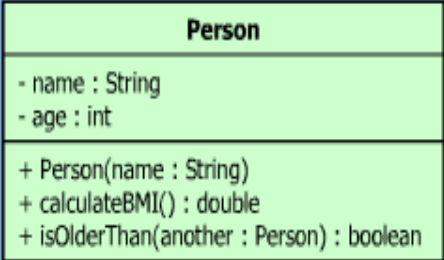# DAY -3&4

Day 3: Advanced Class Features

- Loose coupling and high cohesion
- composition, aggrigation, inheritance, basic of uml
    - Abstract classes and methods
    - Relationship between classes- IS-A, HAS-A, USE-A
    - Interface Vs Abstract, when to use what?
    - Final classes and methods
    - Interfaces, loose coupling and high cohesion
    - SOLID principles, Square – rectangle problem
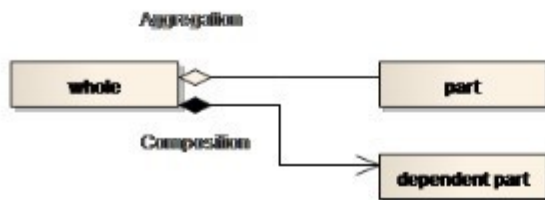   - Hands On & Lab

A bit about UML diagram...

- UML 2.0 aka modeling language has 12 type of diagrams
- Most important once are class diagram, use case diagram and sequence diagram.
- You should know how to read it correctly
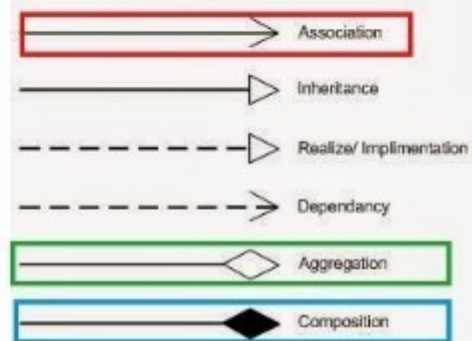- This is not UML session...

UML	Implementation

Person
 	
- name : String
- age : int
 	
+ Person(name : String)
+ calculateBMI() : double
+ isOlderThan(another : Person) : boolean
	`public class Person {`
	` private String name;`
	` private int age;`
	
	` public Person(String name) { ...}`
	
	` public double calculateBMI() { ... }`
	
	` public boolean isOlderThan(Person another) {...}`
	`}`

rgupta.mtech@gmail.com

Relationship between Objects

- USE-A
  - Passanger using metro to reach from office from home
- HAS-A (Association)
  - Compostion
    - Flat is part of Durga apartment
  - Aggregation
    - Ram is musician with RockStart musics group
- IS-A
  - Empoloyee is a person

Aggregation

whole — part

Composition — dependent part

UML Notations:

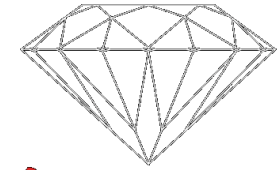|	
——————▷	Association
——————▷	Inheritance
- - - - - -▷	Realize/ Implimentation
- - - - - -▷	Dependancy
——————◇	Aggregation
——————◆	Composition

Inheritance

- **Inheritance is the inclusion of behaviour (i.e. methods) and state (i.e. variables) of a base class in a derived class so** that they are accessible in that derived class.

