

OOP

Introduction

- ❖ A Software is designed/written to provide a particular functionality. It is a collection of programs.

Software can be divided into two categories :

1. System software
2. Application software

System software is the one that is needed on the hardware using which other softwares are built. **Ex:** Operating system is needed to be installed on the hardware using which other applications are developed, Editors, Compilers.

Algorithm

Algorithm is a step-by-step procedure to solve a problem. Algorithm is a finite set of instructions which if followed a particular task.

An Algorithm is described using a natural language (English) in an unambiguous manner and also in Graphical manner (Flow charts).

I/p, an o/p, definiteness, Finiteness, Effectiveness form the criteria of an algorithm

Each instruction must be clear and unambiguous - **definiteness**.

The algorithm must terminate after a finite no. of steps - **Finiteness**

Every Instruction must be Sufficiently basic and must be feasible - **Effectiveness**

- ★ A program need not satisfy finiteness.

Languages related to computers

Some Programming languages are very close to the computer hardware such as Assembly language, Machine language etc. These languages are known as **low-level languages**.

The problem is it's very difficult for a user to write down a program using a machine code/assembly language.

The user needs to have a complete picture of the hardware and registers to use the machine language and slightly better usage can be facilitated by assembly language as it includes mnemonics here the user needs to have a clear picture of what instructions, what operations must be performed by the hardware.

High level programming language allows specification to a problem solution in terms closer to those used by human beings.

These languages remove the need of knowing the hardware/architecture details for the user.

As the computer doesn't understand the high level language they have to be processed by passing them through a program called **Compiler**.

A compiler translates the high level language into internal machine language before they can be executed.

Pros :

Provides expressiveness, enhances readability, safeguard against the bugs, allows the use of symbolic names for values.

Cons:

A system needs to undergo many changes before execution.

All programs have two important elements: **Code and Data**. The variables declared and the values assigned to them are Data. The computations performed on the data is Code.

Any program observed, it has two things: **Syntax & Semantics**. Syntax cares about whether the code statements are correct or not. Semantics is much beyond that; it considers the correctness, the meaning of the code statement.

Process-oriented Programming

A problem is viewed as a sequence of things to be done.

This can be characterized as code acting data, it is focused on what is being done rather than who is being affected.

The code (is given more importance than data) acts on the data without considering the semantics(meaning- syntactic validation).

- ★ C language falls under a **process-oriented paradigm**. There exist storage classes (auto, static, extern, register) in C language which helps to know more about a variable like it's scope, lifetime, place of storage etc.
- ★ The main difference between **Structure and Union** is that each member of a structure has a separate memory space whereas in union the total memory space allocated is equal to the member with largest size.

The object-oriented paradigm is not to replace the existing process-oriented paradigm rather to complement it.

The process-oriented approach is not natural when we look at the real world scenario. Sometimes complex applications cannot be modeled/implemented in a more effective way for certain domains using process-oriented paradigm, in such cases object-oriented paradigm is used.

Decomposition (Dissolving Software Complexity)

1. Algorithmic Decomposition
2. Object oriented Decomposition (OO Decomposition)

Algorithmic Decomposition is followed in a process-oriented approach whereas an Object oriented Decomposition is followed in an Object-oriented approach.

In Algorithmic Decomposition, functions are invoked in a hierarchical order. Algorithmic Decomposition highlights the ordering of events, it is the process where a bigger task is divided into smaller functions which are focused one at a time. **This can be characterized as code acting data**

In an object-oriented decomposition a bigger task is divided into smaller methods, which are always attached(dependent) to an object/entity. It is more about how objects collaborate to provide functionality. A functionality is invoked only through some data reference hence **this can be characterized as data providing access to code.**

OO Decomposition is best suited for more complex systems.

Basis for modularization (dividing a bigger problem into smaller modules/parts) **in Algorithmic decomposition are functions and in OO decomposition the basis is objects/entities.**

The three fundamental principles of object oriented programming are:

- 1. Encapsulation**
- 2. Inheritance**
- 3. Polymorphism**

Encapsulation:

Encapsulation is the mechanism that binds the code and the data together. It can be thought of as a protective wrapper(capsule) that prevents code and data from being arbitrarily accessed by other code defined outside the wrapper(capsule).

Basis for encapsulation is class.

