CYBAGE
Delivering Value. Scientifically.

Object Orientation in Java

Author & Presenter -Asfiya Khan
(Senior Technical Trainer)

Agenda

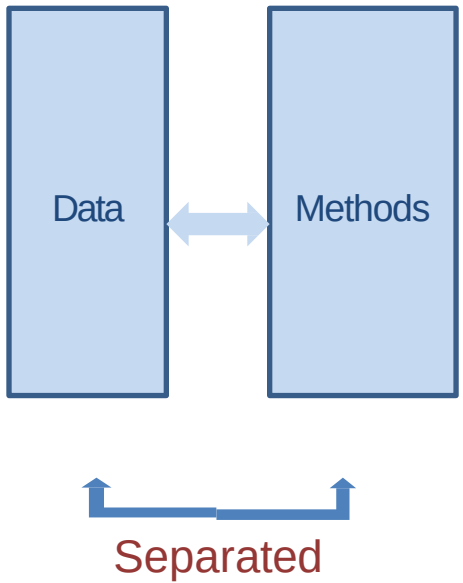
- Need Of OOP
- Procedural Vs OOP
- Object Characteristics
- Major Pillars of OOPS
- Inheritance and Polymorphism
- Abstract Class and Interface
- Object Class
- Lambda Expression

Need Of OOP

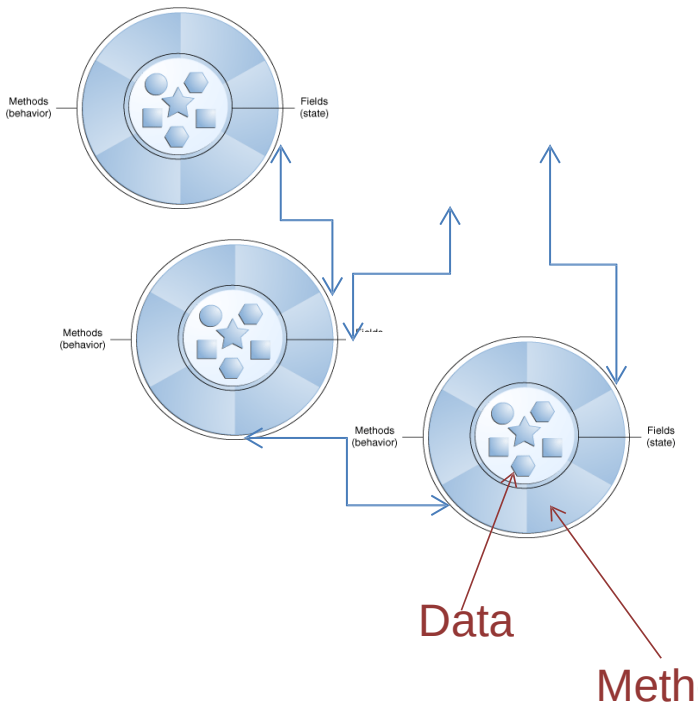
- Impedance mismatch between user of the system & its developer
- Greater Flexibility
- Easy user interface
- Client wants the system to be adaptable & extensible

Procedural VS OOP

Procedural Languages



Object Oriented Language



Object Characteristics

- 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

How the object acts and reacts with respect to state changes.

Example: Bank Account

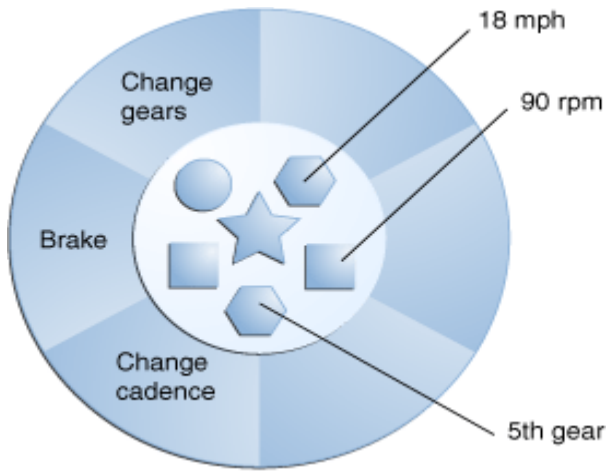
withdraw()

deposit()