- **code reusability.**

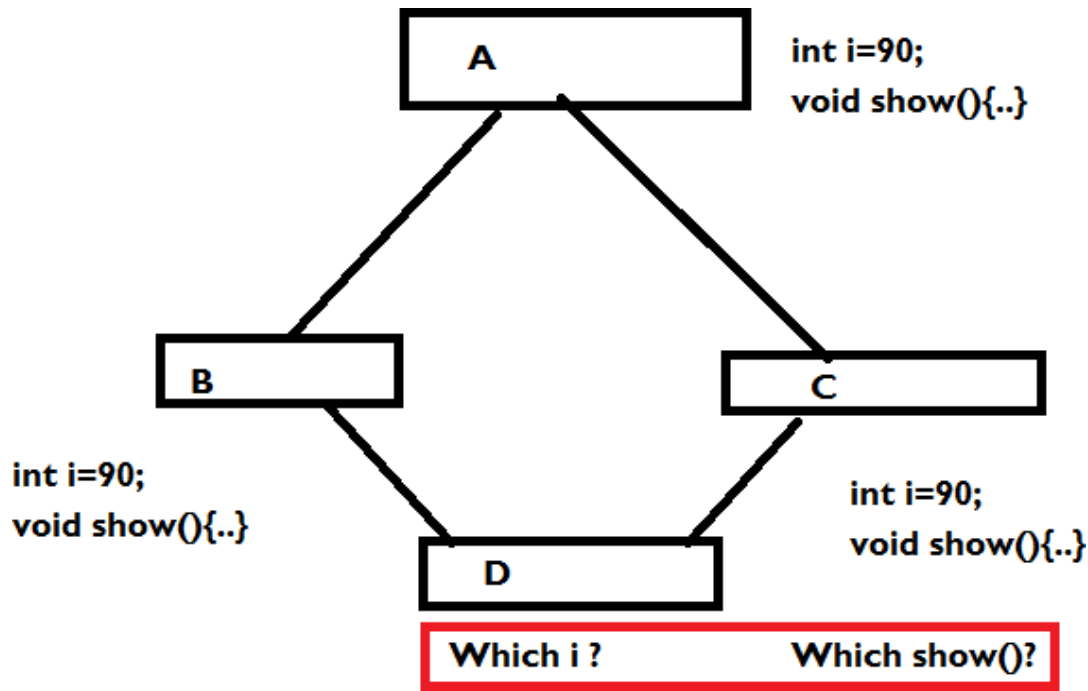- **Subclass and Super class concept**
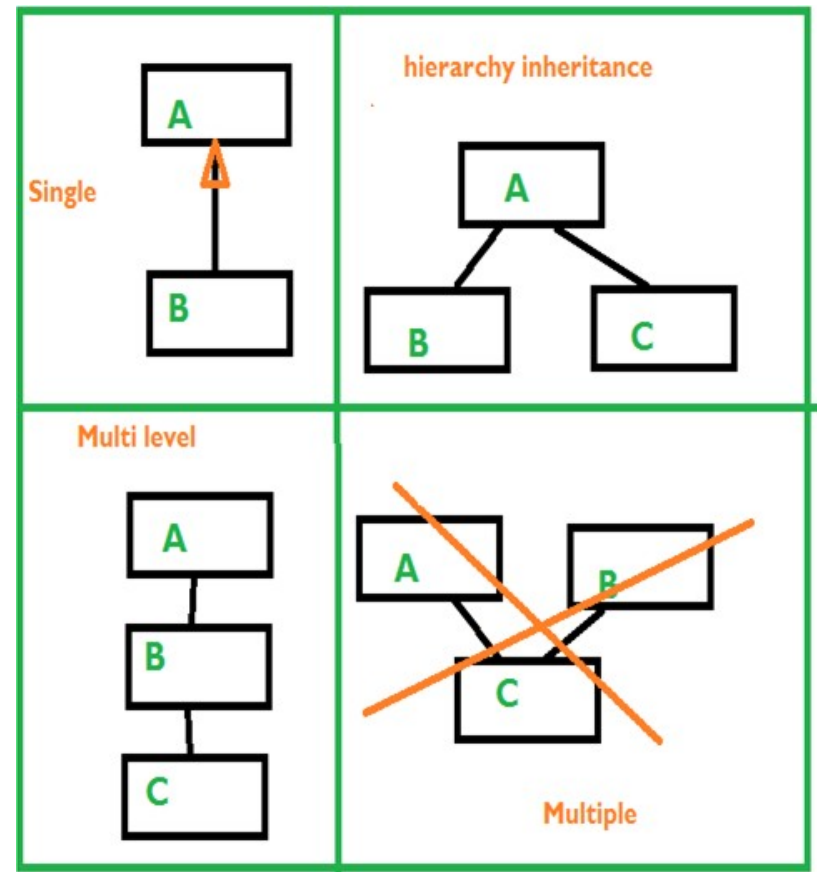
Diamond Problem?

