Association and Inheritance

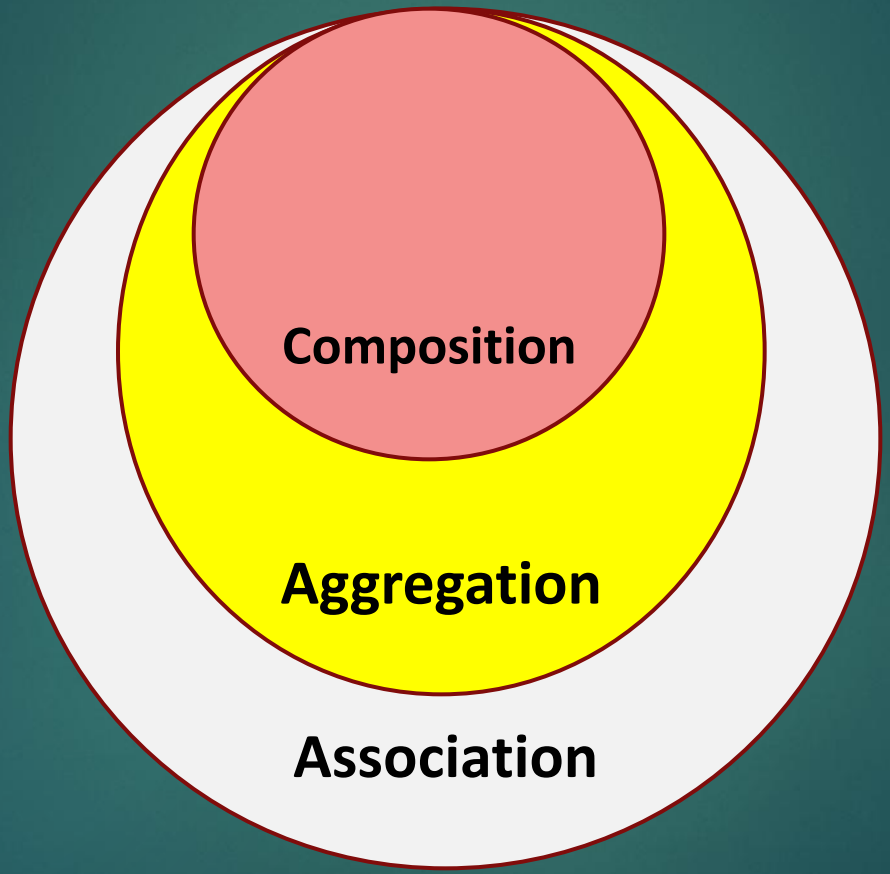
Code re-use techniques in C++

- Functions
- Association (has-a relationship)
- Inheritance (is-a relation)
  - **Code reuse and extension**
- Templates
  - Class template
  - Function template

Association

- One class contains one or more 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. (Cardinality and Modality)
- Example. Car and Driver relationship

Aggregation

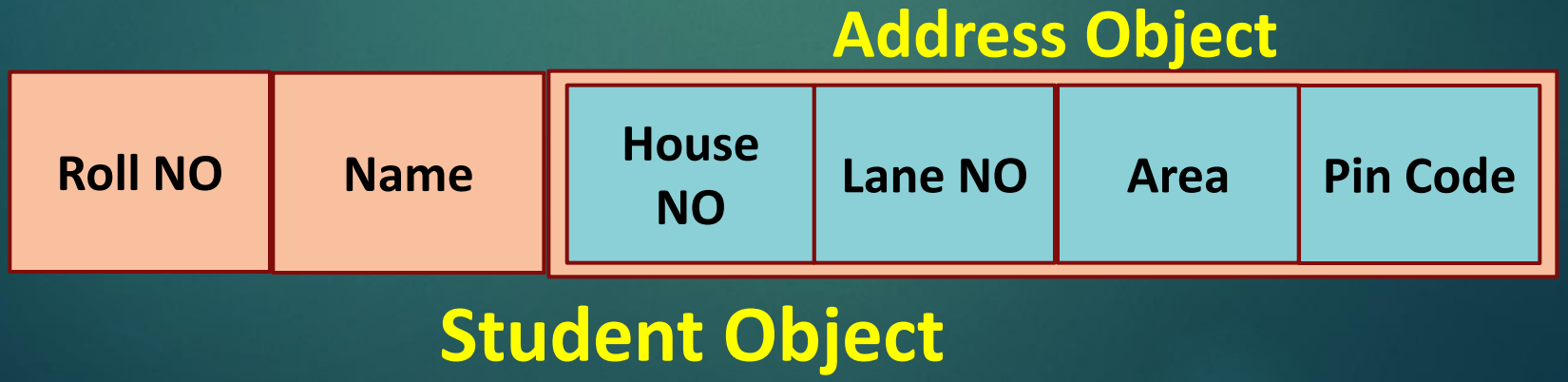
- It is also has-a relationship
- It is special case of Association
- Define directional has-a relationship between objects
- **Direction has to be specified that which object contains which object**
- Example: Car has Engine, Course has Students

