :** Constructs a new scroll bar with the specified orientation, initial value, visible amount, and minimum and maximum values.
     orientation : specifies whether the scrollbar will be horizontal or vertical.
     Value : specify the starting position of the knob of Scrollbar on its track.
     Minimum : specifies the minimum width of track on which scrollbar is moving.
     Maximum : specifies the maximum width of track on which scrollbar is moving.

- **Some methods :**
  1. **int getMaximum() :** It gets the maximum value of the scroll bar.
  2. **int getMinimum() :** It gets the minimum value of the scroll bar.
  3. **int getOrientation() :** It returns the orientation of scroll bar.
  4. **void setMaximum (int newMaximum) :** It sets the maximum value of the scroll bar.
  5. **void setMinimum (int newMinimum) :** It sets the minimum value of the scroll bar.
  6. **void setOrientation (int orientation) :** It sets the orientation for the scroll bar.
- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponents extends Frame
{
        AWTComponents()
        {
                setTitle("ScrollBar Example");
                setSize(400,300);
                setVisible(true);
```

```
                setBackground(Color.gray);
                setLayout(null);
                Scrollbar sb=new Scrollbar();
                sb.setBounds(120,100,150,100);
                add(sb);
                addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent we) {
                                System.exit(0);
                        }
                }
                );
        }
}
class ScrollBarExample
{
        public static void main(String args[ ])
        {
                AWTComponents obj=new AWTComponents();
        }
}
```

Output :



6.6 LAYOUT MANAGERS -For Video

- LayoutManagers are used to arrange components in a particular manner.
- LayoutManagers are used to control the positioning and size of the components in GUI forms.
- LayoutManager is an interface that is implemented by all the classes of layout managers.
- For every type of container there exists a layout manager. Some of them are as follows :
  1. java.awt.BorderLayout
  2. java.awt.FlowLayout
  3. java.awt.GridLayout
  4. java.awt.GridBagLayout

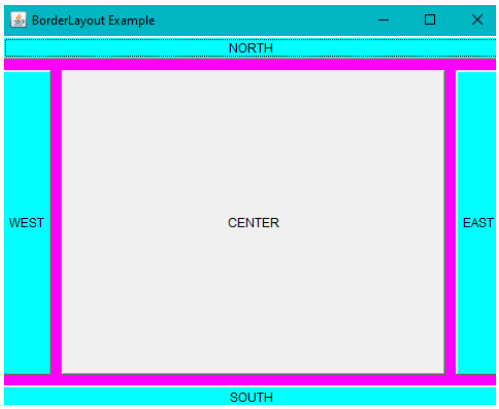
1. **Border Layout :**

- It is a default Layout Manager for Frame or Window.
- In this layout, all the components are going to arrange in a container based on borders of the container.
- BorderLayout is used to arrange the components in five regions.
- The BorderLayout provides five constants for each region:
  1. public static final int NORTH
  2. public static final int SOUTH
  3. public static final int EAST
  4. public static final int WEST
  5. public static final int CENTER
- **Constructors :**
  1. **BorderLayout() :** creates a border layout but with no gaps between the components.
  2. **BorderLayout(int hgap, int vgap) :** creates a border layout with the given horizontal and vertical gaps between the components.
- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponent extends Frame
{
        AWTComponent()
        {
                setTitle("BorderLayout Example");
                setVisible(true);
                setSize(500,400);
                setBackground(Color.magenta);
                BorderLayout b=new BorderLayout(10,10);
                setLayout(b);
                Button b1=new Button("NORTH");
                Button b2=new Button("SOUTH");
                Button b3=new Button("EAST");
                Button b4=new Button("WEST");
                Button b5=new Button("CENTER");
                b1.setBackground(Color.cyan);
                b2.setBackground(Color.cyan);
                b3.setBackground(Color.cyan);
                b4.setBackground(Color.cyan);
                add(b1, BorderLayout.NORTH);
                add(b2, BorderLayout.SOUTH);
                add(b3, BorderLayout.EAST);
                add(b4, BorderLayout.WEST);
                add(b5, BorderLayout.CENTER);
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
```

```
                {
                        System.exit(0);
                }
            }
            );
        }
}
class BorderLayoutExample
{
        public static void main(String args[ ])
        {
            new AWTComponent();
        }
}
```

Output :



2. Flow Layout :

- It is used to arrange the components in a line, one after another (in a flow).
- It is a default Layout manager for **Panel** or **Applet**.
- By default all the components are aligned from left to right.
- **Fields of FlowLayout class :**
  1. public static final int LEFT
  2. public static final int RIGHT
  3. public static final int CENTER
  4. public static final int LEADING
  5. public static final int TRAILING
- **Constructors :**
  1. **FlowLayout() :** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
  2. **FlowLayout(int align) :** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

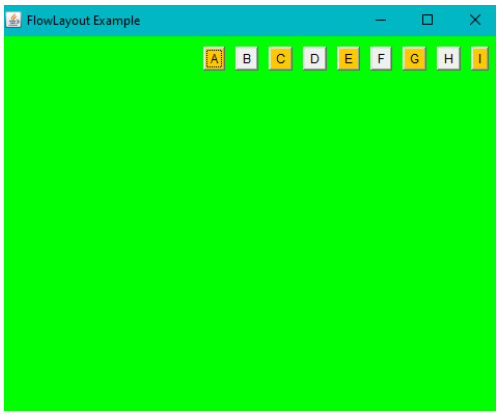
3. **FlowLayout(int align, int hgap, int vgap) :** creates a flow layout with the given alignment and the given horizontal and vertical gap.

- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponent extends Frame
{
        AWTComponent()
        {
                setTitle("FlowLayout Example");
                setVisible(true);
                setSize(500,400);
                setBackground(Color.green);
                FlowLayout f=new FlowLayout(FlowLayout.RIGHT,10,10);
                setLayout(f);
                Button b1=new Button("A");
                Button b2=new Button("B");
                Button b3=new Button("C");
                Button b4=new Button("D");
                Button b5=new Button("E");
                Button b6=new Button("F");
                Button b7=new Button("G");
                Button b8=new Button("H");
                Button b9=new Button("I");
                b1.setBackground(Color.orange);
                b3.setBackground(Color.orange);
                b5.setBackground(Color.orange);
                b7.setBackground(Color.orange);
                b9.setBackground(Color.orange);
                add(b1);
                add(b2);
                add(b3);
                add(b4);
                add(b5);
                add(b6);
                add(b7);
                add(b8);
                add(b9);
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                }
                );
```

```
        }
}
class FlowLayoutExample
{
        public static void main(String args[ ])
        {
                new AWTComponent();
        }
}
```

Output :



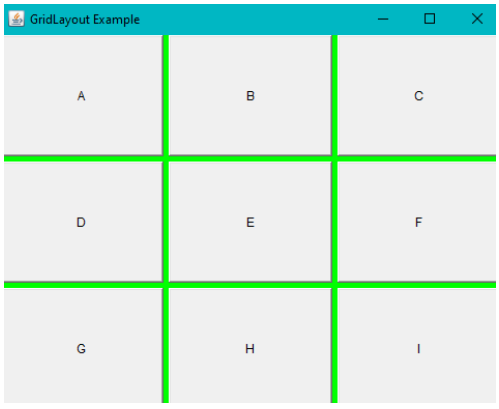
3. Grid Layout :

- It is used to place all the components in the form of rows and columns.
- **Constructors :**
  1. **GridLayout() :** creates a grid layout with one column per component in a row.
  2. **GridLayout(int rows, int columns) :** creates a grid layout with the given rows and columns but no gaps between the components.
  3. **GridLayout(int rows, int columns, int hgap, int vgap) :** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.
- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponent extends Frame
{
        AWTComponent()
        {
                setTitle("GridLayout Example");
                setVisible(true);
                setSize(500,400);
                setBackground(Color.green);
                GridLayout f=new GridLayout(3,3,5,5);
                setLayout(f);
```

```
        Button b1=new Button("A");
        Button b2=new Button("B");
        Button b3=new Button("C");
        Button b4=new Button("D");
        Button b5=new Button("E");
        Button b6=new Button("F");
        Button b7=new Button("G");
        Button b8=new Button("H");
        Button b9=new Button("I");
        add(b1);
        add(b2);
        add(b3);
        add(b4);
        add(b5);
        add(b6);
        add(b7);
        add(b8);
        add(b9);
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        }
        );
    }
}
class GridLayoutExample
{
    public static void main(String args[ ]) {
        new AWTComponent();
    }
}
```

Output :

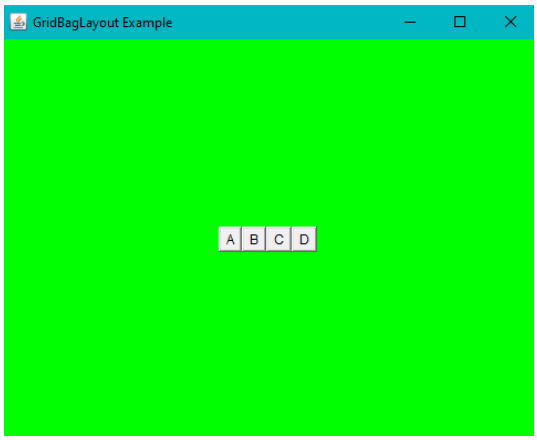


4. GridBag Layout :

- GridBagLayout class is used to align components vertically, horizontally or along their baseline.
- The components may not be of the same size.
- **Constructors :**
  1. **GridBagLayout() :** The parameterless constructor is used to create a grid bag layout manager.
- **Example :**

```
import java.awt.*;
import java.awt.event.*;
class AWTComponent extends Frame
{
        AWTComponent()
        {
                setTitle("GridBagLayout Example");
                setVisible(true);
                setSize(500,400);
                setBackground(Color.green);
                GridBagLayout g=new GridBagLayout();
                setLayout(g);
                Button b1=new Button("A");
                Button b2=new Button("B");
                Button b3=new Button("C");
                Button b4=new Button("D");
                add(b1);
                add(b2);
                add(b3);
                add(b4);
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                }
                );
        }
}
class GridBagLayoutExample
{
        public static void main(String args[ ])
        {
                new AWTComponent();
        }
}
```