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

State and Behavior

Representation of static & dynamic state & behavior of a bicycle



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

Major Pillars of OOPS

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

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

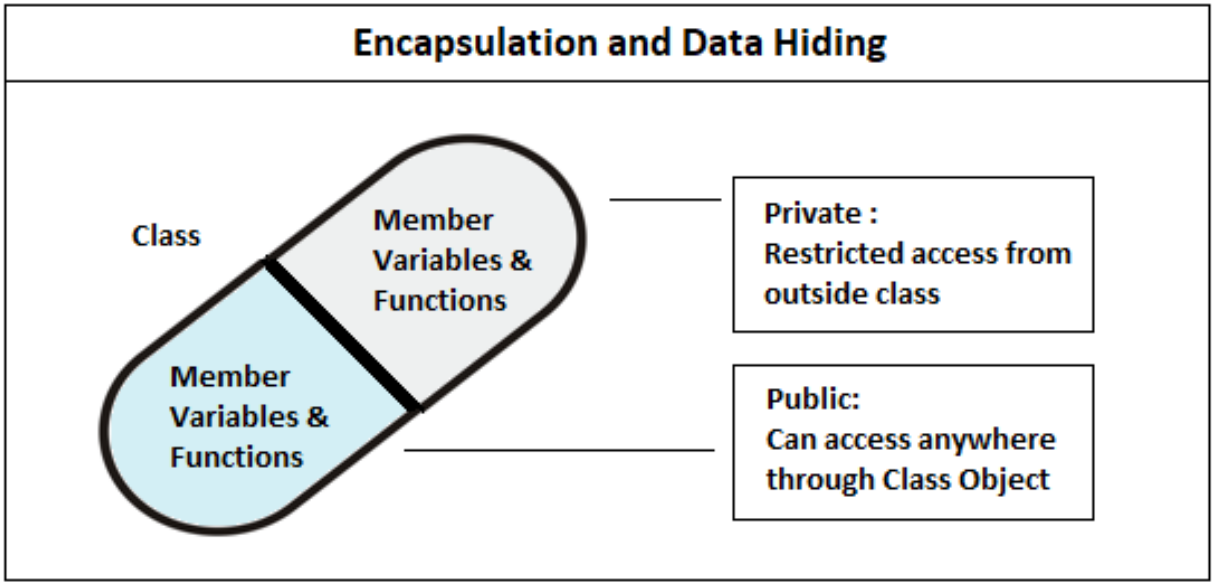
as an Employee	***as a Patient***	***as a Student***
Name	Name	Name
age	age	age
Educational Qualification	------	-------
-------	blood group	------
-------	Medical history	------
-------	--------	batch

- 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.

Encapsulation Contd..



Inheritance

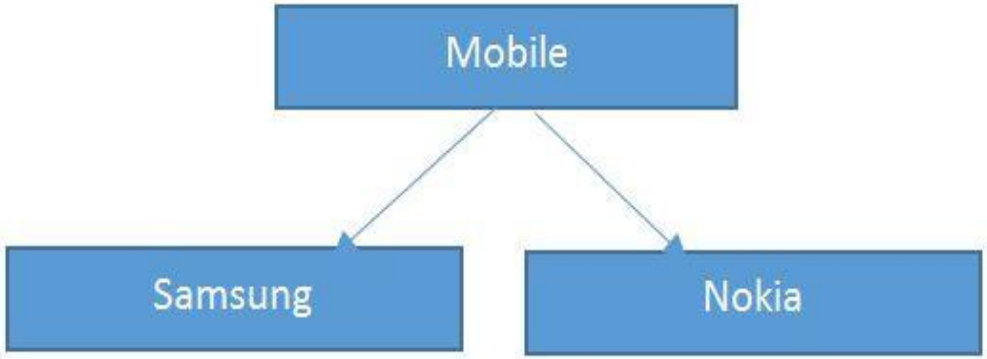
- Inheritance : "*Is-a*"-type of relationship
- Properties of parent are inherited in child.