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.**
- Car has price

Steps to implement Association

- Create Address class with fields like House No, Lane No, Area , Pin Code etc.
- Create Student class with fields like Roll No, Name and Create Object of Address class(has-relationship)
- Student class object can be represented as



Inheritance (is-a relationship)

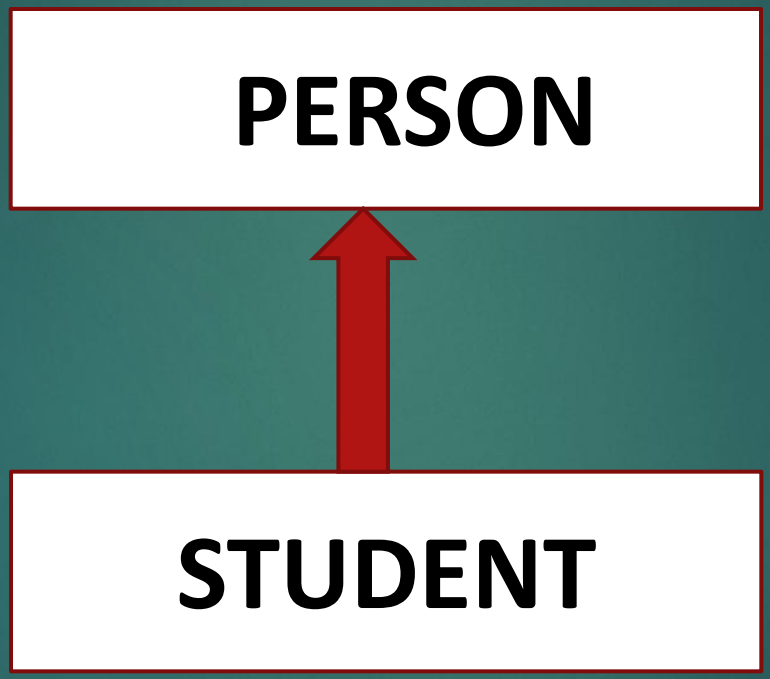
- ***Inheritance is capability of one class to acquire properties and behaviors of another class***.
- Inheritance establishes is-a relationship between objects.
- The class whose properties & behaviors are inherited in another class is called as **Base or Parent or Super** class.
- The class which inherits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

