Object Oriented Paradigm

Object Oriented Paradigm

- Object oriented Paradigm has Object Oriented Principles
- **Object Oriented Principles**
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Cohesion
  - Coupling
- **Programming practices**
  - Data abstraction
  - Data Handling

Object Oriented Principles

- **Encapsulation:**
  - binding data members with member functions of object.
  - **binding properties with behaviours of object.**
- **Inheritance:**
  - Creation of new object by acquiring common properties and behaviours and extending behaviour of parent object if required.
  - **Generalization to Specialization**

Object Oriented Principles

- **Polymorphism:**
  - many forms of same thing
  - **different behaviour for different caller**
- **Coupling:**
  - Interaction between different objects(Message passing)
  - Coupling should be **low** ideally
- **Cohesion:**
  - Interaction within object
  - Cohesion should be **high**(Self sufficient object)

Programming practices

- **Data Abstraction:**
  - Knowing required details about Object
  - **Never details about how the object performs behaviors**

- **Data Hiding:**
  - Controlling the accessibility of objects data(Properties and behaviors)
  - Hiding details from outside world.

Code re-use techniques in Java

- Association (has-a relationship)
  - Using existing objects/functionality
- Inheritance (is-a relation)
  - Code reuse and extension

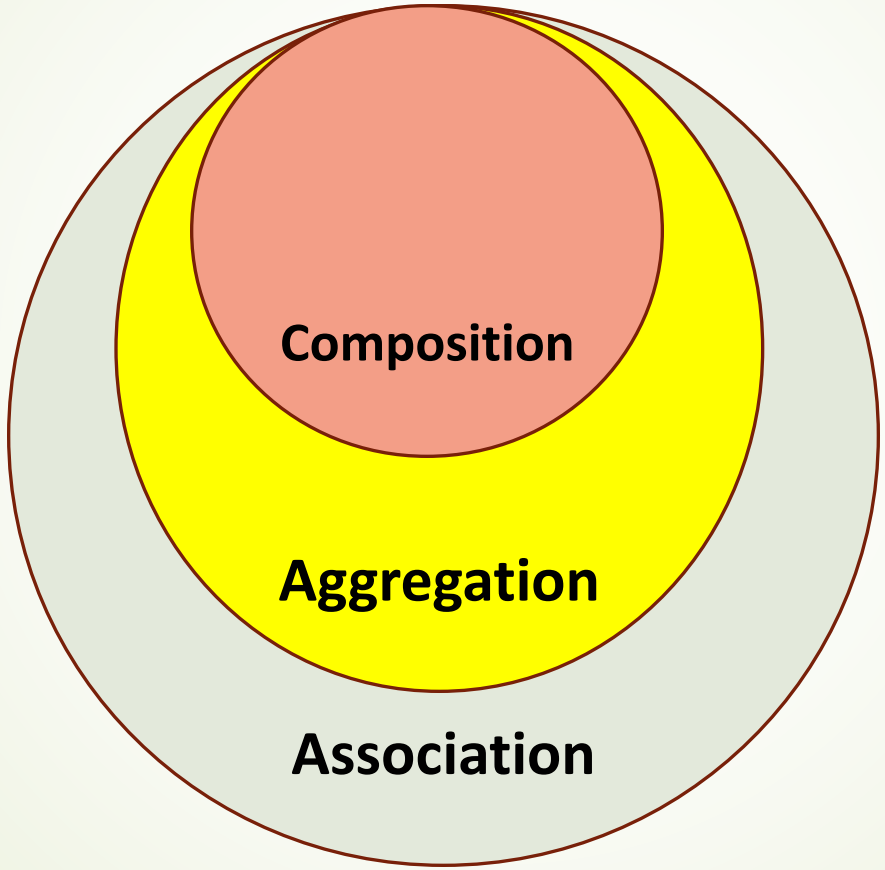
Association

- **One class contains one or more references/objects of other classes. (class has an object from another class as a data member)**
- **Association is also called containment**
- **Through Association we can use existing functionality AS-IS.**
- **Association is specialized to Aggregation and Aggregation specialized to Composition**