Output :



6.7 MENU, MENUBAR AND MENU ITEM -For Video

MenuBar :

- MenuBar class provides menu bar bound to a frame and is platform specific.
- MenuBar is a collection of menus like File, Edit, Image etc.
- **Constructor :** public MenuBar()

Menu :

- The object of Menu class is a pull down menu component which is displayed on the menu bar.
- Menu contains Menu Items and Sub Menus.
- Menu exists if and only if menubar exists.
- **Constructors :**
  1. public Menu() : creates a menu with no label
  2. public Menu(String text) : creates a menu with label specified.
  3. public Menu(Lable, boolean) : True to adjust the menu, by default it is False.

Menu Item :

- The object of MenuItem class adds a simple labeled menu item on menu.
- The items used in a menu must belong to the MenuItem or any of its subclass.
- **Constructors :**
  1. public MenuItem() : creates a menu item with no label
  2. public MenuItem(String text) : creates a menu item with label specified.

CheckBoxMenuItem

- The CheckboxMenuItem class represents a check box which can be included in a menu.
- Selecting the check box in the menu changes control's state from on to off or from off to on.

- **Constructors :**
  1. CheckboxMenuItem() : Create a check box menu item with an empty label.
  2. CheckboxMenuItem(label) : Create a check box menu item with the specified label.
  3. CheckboxMenuItem(label, boolean state) : Create a check box menu item with the specified label and state.

Steps for creation :

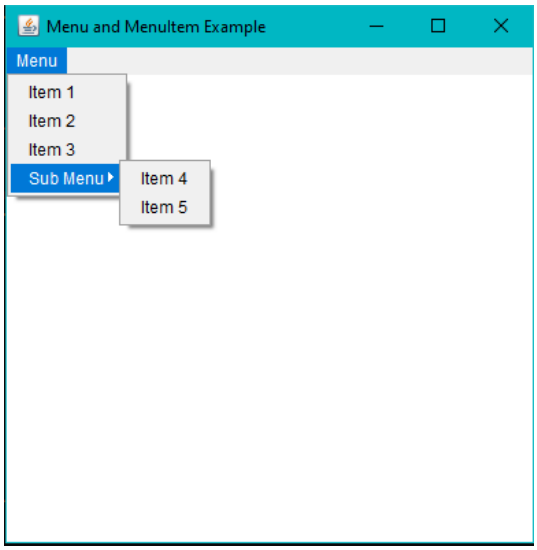
1. Create an object of MenuBar.
2. Create an object of Menu.
3. Create an object of MenuItem.
4. Create an object of Checkbox MenuItem.
5. Add MenuItem to the Menu.
6. Add Menu to the MenuBar.

Example :

```
import java.awt.*;
class MenuExample
{
        MenuExample()
        {
                Frame f= new Frame("Menu and MenuItem Example");
                MenuBar mb=new MenuBar();
                Menu menu=new Menu("Menu");
                Menu submenu=new Menu("Sub Menu");
                MenuItem i1=new MenuItem("Item 1");
                MenuItem i2=new MenuItem("Item 2");
                MenuItem i3=new MenuItem("Item 3");
                MenuItem i4=new MenuItem("Item 4");
                MenuItem i5=new MenuItem("Item 5");
                menu.add(i1);
                menu.add(i2);
                menu.add(i3);
                submenu.add(i4);
                submenu.add(i5);
                menu.add(submenu);
                mb.add(menu);
                f.setMenuBar(mb);
                f.setSize(400,400);
                f.setLayout(null);
                f.setVisible(true);
        }
}
class MenuMenuBarMenuItems
{
        public static void main(String args[ ])
```

```
        {
                new MenuExample();
        }
}
```

Output :



6.8 SWINGS -For Video

- With the help of SWINGS we can able to develop standalone or Desktop or window-based applications.
- All the fetures of AWT are available in SWINGS also and it can have extra features also.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing :

AWT	SWING
AWT components are platform-dependent.	Java swing components are platform-independent.
AWT provides less components than Swing.	It provides more powerful components like tables, lists,etc.
AWT components are heavyweight.	Swing components are lightweight.
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
AWT doesn't follows MVC	Swing follows MVC (Model View Controller).

Note :

1. In MVC model represents data, view represents presentation and controller acts as an interface between model and view.
2. We can execute all the programs of AWT by importing **javax.swing** package and by adding prefix **J** to all the components of AWT.

Creating a JFrame :

```
import javax.swing.JFrame;
class JFrameExample
{
```

```
    public static void main(String args[ ])
    {
        JFrame f=new JFrame();
        f.setTitle("JFrame Creation");
        f.setVisible(true);
        f.setSize(400,300);
    }
}
```

Output :

Note :

- **We can execute all the programs of AWT by importing javax.swing package and by adding prefix J to all the components of AWT.**

****LEARNING NEVER ENDS****