Types of Inheritance

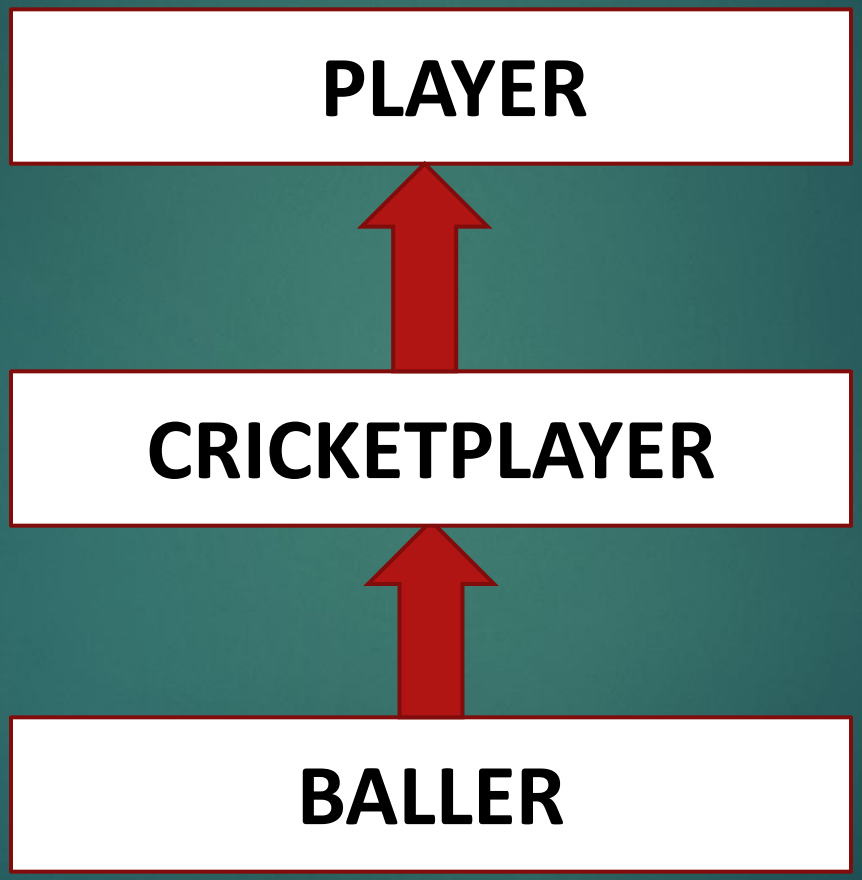
- There are 5 different types of inheritance

