Object Oriented Programming

Authored by : Sangeeta Joshi ***Presented by: Sangeeta Joshi***

This presentation is the intellectual property of Cybage Software Pvt. Ltd. and is meant for the usage of the intended Cybage employee/s for training purpose only.
This should not be used for any other purpose or reproduced in any other form without written permission and consent of the concerned authorities.

Agenda

- History of OOPs
- Need of OOP
- Object Characteristics
- Major Pillars of OOP
- Minor Pillars of OOP
- Inheritance – subclassing
- Polymorphism : Static & Dynamic
- Super keyword

The History of Object Oriented Programming

- The basis for OOP started in the early 1960s
- A breakthrough involving instances and objects was achieved at MIT with the PDP-1.
- The first programming language to use objects was Simula 67
- It was designed for the purpose of creating simulations
- It was developed by Kristen Nygaard and Ole-Johan Dahl in Norway.

The History of Object Oriented Programming(contd)

The term "object oriented programming " was first used by Xerox PARC in their Smalltalk programming language. The term was used to refer to the process of using objects as the foundation for computation.

The objects could be changed, created, or deleted, and this was different from the static systems that were commonly used. Smalltalk was also the first programming language to introduce the inheritance concept.

By the 1980s, object oriented programming had become prominent, and the primary factor in this is C++. Object oriented programming was also important for the development of Graphical user interfaces.

Currently, OOPs such as Java, J2EE, C++, C#, Visual Basic.NET are popular OOP programming languages that any career-oriented Software Engineer or developer should be familiar with.

Need of OOP

- Impedance mismatch between user of the system & its developer
- Greater Flexibility
- Easy user interface
- Client wants the system to be adaptable & extensible

Hello World ….

```
class  Greeting
{
    public static void main (String args[])
  {
         System.out.println(“Hello World”);
   }
}
```

- Save this file with name as Greeting.java
- Compile the file using command : javac Greeting.java
- Run it using command: java Greeting

Characteristics of an Object

Characteristics of an Object

- State
- Behavior
- Identity
- Responsibility

State

State: Current values of the parameters

State can be either static or dynamic

Example: CAR

Static state	Dynamic state
Color	Speed
Make	Fuel Level
Model	Tyre pressure

Behavior

Behavior :

How the object behaves or reacts such that its dynamic state may
change.

Example: Bank Account

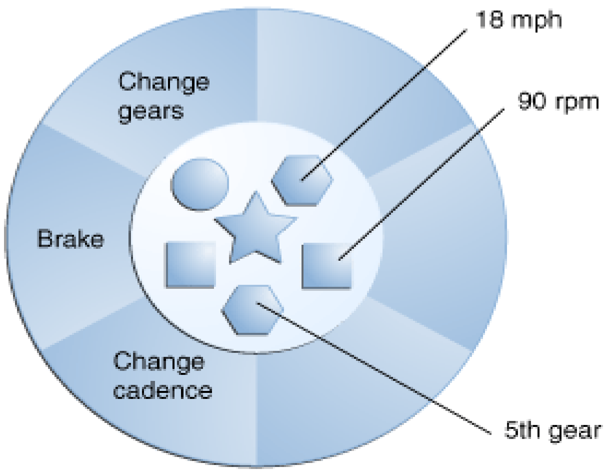
withdraw()

deposit()

It will change the “balance” (i.e. Dynamic state of Account object)

State & Behavior

Representation of static & dynamic state & behavior of a bicycle



Identity

Identity : That property which uniquely distinguishes the entity from all other entities

