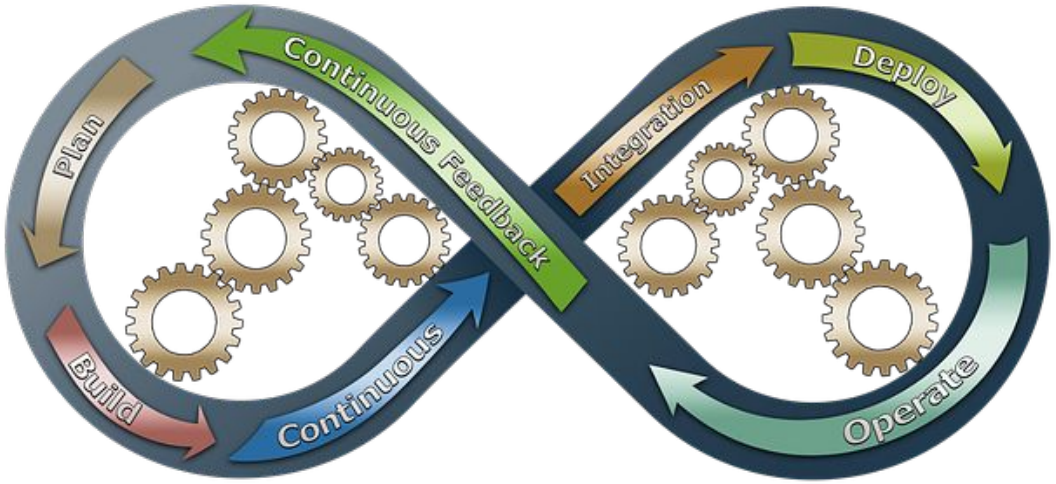


Session 3.6

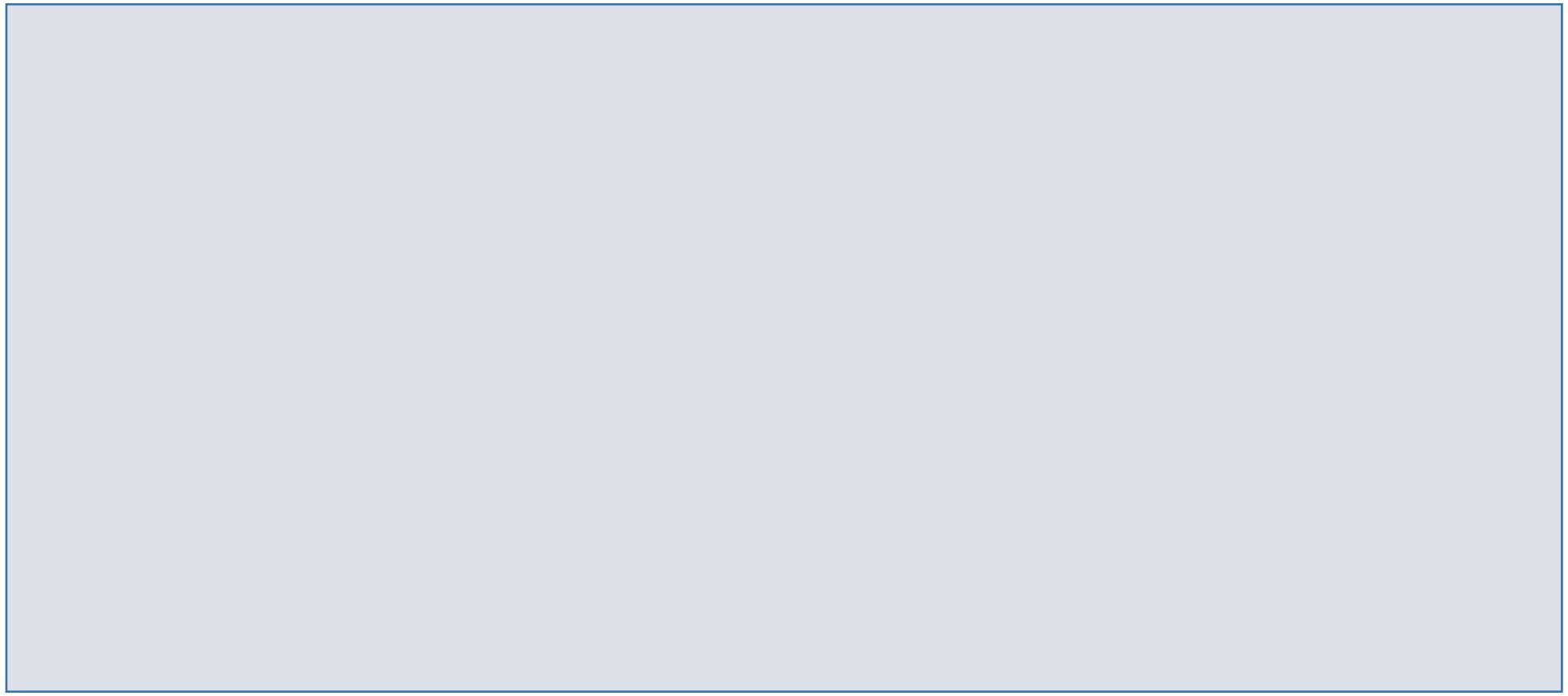
OOPS - An Overview

AN INITIATIVE BY

UNICAL ACADEMY

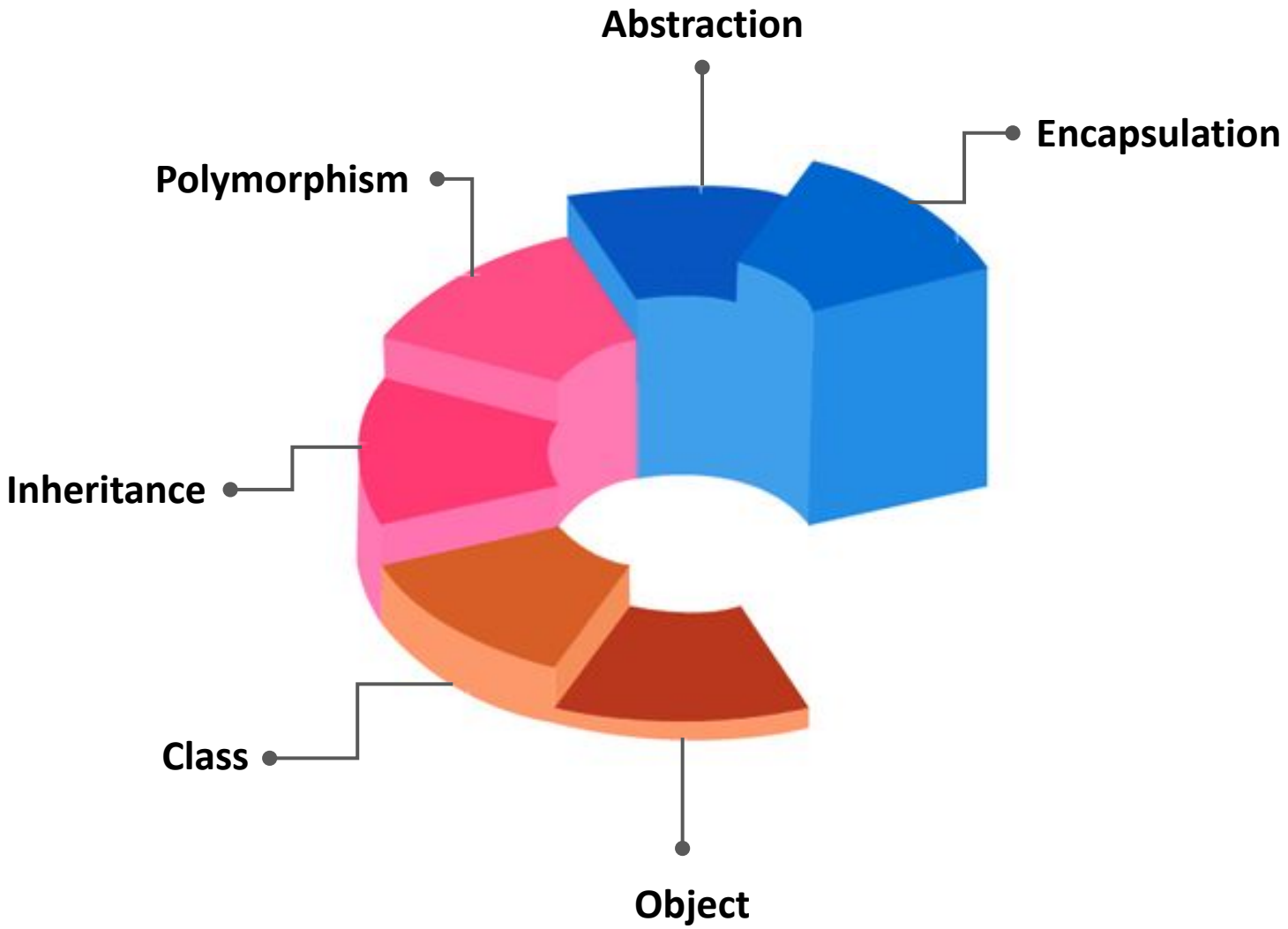
UNICAL ACADEMY

Introduction



Let’s go!!!

OOPS - An Overview

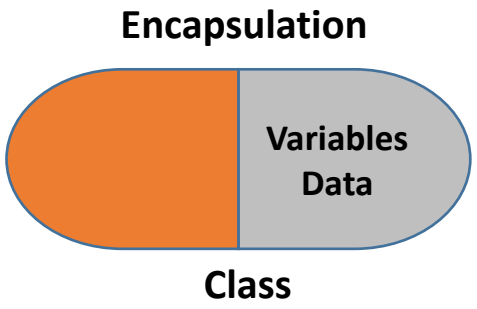


Encapsulation

UNICAL ACADEMY

Encapsulation:

- Encapsulation is defined as the wrapping up of data under a single unit.
- Encapsulation is also known as "**data Hiding**".



Implementation of Encapsulation:

- Make the instance variables private so that they cannot be accessed directly from outside the class. We can only set and get values of these variables through the methods of the class.
- Have getter and setter methods in the class to set and get the values of the fields.

```
class EncapsulationDemo {
        private int ssn;
        private String empName;
        private int empAge; //Getter and Setter methods
        public int getEmpSSN() {
           return ssn;
                   }
       public String getEmpName() {
           return empName;
                  }
       public int getEmpAge(){
            return empAge;
                  }
      public void setEmpAge(int newValue) {