Examples

- Person has Addresses
- Student has Certificates
- Person has AdharCard

Association, Aggregation and Composition



Association

- **It defines has- relationship between objects**
- **Define Multiplicity between objects**
- **Association can be used for implementing one-to –one, one-to-many and many-to-many kinds of relationship between objects.**

Example. Car and Driver relationship

Aggregation

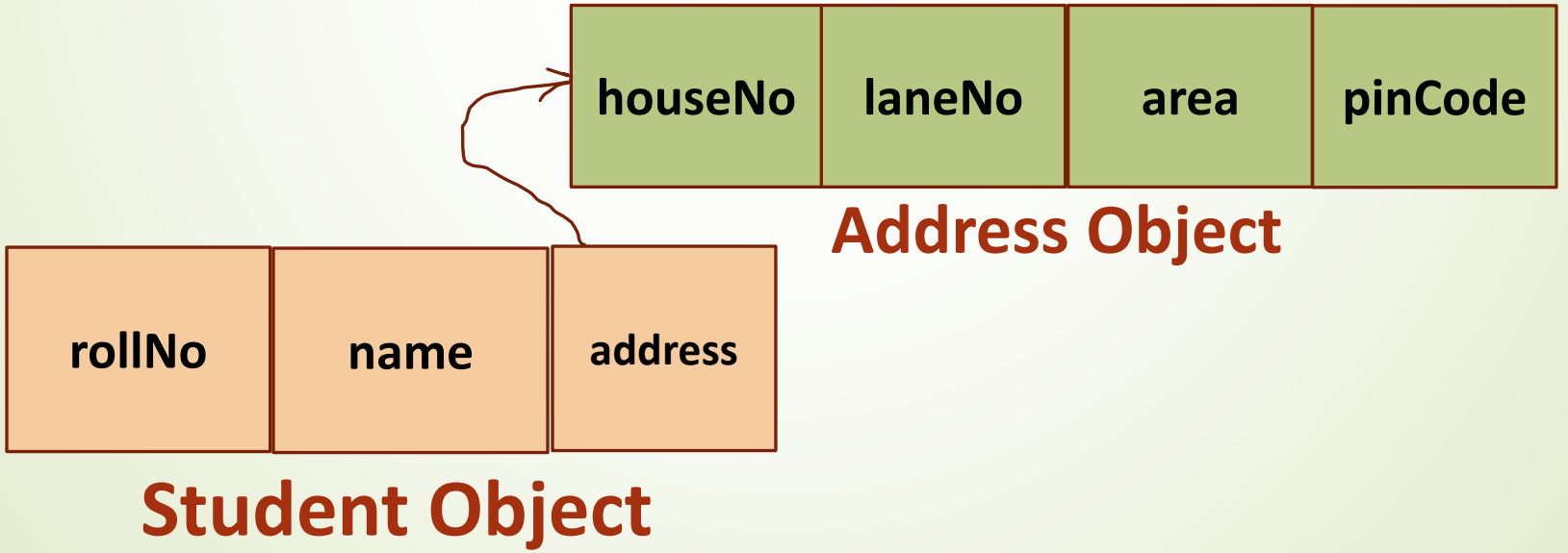
- It is also has-a relationship
- It is special case of Association
- Define directional has-a relationship between objects
- Direction has to be specified that which object contains which object
  - Example: **Car has Engine, Course has Students**

Composition

- It is also has-a relationship
- Restricted Aggregation is called **Composition**
- Directional has-a relationship between objects
- If one object contains another object, if the contained objects does not exists without existence of container object.
  - Example : **Car has price**

Steps to implement Association

- Create **Address** class with fields like **houseNo, laneNo, area , pinCode** etc.
- Create **Student** class with fields like **rollNo, name** and Create reference/object of **Address** class(has-relationship)
- Student class object can be represented as



Inheritance (is-a relationship)

