**OOPS**

* Reusability
* More Programmer Benefitted
* Identify Entity and based on the entity, properties and behaviors are decided

**Characteristic**

* Polymorphism
* Encapsulation-
* Inheritance
* Abstraction

**Encapsulation**

Hide/visibility of the data/functional members (implemented using access specifiers)

* Private
* Default
* Protected
* Public
* **Note:** Access Specifiers shouldn’t be used inside the method
* **Private**- visibility: same class, least visible
* **Default-** visibility: same package , not visible outside the package.
* **Protected-** visibility: same package , inherited members can be accessed
* **Note**: All the protected members visible only if there is a “**is-a**” relation,
* If it is a “**has -a**” relation, its not visible.
* **Public-** visibility- visible outside the package.

Composition: Two classes related to each other( one class cannot exist without another)( both are related wrt code)

Aggregation: Two classes can exist without each other (related in form of data)

Association: Composition +Aggregation

The classes can be of Different types.

* TO Classes ( Transfer Object/Bean/POJO )
* Transfer data from one part to another
* BO Classes ( Business Object)
* Contains only the logics, which will be processed in DAO
* DAO Classes (Data Access Obj)
* Class connected to the database, read/write data from Database

Write a Program to Add 2 numbers using Encapsulation.

**Polymorphism**

* Static Polymorphism
* If the binding happens (recognizing which method to execute) at the compile time is called static polymorphism
* Method Overloading
* Creating number of methods for the same functionalities, but for different datatypes is called method overloading. However Java does not support Operator overloading (static members consumes unwanted memory).
* Dynamic Polymorphism
* If the binding happens in the runtime, which object to be called, is called dynamic polymorphism
* Method Overriding + Implementing the loosely coupling technique, is example of dynamic polymorphism.

Write a overloaded function to reverse the data.

Delegating:

Ex:

**Abstract Class**

* An abstract class can have abstract methods which is non concrete. This abstract should be compulsorily overridden. That means object of Abstract class cannot be created.
* If a class is abstract methods need not be abstract but if the method is abstract class should be abstract.
* An abstract class can have normal methods, constructors, destructors and properties(data)
* One abstract class can extend another abstract class.
* When we want multiple implementations like Addition, multiplication, subtraction so on… but sudden features are common, in those case we can create abstract classes.
* If the features are not common, then we have to go with the interface.
* We cannot create object of Abstract class, but we can create reference of object class.( We can have loosely coupled technique.)

**Interface**

* Pure and complete abstract.
* Here all the methods are abstracts.
* Interface cannot have constructor variables
* If variable is declared inside the interface, it automatically becomes final.
* If the interface contains more than one abstract methods, we have to create a class and implement.
* If the interface contains only one abstract methods, then we can create **lambda Expressions** which is also called as functional interface.
* Interfaces helps for loosely coupled techniques.
* It is always better to create a interface before we create a class.
* One interface can extend another interface.
* A class can implement multiple interfaces, but it cannot extend multiple classes
* It is better to create interface with **interface segregation pattern.** Here we will create multiple interfaces which contains only one abstract methods and segregate interface is created based on the requirement

What is Predicate, Consumer, bi consumer, supplier, bi supplier , functional.