- Hierarchy inheritance can leads to poor design..
- Java don't support it directly...(Possible using interface)

Inheritance example
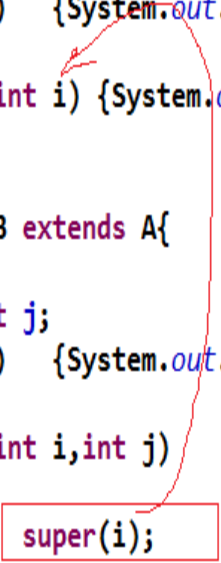
- Use extends keyword
- Use super to pass values to base class constructor.

```
class A{
    int i;
    A()    {System.out.println("Default ctr of A");}

    A(int i) {System.out.println("Parameterized ctr of A");}
}


class B extends A{

    int j;
    B()    {System.out.println("Default ctr of B");}

    B(int i,int j)
    {
        super(i);
        System.out.println("Parameterized ctr of B");
    }

}
```

Overloading

❖ Overloading deals with multiple methods in the same class with the same name but different method signatures.

```
class MyClass {
    public void getInvestAmount(int rate) {…}

    public void getInvestAmount(int rate, long principal)
    { … }
}
```

- **Both the above methods have the same method names but different method signatures, which mean the methods are overloaded.**
- **Overloading lets you define the same operation in different ways for different data.**
- **Constructor can be overloaded ambiguity**
- ***Overloading in case of var-arg and Wrapper objects**…

Overriding...

- Overriding deals with two methods, one in the parent class and the other one in the child class and has the same name and signatures.

- Both the above methods have the same method names and the signatures but the method in the subclass *MyClass* overrides the method in the superclass *BaseClass*

- Overriding lets you define the **same operation in different ways for different object types.**

```
class BaseClass{
    public void getInvestAmount(int rate) {…}
}

class MyClass extends BaseClass {
    public void getInvestAmount(int rate) { …}
}
```

Polymorphism

❖ Polymorphism=many forms of one things

❖ **Substitutability**

❖ **Overriding**
❖ **Polymorphism means the ability of a single variable of a given type to be used to reference objects of** different types, and automatically call the method that is specific to the type of object the
❖ variable references.

Polymorphism

- Every animal sound but differently...
- We want to have Pointer of Animal class assigned by object of derived class

Animal a=new Bird();

ref of
base class

object of drived
class
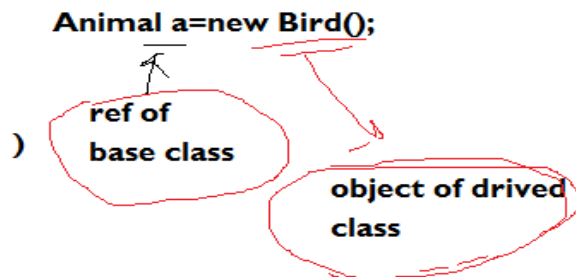
Which method is going is to be
called is not decided by the type of
pointer rather object assigned will
decide at run time
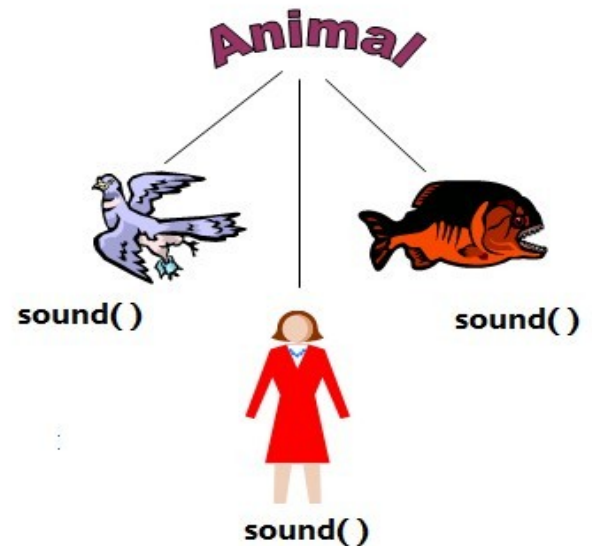
Animal

sound( )   sound( )

sound( )

Example...

```java
class Animal
{
    public void sound()
    {
        System.out.println("Don't know how generic animal sound.....");
    }
}
class Bird extends Animal
{@Override
    public void sound()
    {
        System.out.println("Bird sound.....");
    }
}
class Person extends Animal
{@Override
    public void sound()
    {
        System.out.println("Person sound.....");
    }
}
```

rgupta.mtech@gmail.com

Need of abstract class?