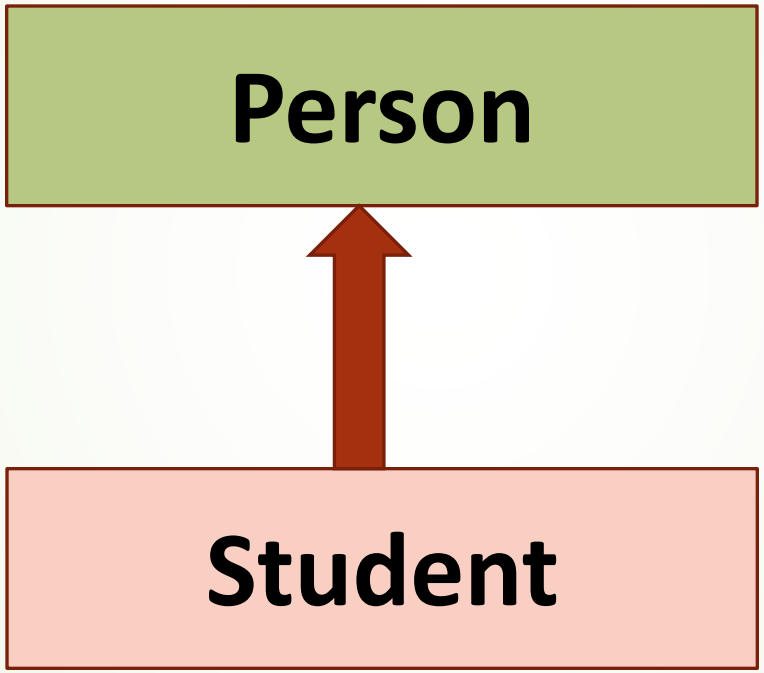
- ***Inheritance allows one class to acquire properties and behaviors of another class***.
- Inheritance establishes is-a relationship between objects.
- The class whose properties & behaviors are inherited in another class is called as **Base or Parent or Super** class.
- The class which inherits properties & behaviors another class is called as **Derived or Child or Sub** class.
- Runtime Polymorphism can not be done without Inheritance.

Types of Inheritance

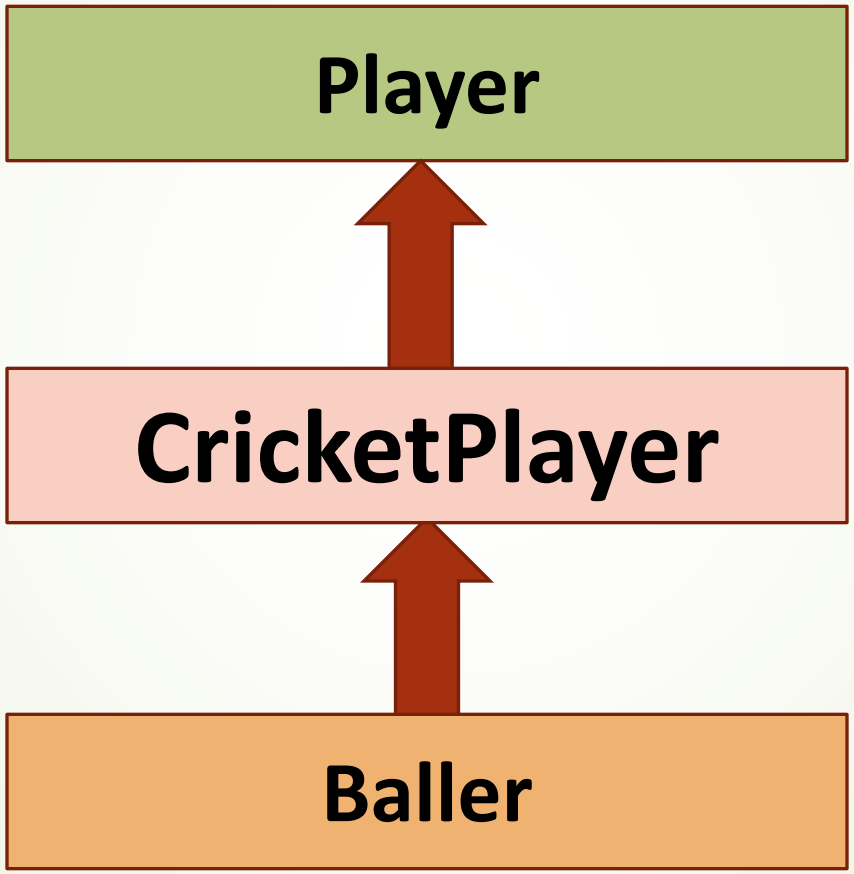
- There are 5 major types of inheritance