- Any method needs object reference for invocation.
- A public method can be invoked from outside the class.
- A private method can only be invoked from inside the class.
- Every class provides some interface(publicly accessible method).

Inheritance:

Inheritance is the process by which one object acquires the properties of another object. (super/parent/base <———— sub/child/derived).This supports the concept of hierarchical classification. Specialization is moving from a generalized version to a specialized version. It facilitates reusability of the code.

Super class object reference can point sub class.

- **A Superclass object reference can point to a subclass.**

Polymorphism:

Polymorphism is a feature that allows one interface to be used for a general class of action; the type of action executed depends on the situation. **OR**

Same interfaces performing different actions.

These fundamental properties work together to produce a programming environment that supports development of more scalable and robust programs than using process-oriented paradigm.

Paradigm	Basis	Example
Procedure-oriented	Algorithms/ functions	C, Fortran, Pascal, Algol
Logic-oriented	Goals; predicate logic	Prolog
Object-oriented	Objects, classes	C++, Java, Small talk, Ada, Ruby etc
Functional Programming	Only functions	LISP, ML, Haskell

Properties of OOP

1. Abstraction 2. Modularity 3. Typing 4. Binding 5. Concurrency 6. Persistence

Abstraction :

Using abstraction we can ignore the inner details which are not so essential, and still we could use an object as a whole. Humans manage complexity through abstraction.

Modularity :

It is the act of partitioning a program into individual components. This is to reduce the complexity. Any change in a module leads to minimal/ no change in other modules; the change is localized.

Typing :

A programming language may be strongly typed, and some are untyped. Ex: Java , C++ , Ada are strongly typed. Every Object has its own type (the type of the class) and it can refer to that type only. **But remember that Super class object reference can point sub class.**

If Typing is not implemented, any object can invoke some method (defined for another object) even though the method is not defined for it; this violation is not known until execution. Strongly typed language facilitates easy compilation.

Binding :

Binding refers to the time when names are bound to object types. Binding is entirely different from typing.

Static/early binding : Means that the type of object pointed by reference is fixed at the time of compilation, also known as compile-time binding.

There is a lot of difference b/w **Object reference variable and Object**

Object Reference variables are just pointers; objects are the actual ones which will take some space on the memory and hold data pertaining to that memory

Ex - `Box b = new Box();` → b is the object reference variable (name)

The object reference variable occupies 2 bytes of memory whereas the object occupies memory depending upon the template of the class.

By looking at the code statement itself one can say b is pointing to an object of type Box this is known as **compile-time pointing**.

Dynamic binding: Object types are not known until runtime, also known as run-time binding.

Concurrency :

For certain kinds of problems, an automated system may have to handle many different events simultaneously. Threads. Single CPU, multiple CPUs.

Persistence :

Persistence is the property of an object through which its existence transcends time or space to secondary storage (i.e. the object continues to exist after its creator ceases to exist or the object moves from one address space to another)

Multiple Inheritance

Multiple Inheritance is the process by which a class acquires the properties of two different classes. Java doesn't support multiple Inheritance as it needs to address two things:

- ❑ **Name clashes** arise when the two superclasses have methods/variables with the same name. C++ resolves the name clash by attaching the classname qualifier.
- ❑ **Repeated inheritance** occurs when two or more peer superclasses share a common superclass. In such a situation, the inheritance lattice will be a diamond, and so the question arises, does the leaf class have one copy or multiple copies of the structure of the shared superclass.

Object Model

- All the predefined and user-defined classes are subclasses of supermost class called as Object class

Defining a class means defining a template that doesn't occupy any space only when the object is created, the memory gets allocated for members declared inside the class. Hence, class is a blueprint for an object.

An object is an Instance of a class. A single object belongs to a single class only, it can't be an object for two different classes. An object has **State, Behavior, and Identity**.

- **State** of the object is properties and the values assigned to the variables or members.
- **Behavior**(methods) is how an object acts and reacts, in terms of its state changes and message passing.
- **Identity** is that property of an object(memory occupied-memory location) which distinguishes it from all other objects.

It is impossible to create the same objects with the same space and same memory location. **But, two object referencing variables can point to the same object.**

Every class has two parts : Interface part and implementation part; only the interface part is made accessible(**public**) to the outsiders, the implementation of a class is its inside view.

Naming Convention

Hungarian notation is used as a naming convention in JAVA(OOP). Class names start with caps and new words will start with caps. Variables and methods– start with a small case and new words will start with caps.