- Sound( ) method of Animal class don't make any sense …i.e. it don't have semantically valid definition

- Method sound( ) in Animal class should be abstract means incomplete

- Using abstract method Derivatives of Animal class forced to provide meaningful sound() method

```java
class Animal
{
    public void sound()
    {
        System.out.println("Don't know how generic animal sound.....");
    }
}
class Bird extends Animal
{@Override
    public void sound()
    {
        System.out.println("Bird sound.....");
    }
}
class Person extends Animal
{@Override
    public void sound()
    {
        System.out.println("Person sound.....");
    }
}
```

Abstract class

- If an class have at least one abstract method it should be declare abstract class.

- Some time if we want to stop a programmer to create object of some class…

- Class has some default functionality that can be used as it is.

- Can extends only one abstract class

```
class Foo{
    public abstract void foo();
}
```

```
abstract class Foo{
    public abstract void foo();
}
```

```
abstract class Foo{
    }
```

Abstract class use cases...

❖ Want to have some default functionality from base class and class have some abstract functionality that can't be define at that moment.

```
abstract class Animal
{
        public abstract void sound();
        public void eat()
        {
                System.out.println("animal eat...");
        }
}
```

▫Don't want to allow a programmer to create object of an class as it is too generic

Interface **vs** **abstract class**

More example...

```java
public abstract class Account {
    public void deposit (double amount) {
        System.out.println("depositing " + amount);
    }

    public void withdraw (double amount) {
        System.out.println ("withdrawing " + amount);
    }

    public abstract double calculateInterest(double  amount);
}
```

```java
public class SavingsAccount extends Account {

    public double calculateInterest (double amount) {
        // calculate interest for SavingsAccount
        return amount * 0.03;
    }

    public void deposit (double amount) {
        super.deposit (amount); // get code reuse
        // do something else
    }

    public void withdraw (double amount) {
        super.withdraw (amount); // get code reuse
        // do something else
    }
}
```

```java
public class TermDepositAccount extends Account {

    public double calculateInterest (double amount) {
        // calculate interest for SavingsAccount
        return amount * 0.05;
    }

    public void deposit(double amount) {
        super.deposit (amount); // get code reuse
        // do something else
    }

    public void withdraw(double amount) {
        super.withdraw (amount); // get code reuse
        // do something else
    }
}
```

rgupta.mtech@gmail.com

Final

- ❖ What is the meaning of final
  - ❖ Something that can not be change!!!
- ❖ final
  - ❖ Final method arguments
    - ❖ Cant be change inside the method
  - ❖ Final variable
    - ❖ Become constant, once assigned then cant be changed
  - ❖ Final method
    - ❖ Cant overridden
  - ❖ Final class
    - ❖ Can not inherited (Killing extendibility)
    - ❖ Can be reuse

Some examples....

Final class

- Final class can't be subclass i.e.   Can't be extended
- No method of this class can be overridden
- Ex: String class in Java…

Real question is in what situation somebody
should declare a class final

```
package cart;

public final class Beverage{

    public void importantMethod(){
        Sysout("hi");
    }
}


------------------------------------------------
package examStuff;
import cart.*;

class Tea extends beverage{

}
```
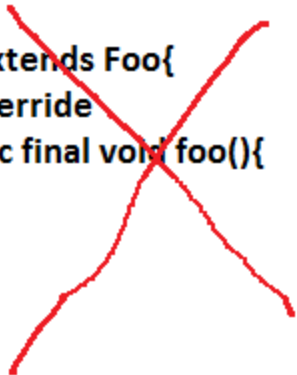
Final Method