Example: Car -- RTO / registration no

Responsibility

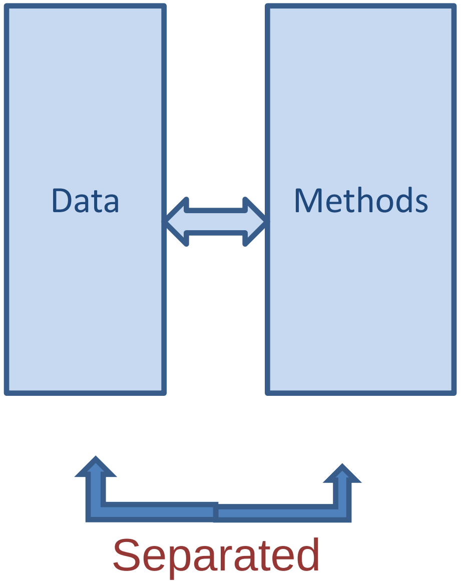
Responsibility: The very purpose or the role that entity serves in the system

Example : Bank account : To enable to carry out money transactions

Car : To take the rider from one place to another

Procedural Vs. OOP

Procedural Languages

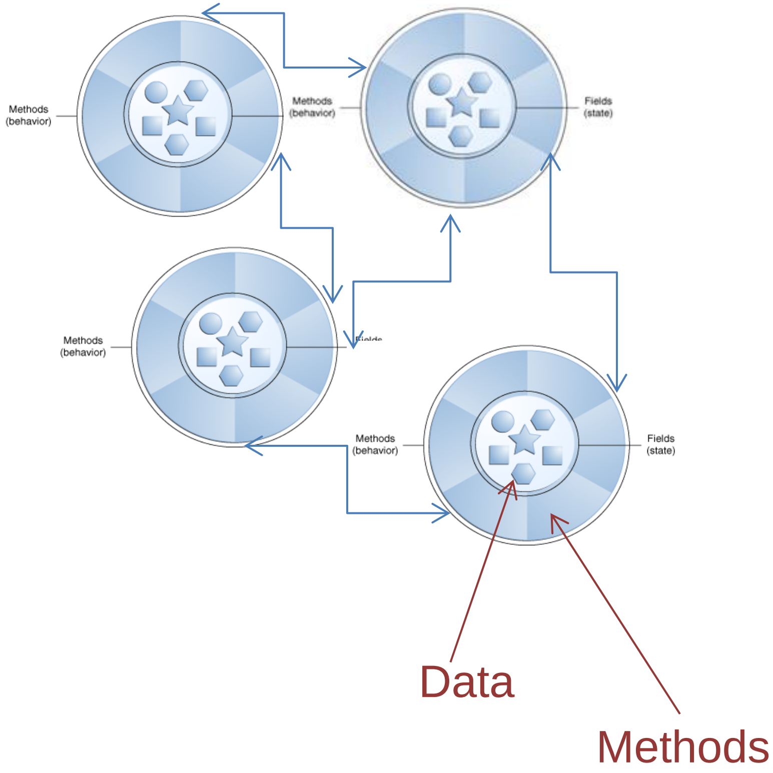


Data

Methods

Separated

Object Oriented Language



Data

Methods

Four Major Pillars

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Abstraction

Abstraction:

- Selective negligence
- Process of identifying the key aspects and concentrating on them by ignoring the rest (Ignore that what is insignificant to you)

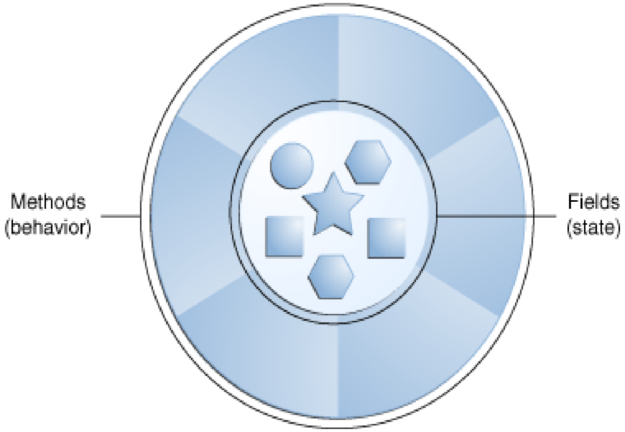
Abstraction of a Person

as an Employee	***as a Patient***	***as a Student***
Name	Name	Name
age	age	age
Educational Qualification	------	-------
-------	blood group	------
-------	Medical history	------
-------	--------	batch