- 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***

Inheritance Contd..



- 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.

Inheritance Contd..

- 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.

Super Keyword

- 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 Constructor

- 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

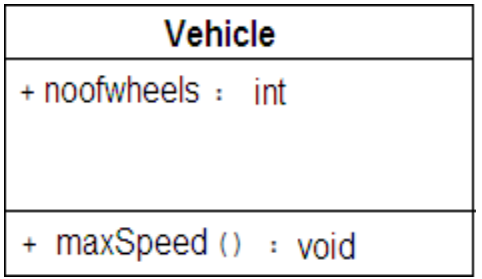
```
class  Car extends Vehicle {

    public Car (int now){
            super(now);
        }
}
```

Subclassing

The Vehicle class is as follows

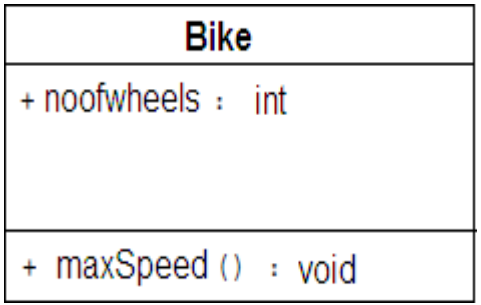
```
public class Vehicle{
public int noofwheels;
public void maxSpeed(){
}
}
```



Subclassing Contd...