- ❖ Final Method Can't overridden
- ❖ Class containing final method can be extended
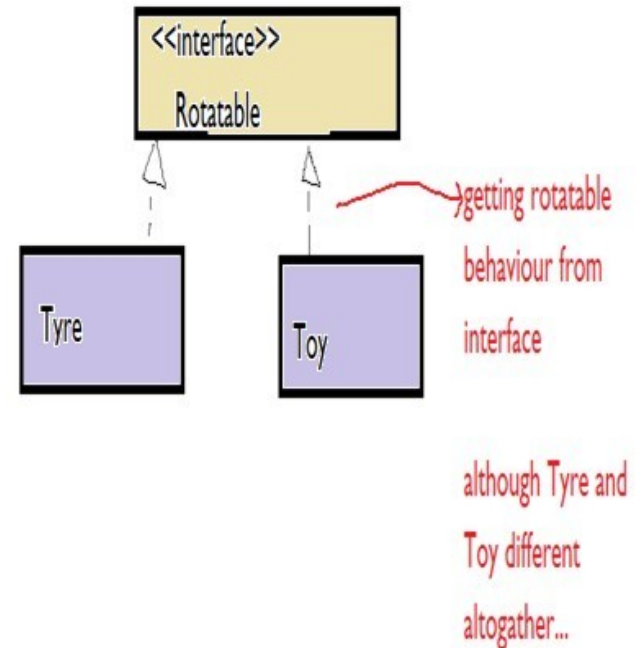- ❖ Killing substitutability

```
class Foo{

        public final void foo(){
                Sysout("i am the best");
                Sysout("You can use me but can't override me");
        }
};

class Bar extends Foo{
        @Override
        public final void foo(){



        }
}
```

Interface?

- Interface : Contract bw two parties
- Interface method
    - Only declaration
    - No method definition
- Interface variable

    - Public static and final constant

    - Its how java support global constant Break the hierarchy

- Solve diamond problem
- Callback in Java* Some Example ....

<<interface>>

Rotatable

Tyre   Toy

getting rotatable behaviour from interface

although Tyre and Toy different altogather...

Interface?

❖ Rules
  ❖ All interface methods are always public and abstract, whether we say it or not.
  ❖ Variable declared inside interface are always public static and final Interface method can't be static or final
  ❖ Interface cant have constructor
  ❖ An interface can extends other interface Can be used polymorphically
  ❖ An class implementing an interface must implement all of its method otherwise it need to declare itself as an abstract class…

Implementing an interface...

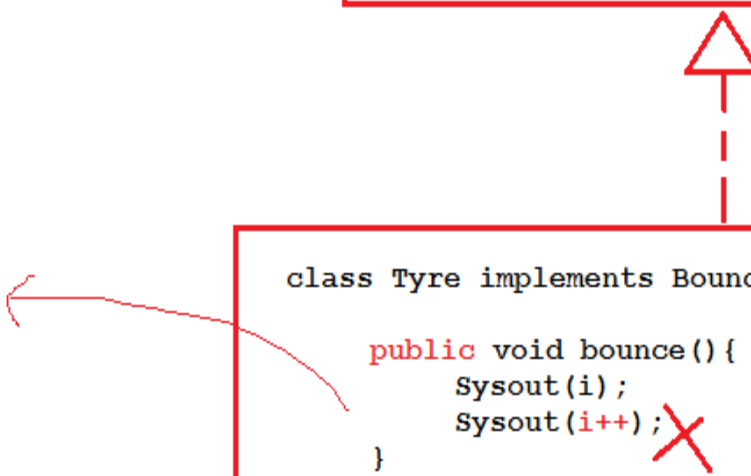What we declare

```
interface Bouncable{
    int i=9;
    void bounce();
    void setBounceFactor();
}
```

What compiler think...

```
interface Bouncable{
    public static final int i=9;
    public abstract void bounce();
    public abstract void setBounceFactor();
}
```

All interface method must be implemented....

```
class Tyre implements Bouncable{

    public void bounce(){
        Sysout(i);
        Sysout(i++);
    }
    public void setBounceFactor(){}
}
```

rgupta.mtech@gmail.com

Note

Following interface constant declaration are identical
- int i=90;
- public static int i=90;
- public int i=90;
- public static int i=90;
- public static final int i=90;