1. **Single Inheritance**
2. **Multilevel Inheritance**
3. **Hierarchical Inheritance**
4. **Multiple Inheritance**
5. **Hybrid Inheritance**

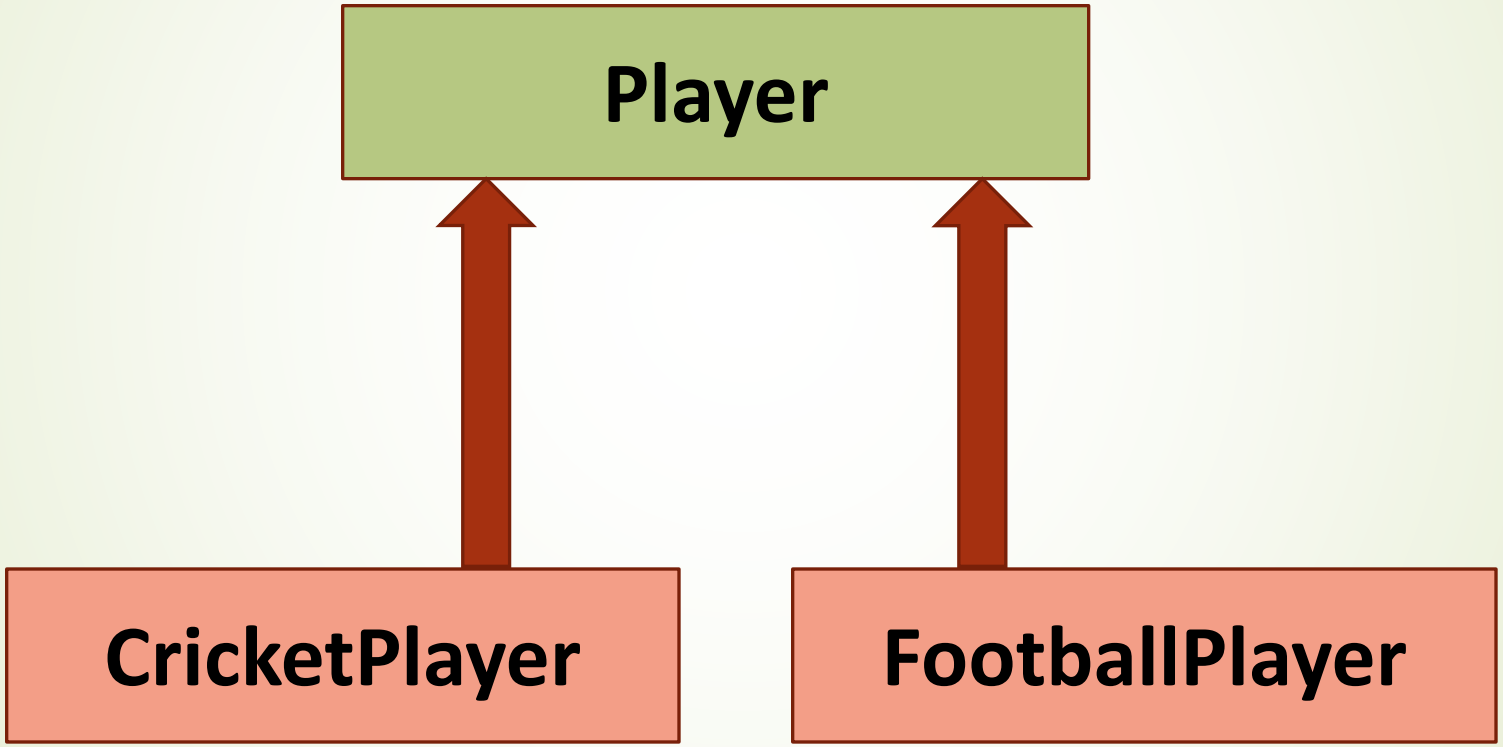
Single Inheritance