- Abstraction of same entity will be different for different users

Encapsulation

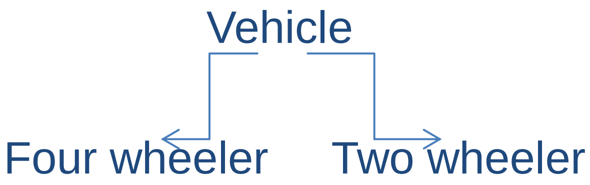
- Software objects are conceptually similar to real-world objects they too consist of state and related behavior.
- An object stores its state in *fields* and exposes its behavior through *methods*.
- Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.
- Hiding internal state and requiring all interaction to be performed through an object's methods is known as ***data encapsulation*** — a fundamental principle of object-oriented programming.



Inheritance

- Inheritance : “*Is-a*” -type of relationship
- Properties of parent are inherited in child.

  Example :



Vehicle

Four wheeler Two wheeler

: a four wheeler is a vehicle.

- Properties of Vehicle are inherited in a four wheeler & a two wheeler

  In addition, a four wheeler can have its own properties specific to it.

- As we go from parent to child we are moving from generic to specific.

  & from child to parent : From specialization to generalization

- Advantage : ***Reusability***
- Another advantage : ***Inheritance builds foundation for dynamic polymorphism***

Polymorphism

Polymorphism: One message and different responses

Example: Traffic signal goes Red – single message :- to Stop
car will stop in its own way
scooter will stop in its own way
bicycle will stop in its own way
This type of behavior is called as polymorphic behavior

In Software Programming, polymorphism is achieved in two ways

- ***method overloading***
- ***method overriding***

Polymorphism helps in writing more maintainable & extensible code.

Three Minor Pillars

- Strong Typecasting
- Concurrency
- Persistence

Inheritence

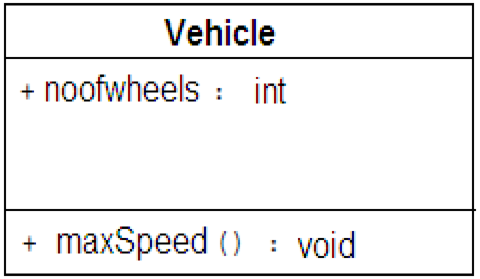
- *Inheritance* is the capability of a class to use the properties and methods of another class while adding its own functionality.

- Java uses the *extends* keyword to set the relationship between a parent class and a child class.

Subclassing

The Vehicle class is as follows

```
public class Vehicle{
public int noofwheels;
public void maxSpeed(){
}
}
```



Vehicle

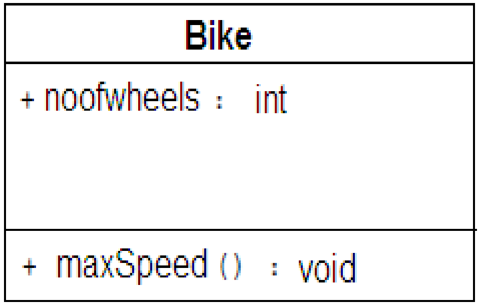
+ noofwheels : int

+ maxSpeed () : void

Subclassing (contd)

The Bike class is as follows

```
public class Bike{
public int noofwheels;
public void maxSpeed(){
}
}
```