Following interface method declaration don't compile
- final void bounce();
- static void bounce();
- private void bounce();
- protected   void bounce();

rgupta.mtech@gmail.com

Decreasing Rigidity..increasing Flexibility
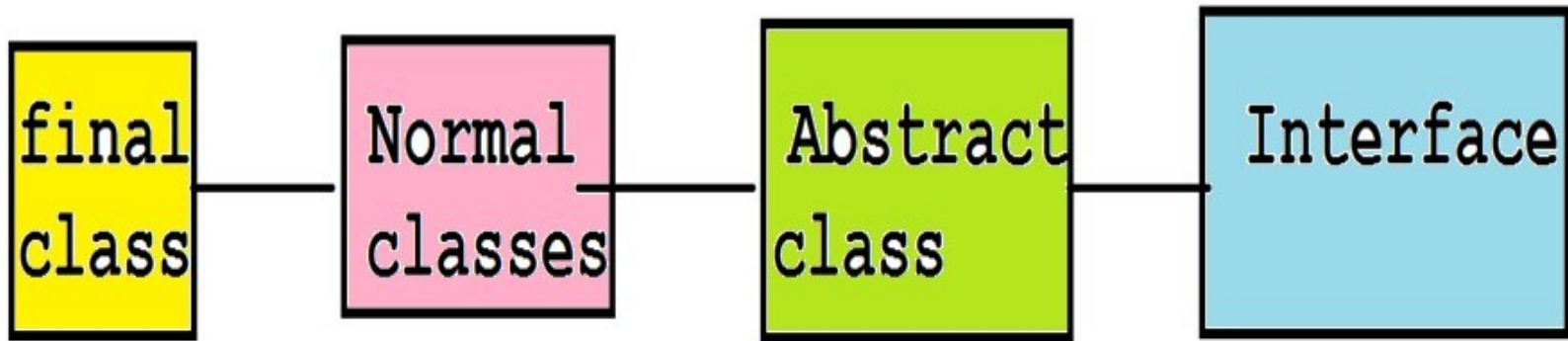
decreasing rigidity...increasing
flexibility...........

Type of relationship bw objects

- ❖ USE-A
- ❖ HAS-A
- ❖ IS-A (Most costly ?)
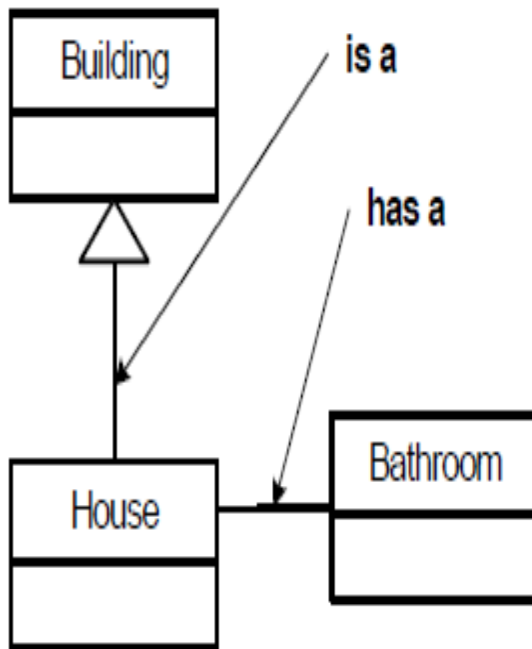
ALWAYS GO FOR LOOSE COUPLING AND HIGH

COHESION…

But HOW?

IS-A VS HAS-A

Inheritance [is a] Vs Composition [has a]

Building

is a

has a

House

Bathroom

is a [House is a Building]

```
class Building{

.......

}


class House extends Building{

..........

}
```

has a [House has a Bathroom]

```
class House {
   Bathroom room = new Bathroom() ;

   ....
   public void getTotMirrors(){
      room.getNoMirrors();

      ....
   }

}
```

rgupta.mtech@gmail.com