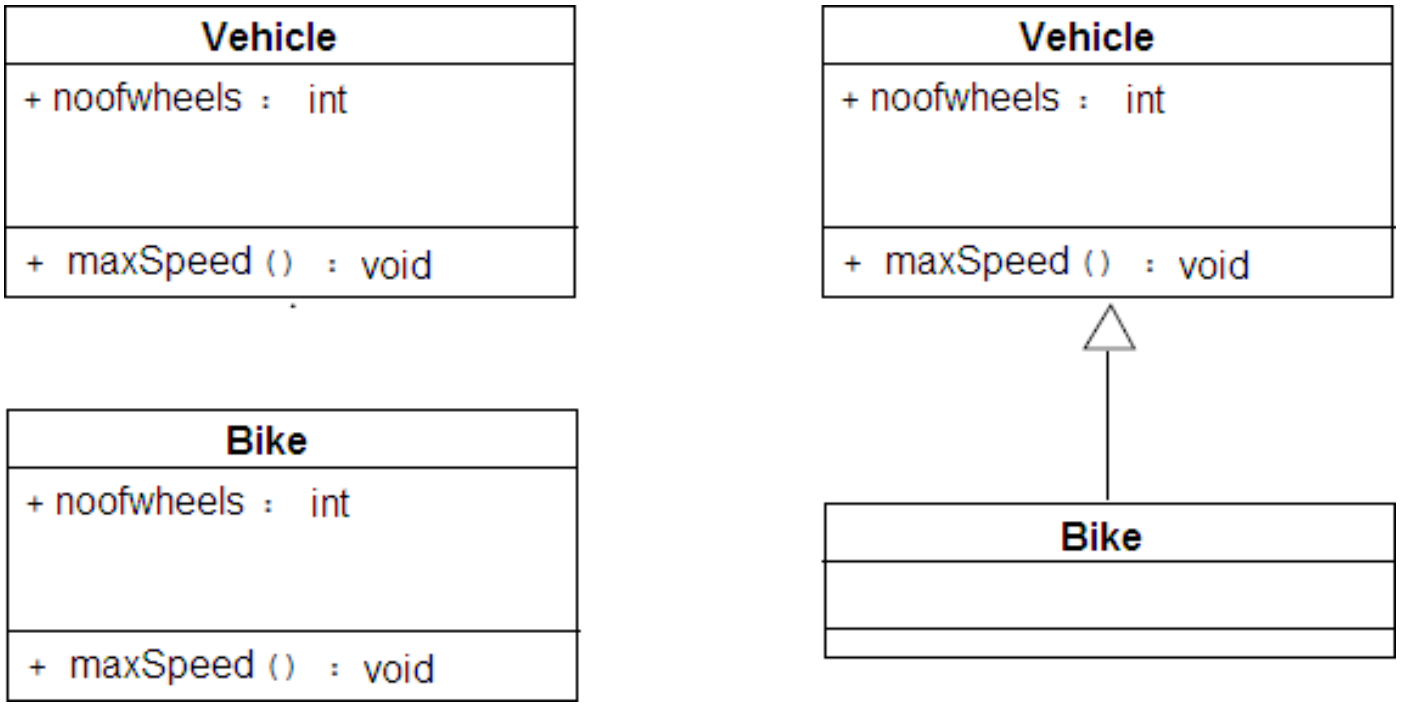
The Bike class is as follows

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



Class Diagram

Class diagram of Vehicle and Bike without and with inheritence



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.

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)

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..

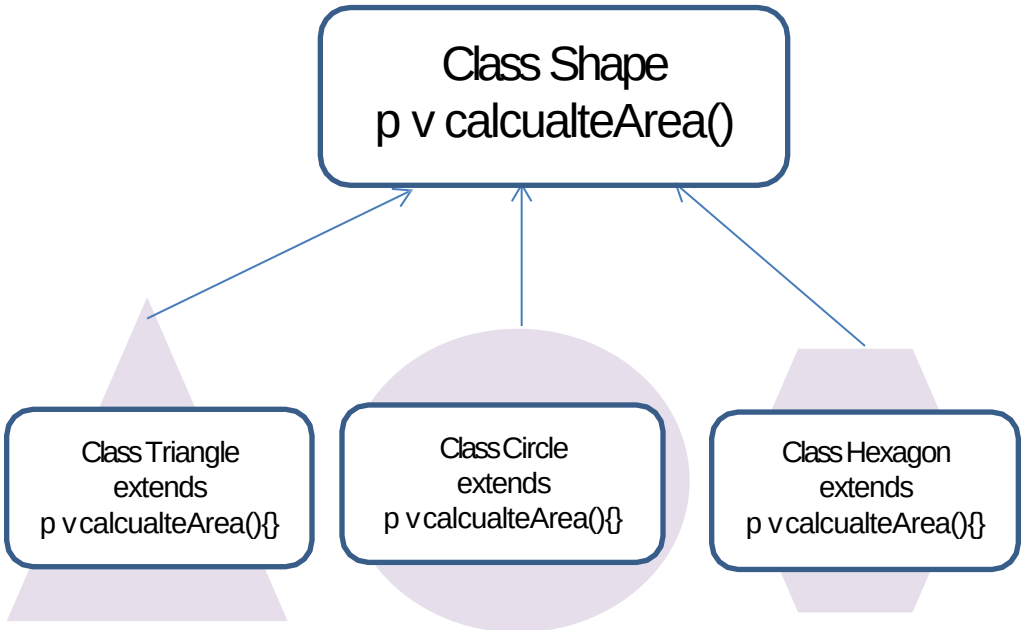
```
class Vehicle
{
   public void showSpeedRange()
    { System.out.println("No range specified")}
}
class   Car extends Vehicle
{
    public void showSpeedRange()
    { System.out.println("Range: 0-300 ")}
}
class   Bike extends Vehicle
{
    public void showSpeedRange()
     { System.out.println("Range: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

Abstract Class



calArea() can be implemented in each subclass i.e. Traingle,Circle, Hexagon

- Method calArea() can not be implemented in Shape class.
- Such a method which is declared but not defined is called an ***abstract*** method
- So class Shape must be declared as abstract & Class Shape should not be instantiated
- Implementation of calArea() can be provided in subclasses i.e. Triangle , Circle, Hexagon

Abstract Class

- **Java Abstract classes** are used to declare common characteristics of subclasses.
- Abstract classes are used to provide a template or design for concrete subclasses down the inheritance tree.
- Abstract classes are declared with the abstract keyword.
- One or more methods in an abstract class are declared but not defined.

Abstract Class Continued..

- Any class containing even a single method as abstract must be declared as abstract
- An abstract class may contain concrete methods as well.
- An abstract class ***can not be*** instantiated
- Sub class of an abstract class must implement all abstract methods of super class or it must also be declared as abstract
- Constructors & static methods can not be declared as abstract

Interface