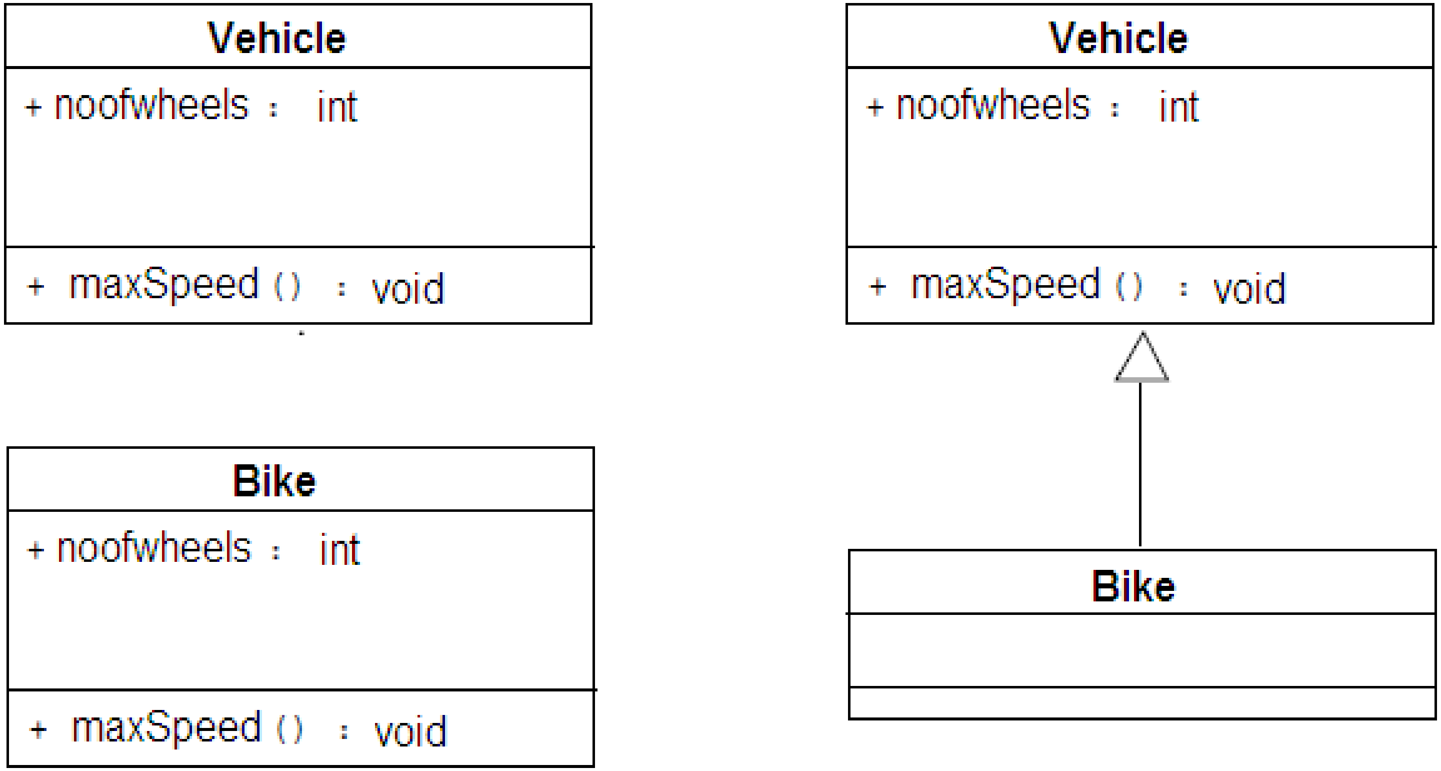
Bike

+ noofwheels : int

+ maxSpeed () : void

Class Diagram

Class diagram of Vehicle and Bike without and with inheritence