Class can be seen as a template which defines the structure for similar kinds of objects. Objects are the instances of the class which occupy memory space in the computer

Constructors and Methods:

Whenever a constructor(no argument/parametrized) is defined in the class the default constructor ceases to Exist. The default constructor exists only if there is no constructor at all.

- **We cannot invoke a non-static method/variables from a static method.**
- A method and a constructor can have the same name.
- Static variables are mutable.
- Every method that is invoked by the system there will be a stack frame corresponding to that invocation.
- All objects occupy space in heap memory; Only variables that are local to methods are stored in stack frame.

Automatic Garbage collection:

Daemon thread : A predefined low-priority thread which is always running in the background which we are unaware of.

The purpose of this thread is that at equal intervals of time it will check every object utilizing space in heap is being used or not; if not it will reclaim that space occupied by the unused object and make it ready for other purposes.

It's life depends on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

finalize method:

It facilitates performing certain actions just before an object is destroyed by the garbage collector.

Syntax: `protected void finalize(){ //code }`

Method Overloading:

- Compile-time Polymorphism
- Defining two or more methods in a class with the same name and different signature is known as Method Overloading.
- MethodOverloading is useful in using the same method name to perform different operations based on the requirement in a different way.
- It is important to note that the return type of the method doesn't matter.
★ Method Overriding concept comes only in the context of Inheritance.

Parameter Passing:

The parameters at the function call are actual parameters and the parameters at the function definition are formal parameters. Always the formal parameters are local variables only their scope ends at the function definition.

At call by value, a copy of the actual parameter is made and modifications are done to it which have no effect on the actual one.

At call by reference when we pass on objects as references (object reference as an argument) whatever modifications done inside the method it gets affected on the actual one.

Recursion:

1. Java supports recursion. Methods which call themselves are known as recursive methods.

2. The main advantage of recursion is that they can be used to write clearer code for algorithms that are iterative in nature.
3. Recursive methods must have a base case to terminate or else many recursive calls to a method can cause stack overflow.
4. In the main memory there may be several frames in the stack at once for a recursive method.
5. Execution of recursive calls are slow because of repetitive allocation and deallocation of heap memory.

Access Specifiers:

Encapsulation gives rise to access control on class members. Access Specifier determines how a member can be accessed.

Using encapsulation a programmer can make a choice of which part of the code an outsider can have access to.

Public, Private, Protected, default are the 4 access specifiers present in java.

Static Members:

A static method can access only a static member but a non-static method can access both a static member and a non-static member; this is because the non-static members vary for every instance .

All static variables are known as class variables which don't require object invocation. Hence, main is static so that we can call it without instantiating the object. All instance variables are known as non-static variables which require object invocation.

These static variables are also allocated in heap memory and have a scope until the end of the program.

So far, the static keyword is used as a prefix for variables, methods. There exists a static block as well which contains some code. Static block is first executed when

the class is loaded into the system, even before any instance is created. All the rules which are true for static methods hold true for static block as well.

Nested Class:

- A Nested class is a class defined in another class. If class B is defined inside class A, it is known to A but not outside A, that B does not exist independent of A.
- All members (including private) of A are accessible by B. But, A can not access members of B.
- It is also possible to declare a nested class local to a block. Any code outside Outer class cannot create an inner class object.
- Representation of classes created: **Outer.class, Outer\$Inner.class, Outer\$Inner1\$Inner2.class**

Overriding equals() :

If we want to compare the objects based on the content inside them we use this equals method which is a part of the supermost class the object class.

Object obj

- ★ A superclass object reference can always point to a subclass object but not vice versa.
- ★ If the method is not overridden then equals() method compares the references only.