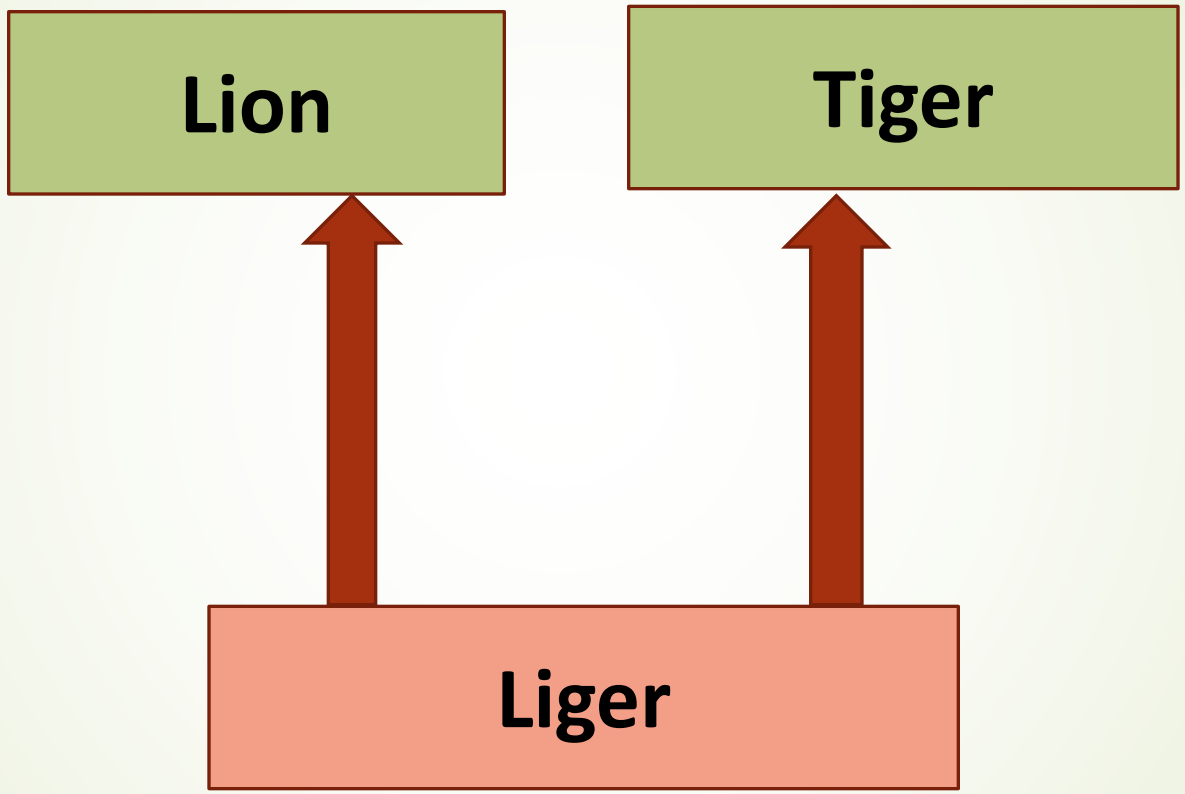
Multilevel Inheritance



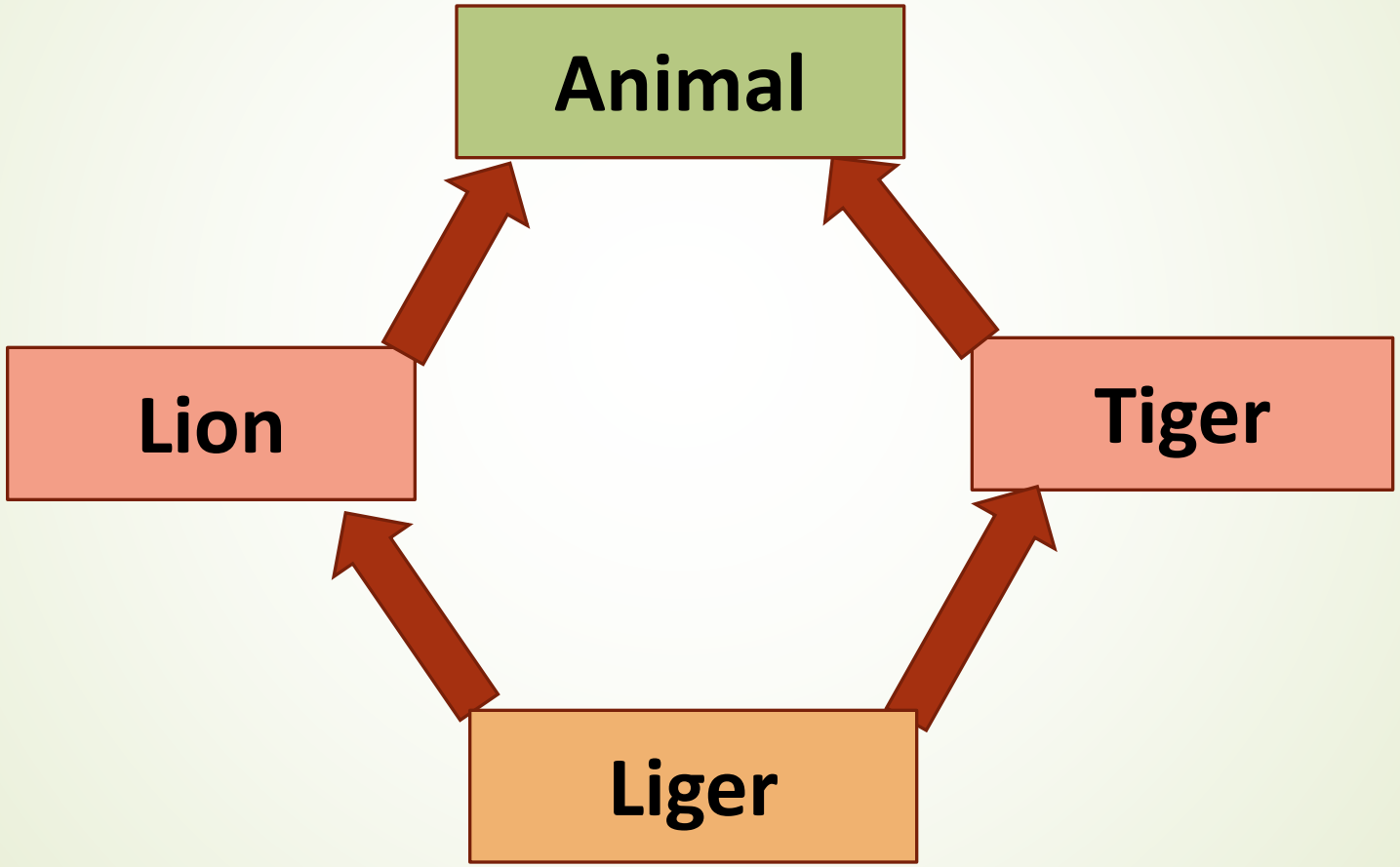
Hierarchical Inheritance



Multiple Inheritance



Hybrid Inheritance(Diamond)



Inheritance Syntax

Inheritance syntax

public class <Derived class Name> ***extends*** <Base class Name> {

// Instance variables/methods

}

Example:

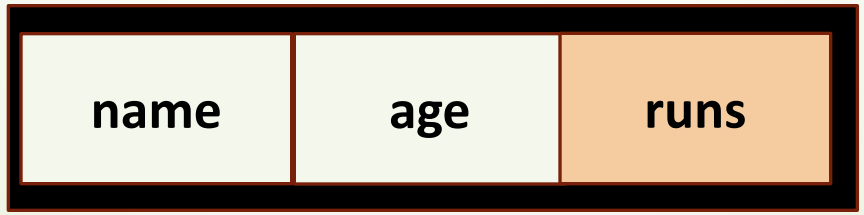
```
public class  CricketPlayer extends Player {
// Instance variables/methods
}
```