Vehicle

+ noofwheels : int

+ maxSpeed () : void

Bike

+ noofwheels : int

+ maxSpeed () : void

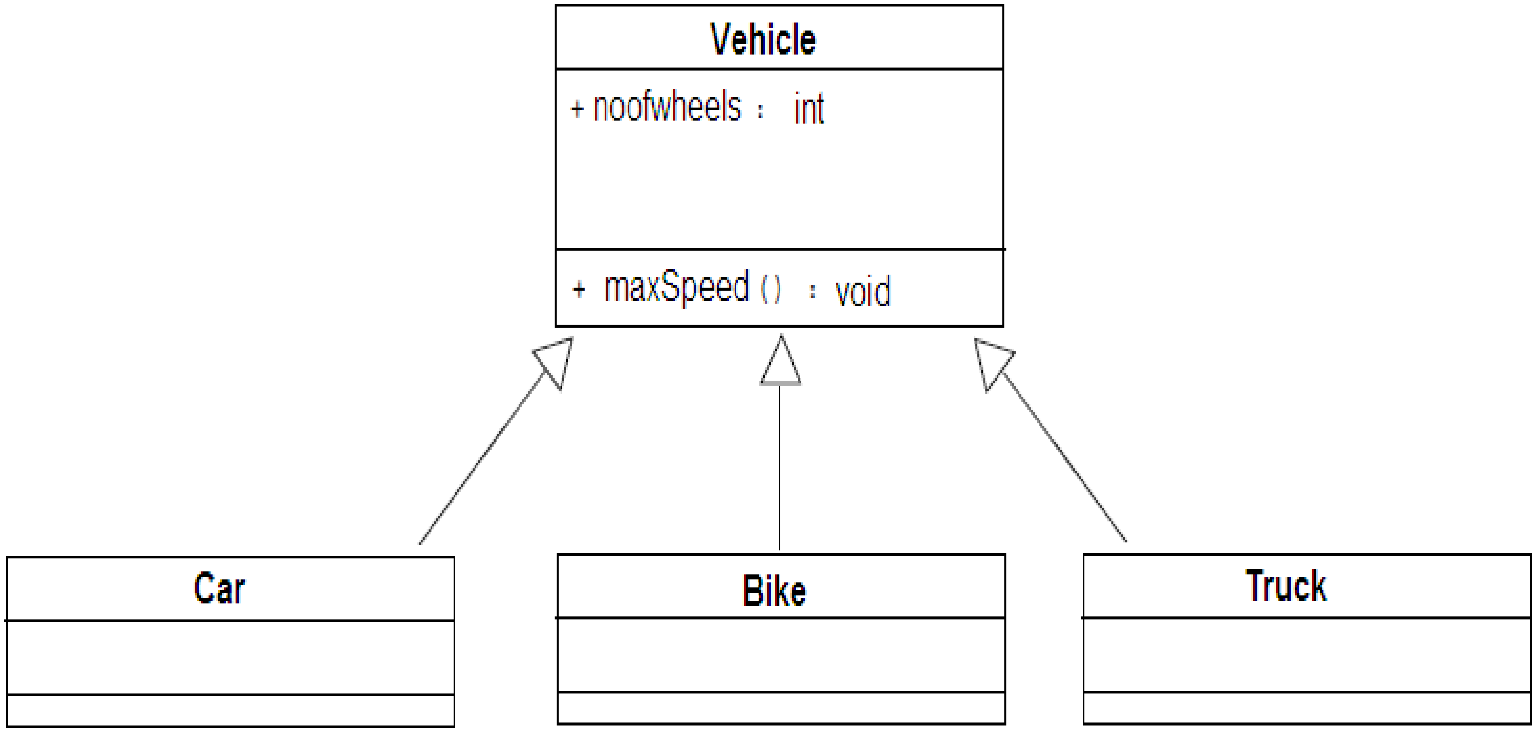
Vehicle

+ noofwheels : int