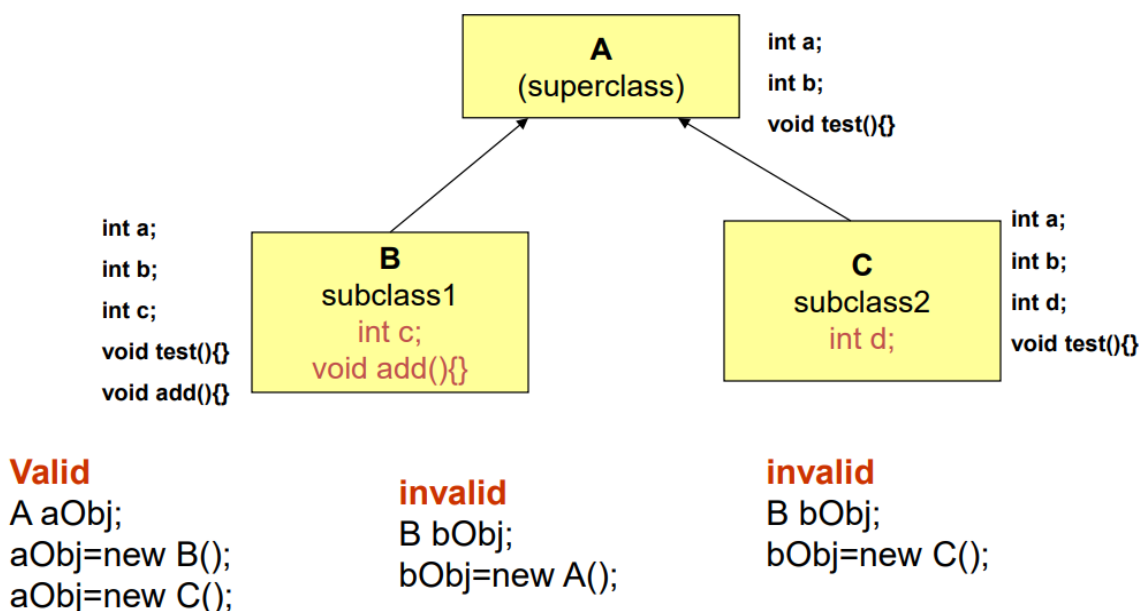
Inheritance

A subclass is inherited by the superclass. The class which inherits is known as superclass and the class which is inherited is known as the subclass.

We use the keyword **extends** for inheriting a class from other.

A subclass can have access to its superclass members but a private member of the superclass is not accessible to the subclass.

An Object reference variable of superclass can be used to point to any subclass object of that superclass and it can have access only to those which are known to superclass.



super:

The keyword super is used in two ways

1. Calling superclass constructor from subclass constructor, and must be the first statement in subclass constructor.
2. Access superclass members when subclass members hide superclass members.

★ **Difference between this and super keyword** : this keyword is used to refer to a current object which is invoked in the constructor/method. super keyword is used to call the superclass constructor in the subclass constructor and also to access the hidden superclass members in the subclass (ambiguity in naming).

Constructor Inheritance:

For a subclass object all its superclass constructors are first executed and then the subclass constructor will execute by default as the subclass object gets created.

Method Overriding:

- Run-time Polymorphism/ Dynamic-method dispatch
- In some situations we are not aware of the type of object the object reference variable is referring to; a mechanism by which a call to an overridden method is resolved at runtime, rather than compile time is known as Dynamic-method dispatch.
- Providing different implementations for the method that has been inherited from the superclass in sub classes is known as Method Overriding.
- When a method defined in superclass is overridden in subclass, the subclass method access specifier must loosen/be the same access as of superclass method access specifier.

A subclass can have a method defined, which has the same name and type signature (arguments & return type) as the one available in its super class; the subclass is said to override the method in its superclass.

Why overriding? superclass can specify methods that will be common to all its derivatives.

Abstract class:

1. A class having atleast one method undefined/unimplemented is known as Abstract class.
2. Since the superclass is unable to define the implementation, it is the responsibility of the sub class to provide implementation required. **If it doesn't do so then the subclass must also be an abstract class.**
3. As there is no completeness an abstract class object can not be instantiated.
4. Hence, we will declare an object reference variable of an abstract class and make it point to some concrete subclass object. An abstract class can have any number of subclasses.

final:

1. For methods: A superclass method specified with 'final' modifier can not be overridden by any of its subclasses.
2. For classes: A class specified with 'final' modifier can not be inherited.

Strings

1. In Java, strings are treated as objects. The String class is used to create a string object.
2. String is immutable in nature, Whenever we change any string, a new instance is created.
3. When String concatenation takes place the string reference variable stops pointing to the old string literal and starts pointing to the newly appended string in a different location.
4. Mutable strings can be created by using StringBuffer and StringBuilder classes.