          empAge = newValue;
                  }
      public void setEmpName(String newValue) {
              empName = newValue;
                 }
          public void setEmpSSN(int newValue) {
            ssn = newValue;
           }
}
public class EncapsTest {
        public static void main(String args[]) {
             EncapsulationDemo obj = new EncapsulationDemo();
           obj.setEmpName("Mario");
           obj.setEmpAge(32);
           obj.setEmpSSN(112233);
         System.out.println("Employee Name: " + obj.getEmpName());
         System.out.println("Employee SSN: " + obj.getEmpSSN());
         System.out.println("Employee Age: " + obj.getEmpAge());
       }
}
```

Abstraction

UNICAL ACADEMY

Abstraction:

- Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- There are two ways to achieve abstraction in java
  1. Abstract class (0 to 100%)
  2. Interface (100%)

Rules for Abstraction class:

- An abstraction class must be declared with an abstract keyword
- It can have abstract and non-abstract method.
- It can not be Instantiated
- It can have final methods
- It can have constructors and static methods also

Uses of java interface

1. It is used to achieve abstraction
2. By interface, we can support the functionality of multiple inheritance
3. It can be used to achieve loose coupling

Polymorphism

Polymorphism:

- Polymorphism is a concept by which we can perform a single action in different ways.
- There are two types of polymorphism :
  1. Compile-time polymorphism
  2. Runtime polymorphism.
- We can perform polymorphism in java by method overloading and method overriding.

Compile-time polymorphism:

- It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading. But Java doesn't support the Operator Overloading.

Runtime polymorphism:

- It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Method Signature

- The method signature consists of the method name and the parameter list.
- Method signature does not include the return type of the method.
- A class cannot have two methods with same signature.

Example:

```
public class MethodSignature {
  public int add(int a, int b){
    int c = a+b;
    return c;
  }
  public static void main(String args[]){
    MethodSignature obj = new MethodSignature();
    int result = obj.add(56, 34);
    System.out.println(result);
  }
}
```

Overloading

Method Overloading:

- When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments.