+ maxSpeed () : void

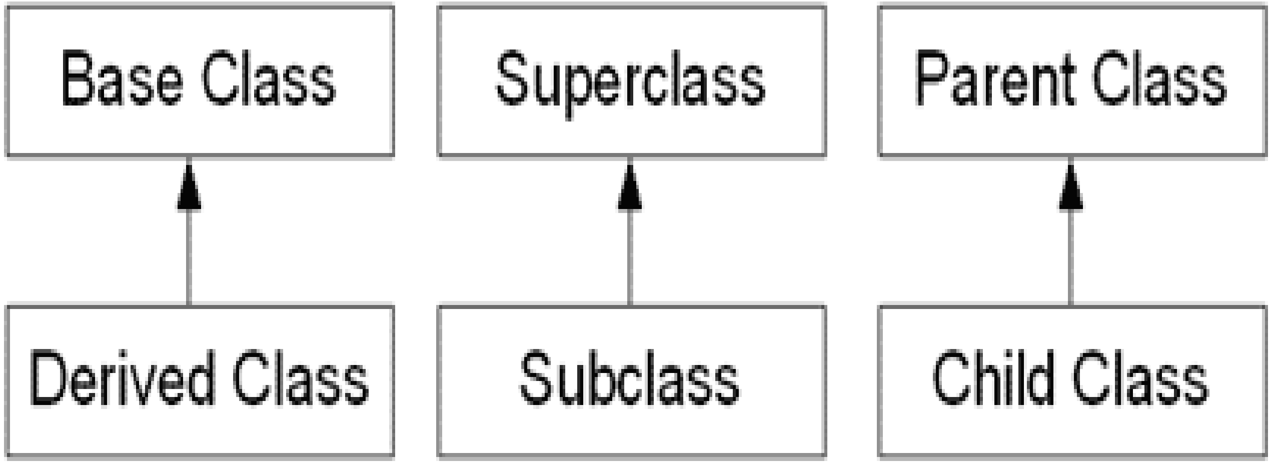
Bike

Single inheritence



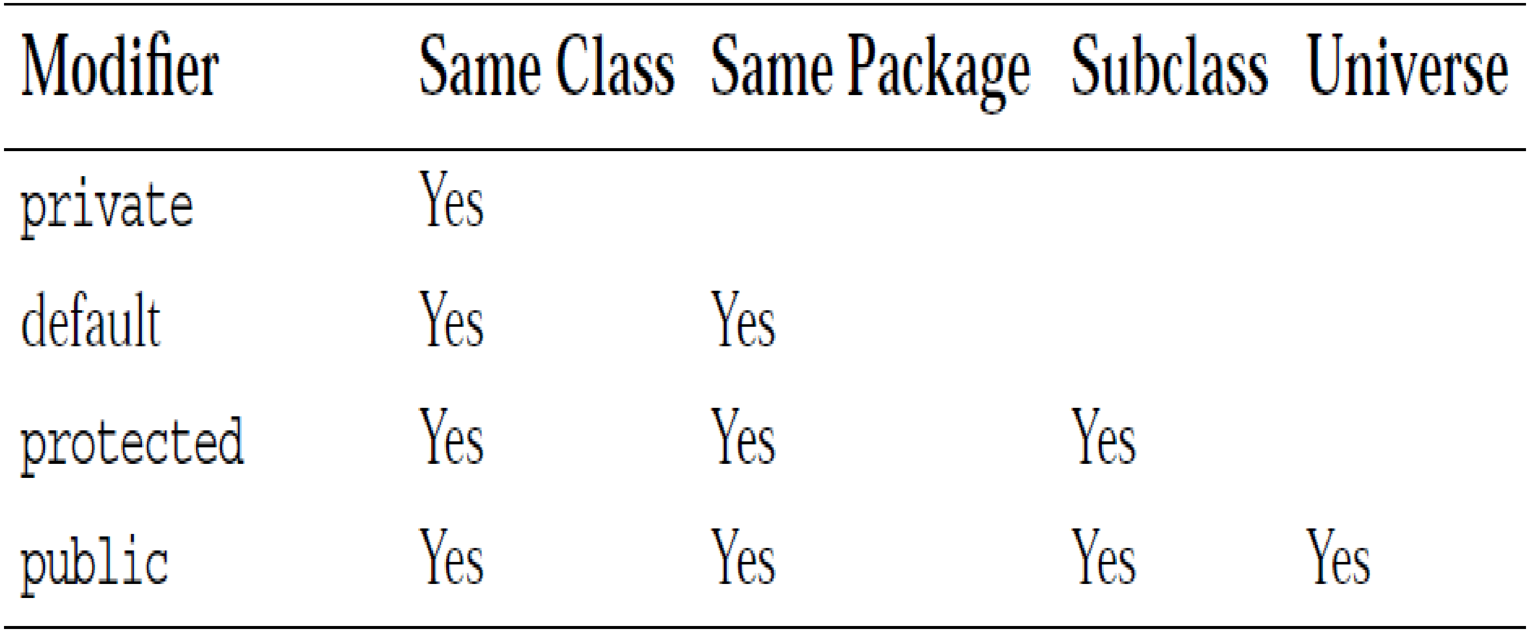
Single inheritence (contd)