Creating strings:

1. **Using literals** : string object is created by the compiler itself.
If the object already exists in the memory (string constant pool) it does not create a new Object rather it assigns the same old object to the new reference.
2. **Using 'new' operator**: string object is created by the programmer.

```
String str1 = new String("Welcome");  
String str2 = new String("Welcome");
```

This would create two different objects with the same string value.

StringBuffer:

StringBuffer supports a modifiable string. StringBuffer may have characters and substrings inserted in the middle or appended to the end. It often has more characters preallocated than are actually needed, to allow room for growth.

StringBuilderClass:

StringBuilder is similar to StringBuffer except for one important difference: it is not synchronized, which means that it is not thread-safe. The advantage of StringBuilder is faster performance.

StringTokenizerClass:

The `java.util.StringTokenizer` class allows you to break a string into tokens. It is a simple way to break strings. It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc.

When an object reference variable of class String is printed it prints the content of the object whereas if any other object reference variable of a user-defined class is printed it prints the reference! Why is it so ?

- The output function `System.out.println()` always considers the content inside the command as String. If at all any other data type or an object reference variable is supposed to be printed; it implicitly typecasts into string type using the `toString()` function in the `java.lang` package.
- When an object reference variable of class String is printed the `toString()` method is overridden and hence we get the content displayed at the output; and when any other object reference variable of a user-defined class is printed the `toString()` method is not overridden and hence we get the address of the content displayed at the output;

Interfaces

An Interface defines a set of methods but does not implement them. A class Implements the Interface thereby agreeing to implement all the methods defined inside the interface.

Variables declared in an interface must be defined there itself.

Java supports Single Inheritance; hence Interfaces in Java are used to achieve Multiple Inheritance. A class can implement any number of interfaces but can inherit only one class.

- If a class M implements an interface A then M is an implementation class of A.
- “extends” is the keyword used when one class is inherited from another.
- “implements” is the keyword used when one class implements an interface
- One interface can inherit another interface by using extends.

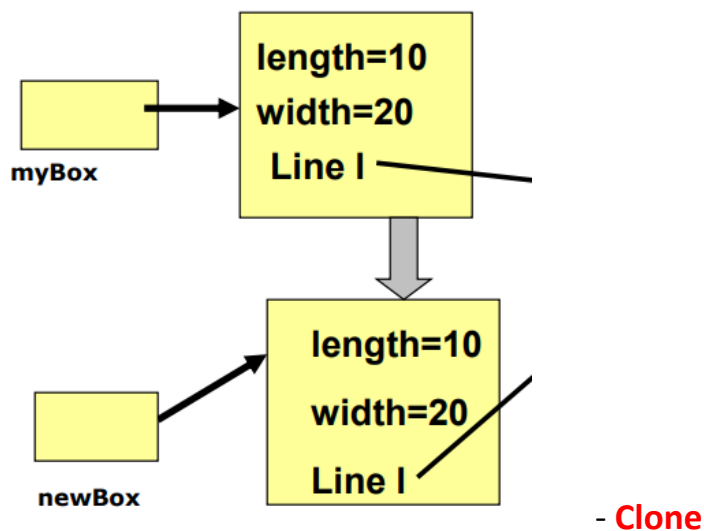
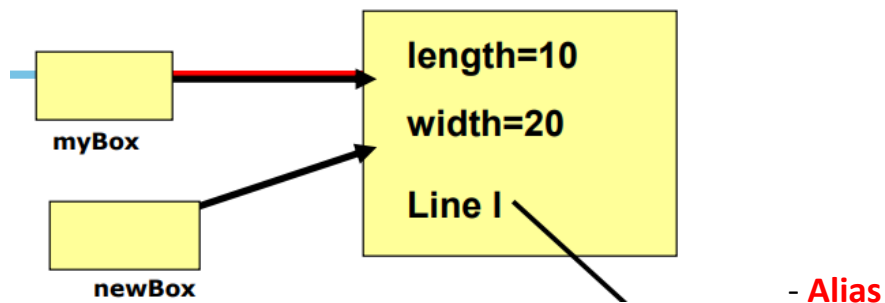
All methods in the interface are by default public and abstract. And all variables defined in the interface are static and final by default.

An object reference variable of an Interface can point to an object of the class implementing the interface. Doing so, it can access the variables/methods declared in that interface only.

Clone:

Instead of creating another object reference which becomes an alias, we are creating the same object again in a different location.

The clone() method Object class generates a duplicate copy of the object on which it is called.