```
interface Printable
{
      public void print()
}
```

- an *interface* is a reference type, similar to a class, that can contain *only* constants & method signatures
- There are no method bodies inside an interface
- Interfaces cannot be instantiated

Interface

- Methods declared in an interface are by default abstract
- Variables declared in an interface are by default
  public ,static & final
- Interfaces help to club non-related classes under one roof
- Interfaces provide support for multiple inheritance in Java. (They support *dynamic polymorphism* but there is no code reusability)
- Interfaces are some times known as “Programming by contract”. A class that implements the interface is bound to implement all the methods defined in Interface.
- Interfaces provide ‘Lose Coupling’

Interface

- If a class that implements an interface does not define
- All the methods of the interface, then it must be declared abstract and the method definitions must be provided by the subclass that extends the abstract class.
- The interface without any method declaration (empty interface) is called as Marker or Tagging interface.

  Ex: Serializable,Cloneable.

Abstract Class Vs Interface

Abstract Classes	Interfaces
“Is a” type relationship exists between abstract super class & subclass extending it	Interfaces are used to club together Non related classes
Can contain implemented methods as well	Can contain only abstract methods
subclass can not extend from more than one abstract class	A class can implement no. of interfaces

Object Class

- Object class is a cosmic super class
- Object class is the ultimate ancestor
- Every class in java implicitly extends Object

Object Class Methods

toString()

equals(Object o)

hashCode()

finalize()

clone()

wait()

notify()

notifyAll()

toString()

public String toString()

Override this method to represent your class objects textually.

```
class Date
{
    public String toString()
    {
        return ("The date is"+ dd+""+mm+""+ yy);
     }
  }
```

finalize()

- Garbage collector generally reclaims orphaned object spaces
- finalize() gets exceuted just before an object is garbage collected
- Override this method to write clean up code

equals()

- equals() method tests objects for equality.
- As implemented in Object class, it checks whether two references point to same memory location or not.
- Override this method when you want to compare two objects for equality based on their contents (& not on memory location).

hashCode()

- int hashCode()
- As implemented in object class, this method returns memory representation of the object in decimal format.
- ***It is recommended that whenever you override equals(), override hashCode() as well*****.** **(….**WHY?)

Lambda Expression

- Lambda expressions basically express instances of functional interfaces.
- An interface with single abstract method is called functional interface
  example is java.lang.Runnable
- Enable to treat functionality as a method argument, or code as data
- A function that can be created without belonging to any class
- A lambda expression can be passed around as if it was an object and executed on demand.

Lambda Expression

```
interface FuncInterface
{
   void abstractFun(int x);
    default void normalFun()
   {
     System.out.println("Hello");
   }
}
```

```
class Test{

public static void main(String args[]) {
    FuncInterface fobj = (int x)->System.out.println(2*x);
    fobj.abstractFun(5);
}
}
```

Lambda Expression

Zero Parameter

```
() -> System.out.println("Zero parameter lambda");
```

One Parameter

```
(p) -> System.out.println("One parameter: " + p);
```

Multiple Parameters

```
(p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2);
```

CYBAGE
Delivering Value. Scientifically.

Thank You!

www.cybage.com