Several pairs of terms are used to discuss class relationships (these are not keywords)



Access Control

Access modifiers on class members are:



Modifier	Same Class	Same Package	Subclass	Universe
private	Yes			
default	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

Polymorphism

- Polymorphism (one message & different responses):
  - It involves one method name with no.of different implementations
- The method call is resolved to appropriate method implementation
- Polymorphism can be achieved in two ways:
  method overloading & method overriding
- Polymorphism helps to desgin & implement systems which are more easily extensible & maintainable

Types of Polymorphism

The two types of polymorphism :

1) Static Polymorphism also known as Compile time polymorphism

2) Dynamic Polymorphism also known as Run time polymorphism

Static Polymorphism

- Function Overloading is an example of static polymorphism.
- Overloaded methods have same name but different method signatures. (Method signature may vary in 3 ways)
- The method call is resolved to suitable method implementation at compile time .
  (compiler searches for matching function signature)

Static Polymorphism (contd)

Example:

```
public class Mclass
{
    public void add(int x){}
    public void add(int x , int y){}
     public void add(int x, float y){}
     public void add(float  x, int y){}
}
```

Dynamic Polymorphism

- Dynamic polymorphism is achieved through method Overriding
- Overriding is directly related to Sub -classing
- Method is said to be overridden when a subclass modifies the behavior of superclass method to suit its requirement.
- The new method definition must have the same method signature (i.e., method name and parameters) and return type should also match.

Dynamic Polymorphism (contd)

Example:

```
class vehicle
{
    public void showSpeedRange()
     { System.out.println(“No range specified”)}
}
class  Car extends vehicle
{
     public void showSpeedRange()
     { System.out.println(“Range: 0-300 ”)}
}
class  Bike extends vehicle
{
     public void showSpeedRange()
      { System.out.println(“Range: 0-120 ”)}
}
```

Dynamic Polymorphism (contd)

```
class SpeedDemo
  {
        public static void main(String args[])
            {
                  Vehicle v[] = {  new Car() ,new Bike(),new Car}
                     for(int i=0; i< v.length; i++)
                            v[i].shoeSpeedRange();
            }
   }
```

Static data type of v[i] is vehicle & *dynamic data type* is either Car or Bike. Here **Dynamic Data Type will govern the method selection**…. So its dynamic polymorphism

Dynamic Polymorphism (contd)

The new method definition cannot narrow the accessibility of the method, but it can widen it.

```
class vehicle
{
   public void mileage(){}
}
class  Car extends vehicle
{
   private void mileage(){} \\ not allowed
}
```

Super keyword

Super:

- The keyword super is used in a class to refer to its superclass.
- It can refer to both data attributes and methods of super class
- A subclass method may invoke a super class method using the super keyword

Invoking super Class Constructors

- When a subclass object is created, constructor gets called in the order from Super to sub.
- If there is an explicit call to super class constructor from sub class constructor then that call should be the first statement

```
class Car extends Vehicle
 {
        public Car (int now)
        {
                super(now);      //--------------- call to super must
                                    be the first statement
         }
 -----
-----}
```

Dynamic Polymorphism (contd)

Example:

```
class Mainclass1{
   public string display(){
   return “hello”;
   }
}
class Mainclass2 extends Mainclass1{
   String s;
  public string display(){
   return (super.display()+s);
   }
}
```

Any Questions?