When a clone is made, the constructor is not called. A clone is simply an exact copy of the original.

The class using the box function must implement the interface cloneable which does not contain any methods. It is just a token for the objects/classes to become cloneable.

The clone method is available in the supermost class object; it has been implemented as a protected method.

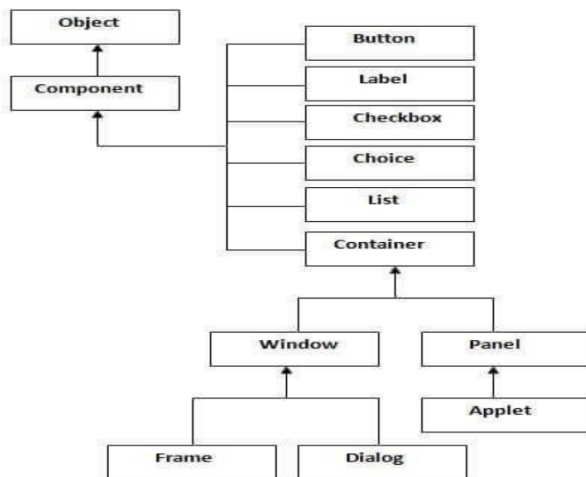
```
class_name new_obj = obj_name.clone();
```

As long as we modify simple data types of a cloned object, it will modify variables independently, but if an object of another class is used as an instance variable then that object gets aliased.

AWT

AWT stands for Abstract Windowing Toolkit. It is used to implement GUI design with classes available in the package [java.awt](#)

Java AWT is an API used to develop GUI/window-based applications. Java AWT components are platform dependent i.e components are displayed according to the view of the operating system. AWT components use the resources of operating system and hence it is heavy weight



- GUI on web browser: Applet

Applet is a java code which can be compiled and embedded within html, which can be launched on a web server and the html page can be transferred to the client server via http protocol. When the html page is opened in the browser automatically the applet which is embedded as the part of html will be executed in the JVM of that browser.

- GUI on local machine : Frames

Frame classes are used When we need to run the GUI on the local system itself.

★ **Applets and Frames differ in life cycle.**

A window has the title bar and it has minimize, maximize, close options at its right top corner whereas the panel doesn't. A dialog is a pop window which arises during a particular installation or execution.

1. Components
2. Layout Managers
3. Event Handling

Components

A component is an object having graphical representation that can be displayed on screen and can interact with the user.

Ex: Buttons, textfield, Lists, labels, checkboxes etc.

Methods used : add(), setSize(), setLayout(), setVisible()

Container is an AWT component itself and it adds the capacity to add components to itself. Subclasses of containers are also containers. A default layout is present in every container which can be overridden.

Ex: Window, Panel, Frame.

Frame is a container that can have title bar and menu bars. It can have other components like button, textfield

Delegation based event handling mechanism: Rather than source handling the event it is delegated to the object of the class implementing an action interface.

Layout Managers

A container has a layout manager to arrange its components. A container has a `setLayout()` method to set its layout manager.

To setup the layout of a container

1. Construct an instance of the chosen layout object, via new and constructor
Ex: `new FlowLayout()`

2. Invoke the `setLayout()` method of the Container, with the layout object created as the argument -> `setLayout(new FlowLayout())`

3. **`Frame f= new Frame();`**
`f.setLayout(new FlowLayout());`

AWT provides 6 kinds of layout managers:

- FlowLayout
- GridLayout
- BorderLayout
- CardLayout
- GridBagLayout
- BoxLayout

Flow Layout:

Components are arranged from left-to-right inside the container in the order that they are added.

`public FlowLayout(int alignment, int hgap, int vgap);`

alignment can be `FlowLayout.LEFT`, `FlowLayout.RIGHT` or `FlowLayout.CENTER` **(d)**

hgap is the horizontal gap between the components and vgap is the vertical gap between the components. `hgap = vgap = 5` **(d)**

(d) : default values

Grid layout:

components are arranged in a grid (matrix) of rows and columns inside the Container. Components are added in a left-to-right, top-to-bottom manner in the order they are added.

```
public GridLayout(int rows, int columns, int hgap, int vgap);
```

```
rows = 1, cols = 0, hgap = 0, vgap = 0 (d)
```

Border layout:

The container is divided into 5 zones: EAST, WEST, SOUTH, NORTH, and CENTER.