Test
Void fun(int a) Void fun(int a, int b) Void fun(char a)

Overloading

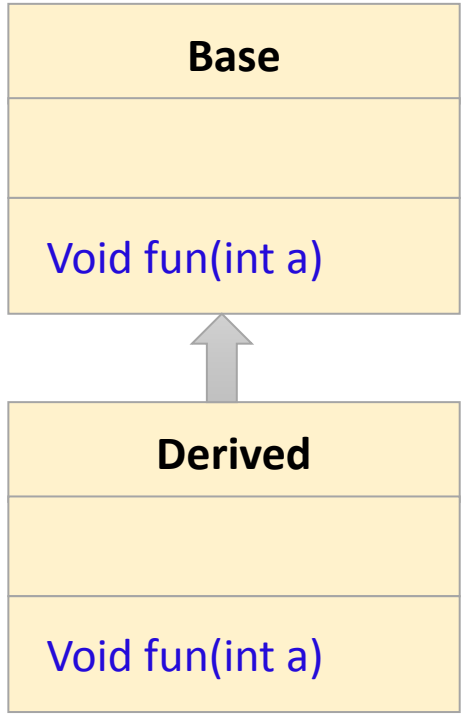
Different ways of doing overloading methods:

- The number of parameters in two methods.
- The data types of the parameters of methods.
- The Order of the parameters of methods.

Overriding

Method Overriding:

- Method Overriding occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.
- Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.



Overriding

Inheritance

Inheritance:

- Inheritance is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- Inheritance represents the **IS-A relationship** which is also known as a **parent-child** relationship.

```
class  Employee{
 float salary = 40000;
}
class Programmer extends Employee{
 int bonus = 10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Terminology used in Inheritance:

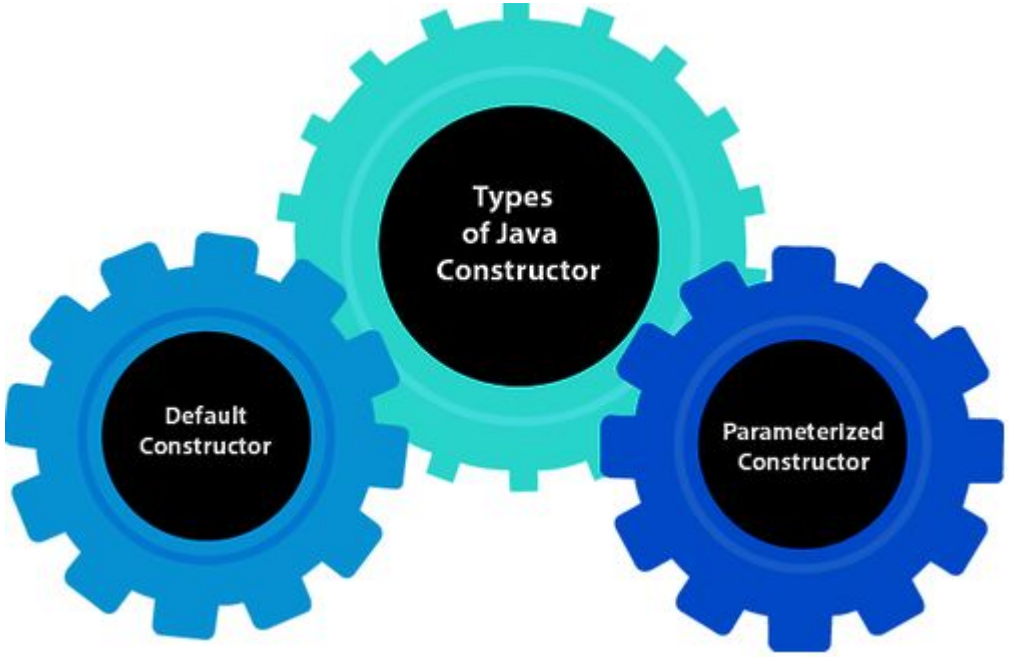
- Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Constructors

UNICAL ACADEMY

Default Constructor:

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.



Parameterized Constructor:

- A constructor which has a specific number of parameters is called a parameterized constructor.
- The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

UNICAL ACADEMY

Example of default constructor:

```
//Let us see another example of default constructor
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

Example of parameterized constructor:

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
   int id;
   String name;
   //creating a parameterized constructor
   Student4(int i,String n){
   id = i;
   name = n;
   }
   //method to display the values
   void display(){System.out.println(id+" "+name);}

   public static void main(String args[]){
   //creating objects and passing values
   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");
   //calling method to display the values of object
   s1.display();
   s2.display();
  }
}
```

This or super

- **super** keyword is used to access methods of the parent class while **this** is used to access methods of the current class.

This keyword:

- **this** is a reserved keyword in java i.e, we can't use it as an identifier.
- **this** is used to refer current-class's instance as well as static members.

Super keyword:

- **super** is a reserved keyword in java i.e, we can't use it as an identifier.
- **super** is used to refer super-class's instance as well as static members.
- **super** is also used to invoke super-class's method or constructor.

Similarities in this and super:

- We can use this as well as super anywhere except static area. Example of this is already shown above where we use this as well as super inside public static void main(String[]args) hence we get Compile Time Error since cannot use them inside static area.
- We can use this as well as super any number of times in a program.

Final & Static

Static:

- It can be applied to nested static class, variables, methods and block.
- It is not required to initialize the static variable when it is declared.
- This variable can be re-initialized.
- It can access the static members of the class only.
- It can be called only by other static methods.
- Objects of static class can't be created.
- Static class can only contain static members.
- It is used to initialize the static variables.

Final:

- It is a keyword.
- It is used to apply restrictions on classes, methods and variables.
- It can't be inherited.
- It can't be overridden.
- Final methods can't be inherited by any class.
- It is needed to initialize the final variable when it is being declared.
- Its value, once declared, can't be changed or re-initialized.

Abstract class and Interface

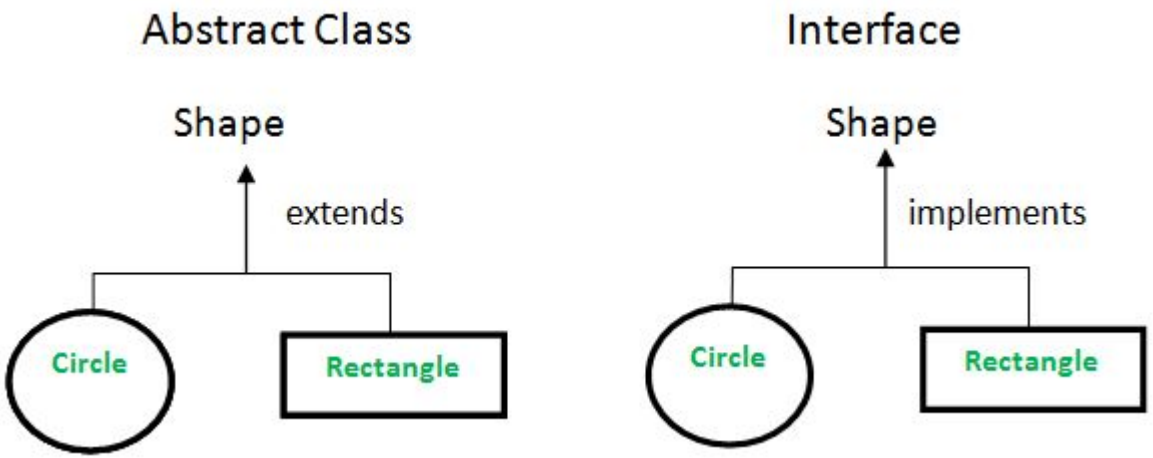
- **Inheritance :** A Java interface can be implemented using the keyword “implements” .
- **Abstraction:** An abstract class can be extended using the keyword “extends”.

Important Reasons For Using Interfaces:

- Interfaces are used to achieve abstraction.
- Designed to support dynamic method resolution at run time
- It helps you to achieve loose coupling.
- Allows you to separate the definition of a method from the inheritance hierarchy

Important Reasons For Using Abstract Class:

- Abstract classes offer default functionality for the subclasses.
- Provides a template for future specific classes
- Helps you to define a common interface for its subclasses
- Abstract class allows code reusability.



- **Type of methods**: Interface can have only abstract methods. An abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also.
- **Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.
- **Type of variables:** Abstract class can have final, non-final, static and non-static variables. The interface has only static and final variables.
- **Implementation:** Abstract class can provide the implementation of the interface. Interface can't provide the implementation of an abstract class.
- **Multiple implementations:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
- **Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.

UNICAL ACADEMY

Session Recap