Extends keyword denote inheritance

Implementing Inheritance Example

- Steps to be followed

1. Create class **Player** with **name** and **age** .
2. Create class **CricketPlayer** by extending/inheriting **Player** class. Add **runs** as extra instance data member
3. Create object of **Player** and **CricketPlayer** class and call methods
4. Object of CricketPlayer can be represented as



Access specifiers for data members and methods

Access Specifier	Same Class	Same Package	Sub-class or child class outside package	Outside class, package and sub-class
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
default	Yes	Yes	No	No
private	Yes	No	No	No

Constructor Calls in Inheritance (chaining)

- If Derived class object is created, first Base class constructor and then Derived class constructor gets called
- If base class has only parameterized constructor, it can not be called automatically hence programmer needs to call it explicitly (To avoid this situation, we can implement parameterized as well as default constructor in base class)

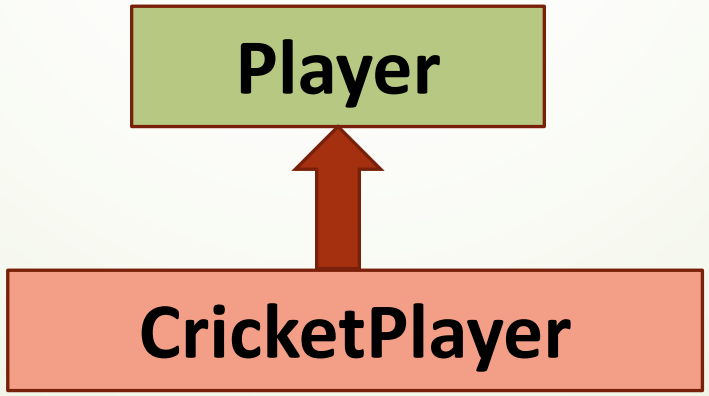
Constructor Calls in Inheritance

Why Base class constructor gets called in inheritance?

Constructor is object specific and to initialize base class object within Derived class object constructor call is required.

Up casting

- **Up casting** is using the Super class reference to refer the **sub-class** object
- Conversion of sub-class reference into its super class reference is called **Up casting.**



Polymorphism

- Polymorphism means many forms of same thing
- There are two type of Polymorphism
- Static Polymorphism (Early binding)
  - Achieved through Method Overloading
- Runtime Polymorphism (Late or runtime binding)
  - Achieved through Method Overriding & late binding
  - Methods are called using Dynamic Method dispatch

Method Overriding

- When both super class and sub-class have instance method with same signature but different implementation is called as **Method Overriding**
- **Extends behavior** of parent class method in child class
- Requirements for Method Overriding
  - **Inheritance** is must for method overriding
  - Same signature means **same return type, parameter list and name** of method
  - **Static** and **final** methods can not be overridden as they are not inherited in child class

Method Overloading vs Method Overriding

Polymorphism --changing forms of behaviour

static polymorphism -- detected by javac(early binding) Via method overloading

1.Can exist in same class or in inh hierarchy.

2. same method name,ret type ignored

3. signature - different (no/type/or both)

4. No rules on access specifiers

5. No rules regarding exc handling

Dynamic form of polymorphism (late binding)

1.can exist only in inh hierrachy Overriding method

2.Must have same name,same signature,ret type can be either same or sub type of the super class method ret type(co variance)

4.Overridng form of the method must either same access specifier or wider.

5. Overriding form of the method can't throw any NEW or wider checked excs.

Virtual Methods and dynamic method dispatch

- **Dynamic method dispatch** is achieved through virtual methods
- All **instance methods** are by default **virtual** in Java
- Method calls are resolved at runtime based on type of object but not considering type of reference

Abstract class and Interface

- Abstract class is class which has at least one abstract method or class declared as abstract
- abstract keyword can be applied to methods and classes
- Abstract class is incomplete which has common functionality implemented and some functionality is unimplemented **(incomplete type)**
- Interface has all methods are **public and abstract**
- **Object of Abstract class & Interface can not be created.**