Components are added using method `aContainer.add(acomponent, zone)`

Zones: BorderLayout.NORTH (or PAGE_START),
BorderLayout.SOUTH (or PAGE_END),
BorderLayout.WEST (or LINE_START),
BorderLayout.EAST (or LINE_END),
BorderLayout.CENTER.

Card layout:

It treats each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards. ... Object) method can be used to associate a string identifier with a given card for fast random access.

Exception Handling

Exception is an abnormal condition that arises in the code section during runtime.

Java runtime system creates an object that generates an exception which immediately halts the further execution of the program

Java provides a the user to handle the exception in some manner and proceed further with the remaining part of the code without abrupt termination

Java uses **“try- catch – throw – throws – finally”** keywords.

Try block and catch block:

- Try & catch blocks that always come together. It means catch block should begin at very next of try block termination.
- try block is used to check if any exception exists or not. So, the code section which might throw an exception must be written in try block.
- Once an exception is raised it immediately exits the try block and enters the Catch block which tries to resolve the exception; it is the exceptional handler.
- If no exception rises in the try block then the compiler doesn't enter the catch block. There can be more than one exception in the catch block.

Exception mismatch occurs if the exception thrown is not the same as the exception object that is passed in the catch block.

Once the try block execution is finished the try block searches for the catch block until it finds a suitable one; if no suitable catch block is found the program halts there itself.

Though we have multiple handlers(catch blocks) the system will find out appropriate exception handler (catch block) which can handle the exception.

In multiple catch the catch block which can catch all types exceptions must be put in the end because if none of the catch blocks catches the exception then only it should catch it and this catch block is known as **Generic catch handler**. All other catch handlers specific to an exception are known as **Specific catch handlers**.

It is important to remember that Exception subclasses must come before their super class. Once the Generic catch block is used it resolves the exception there itself and any Specific catch blocks written after this make no sense and hence throws an error. Hence we need to write the Generic catch handler after all the Specific catch handlers are written.

There are two types of exceptions:

Checked : A checked exception is an exception that is checked at compile time.

Unchecked : All other exceptions are unchecked exceptions, also called runtime exceptions, because they are unchecked at compile time and are detected only at runtime.

1. To forcibly throw an exception, **'throw'** is used as a prefix to the exception name.

```
throw new Ex1();
```

2. Any exception thrown out of a method must be specified as such by a **'throws'** clause.

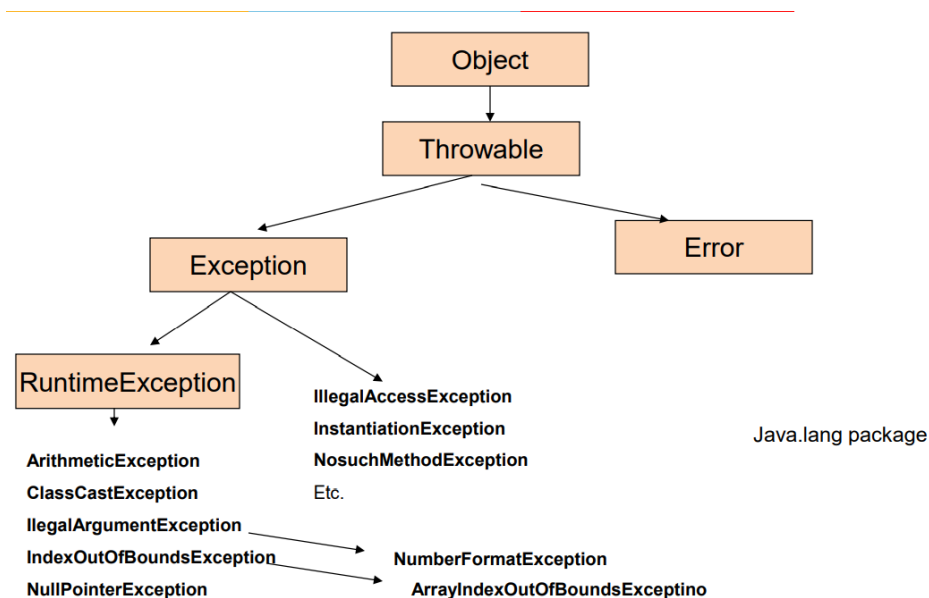
3. Any code that must be executed before a method returns is put in a **'finally'** block. There can be only one finally block.

- A try block without catch or finally block is an error at compile-time.
- A try block without catch block but a finally block is invalid
- A try block without a finally block but a catch block is valid.
- A try block with multiple catch blocks is valid
- A finally block without try block is invalid
- A catch block without a try block is invalid.
- Finally block must be written after catch block only; it can't be kept in between try and catch.

Code written in the finally block gets executed irrespective of whether the exception is resolved or not. In case of abnormal termination also the code written in the finally block gets executed. It gets executed before the normal code section begins to execute.

Any exception must be a subclass of throwable class.

Error Class is a sibling class of Exception. Exceptions of type Error are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. Stack overflow is an example of such an error.



Multiple stack frames can be seen in the case of recursion and also if a method makes another method call in it.

Whether a thread is created or not (or) whether the program is a multithreaded program or not there always exists one thread known as the main thread.

Objects that can implement a cloneable interface can be cloned.

NESTED Try:


The tries can be nested. In case of nested try, any exception object is thrown in the inner try block and if there is no matching catch for that particular exception, then it searches for a catch block after the outer try block..

Throws:

- If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception.
- We do this by including a throws clause in the method's declaration.
- A throws clause lists the types of exceptions that a method might throw.

```
void method( ) throws ABCException
{
    //ABC exception can be thrown which is not handled
    here.
    ----- ;
}

try {
    method( );
}
catch( ABCException e ){
    // handle the Exception e.
}
```



Multi-Threading

Multi threading means to split a task into parts which then be executed concurrently in its own execution path.

Multi threading is a specialized form of multi tasking.

Concurrency vs Parallelism:

We define two processes to be in **parallel** when they have the capability to run uninterrupted on a dedicated resource (CPU) till it completes.

Ex1 : Making 4 items for breakfast on 4 burners at a time without interrupting the other.

Ex2 : Now at the same time, in BITS multiple lectures are going on in parallel

We define **Concurrency** where progress is observed on more than one task, however, all the tasks cannot be executed at the same time.

Ex1: Making 4 items for breakfast on 2 burners by juggling the tasks

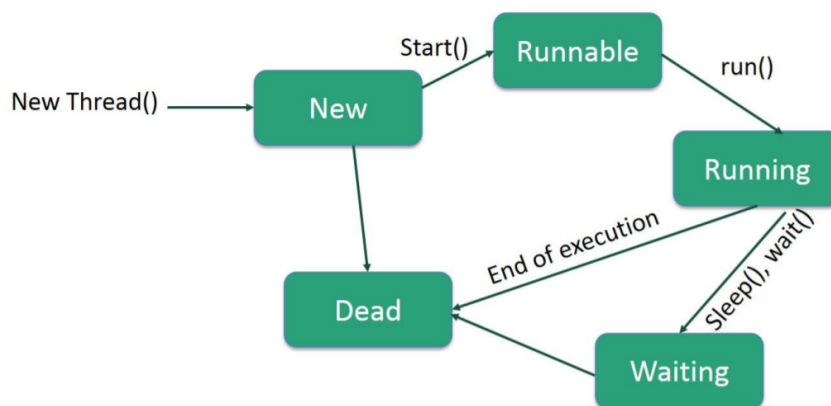
Ex2: : In this session, if another student / TA has to share the screen, they will have to wait till I stop my screen sharing.

Examples of Multi-Threading: While browsing, you can play music videos or while using words you can print while formatting text. Only condition is that these actions run on separate threads.

Multi-threading can be done on processes as well as threads

In java Multi-threading one thread can be stopped without interrupting other threads

Thread Life-cycle:



Thread Creation :

There are 2 ways to create/write a thread in JAVA:

1. Write a class that implements the Runnable interface, then associate an instance of your class with a java.lang.Thread object
2. Write a class that extends the Thread class.

In either case, you define the one method: public void run()

```
public class MyRunnable implements Runnable
{
    public void run()
    {
        thread action
    }
} ...
Runnable r = new MyRunnable();
Thread t = new Thread(r);
t.start();
```

1 ->

The body of the `run()` method is the path of execution for your thread. When the thread starts running, the `run()` method is invoked, and the thread becomes dead when the `run()` method runs to completion.

We know that main thread that runs throughout the program; all the threads we create are spawned from the main thread. All threads get terminated as soon as the main thread terminates.