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

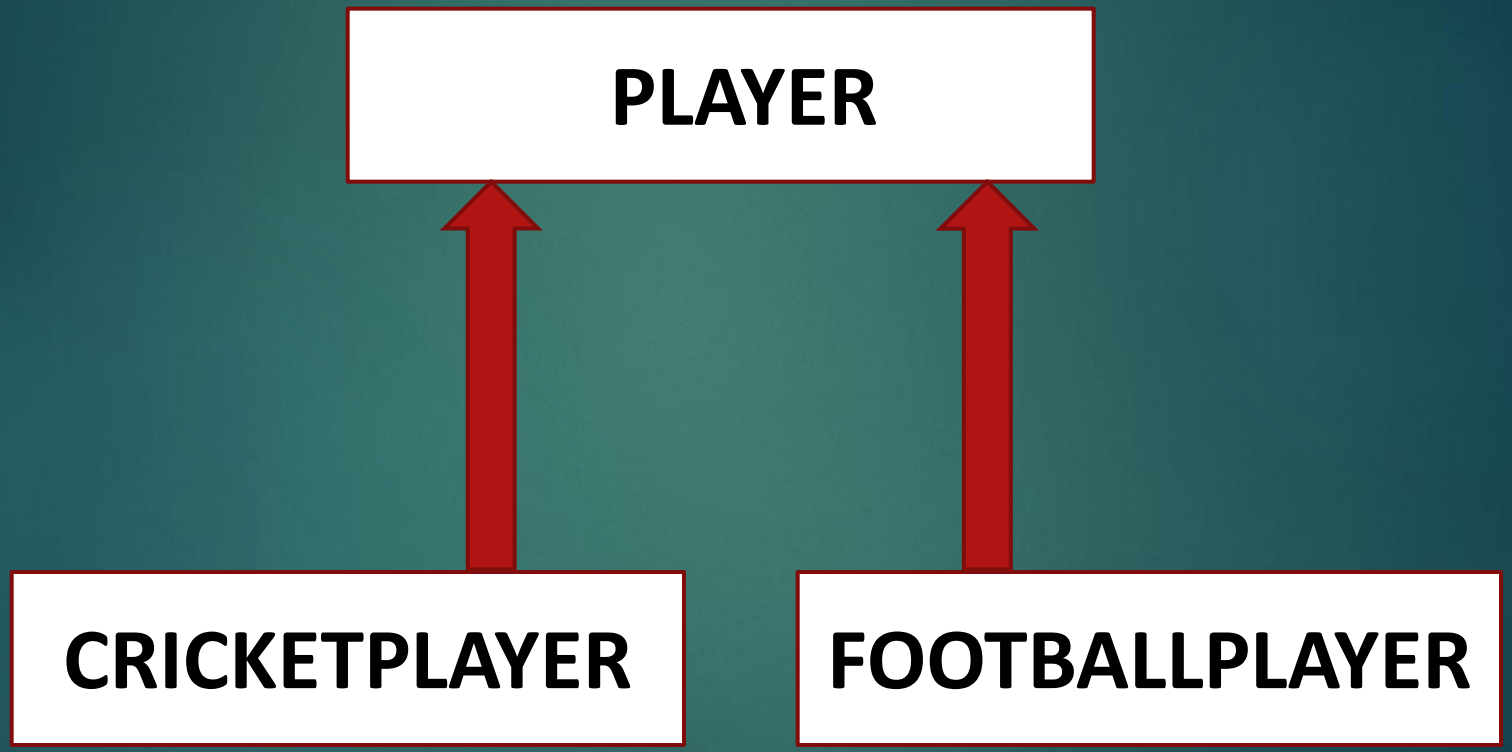
Single Inheritance



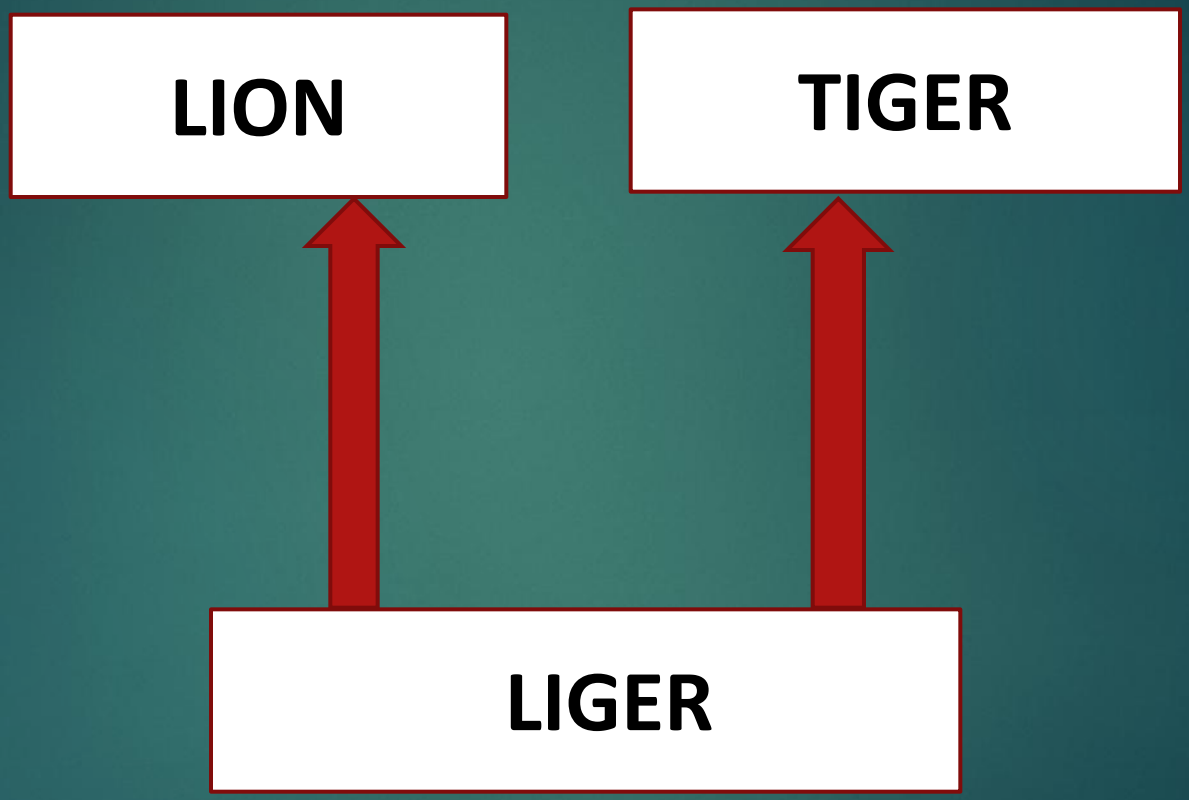
Multilevel Inheritance



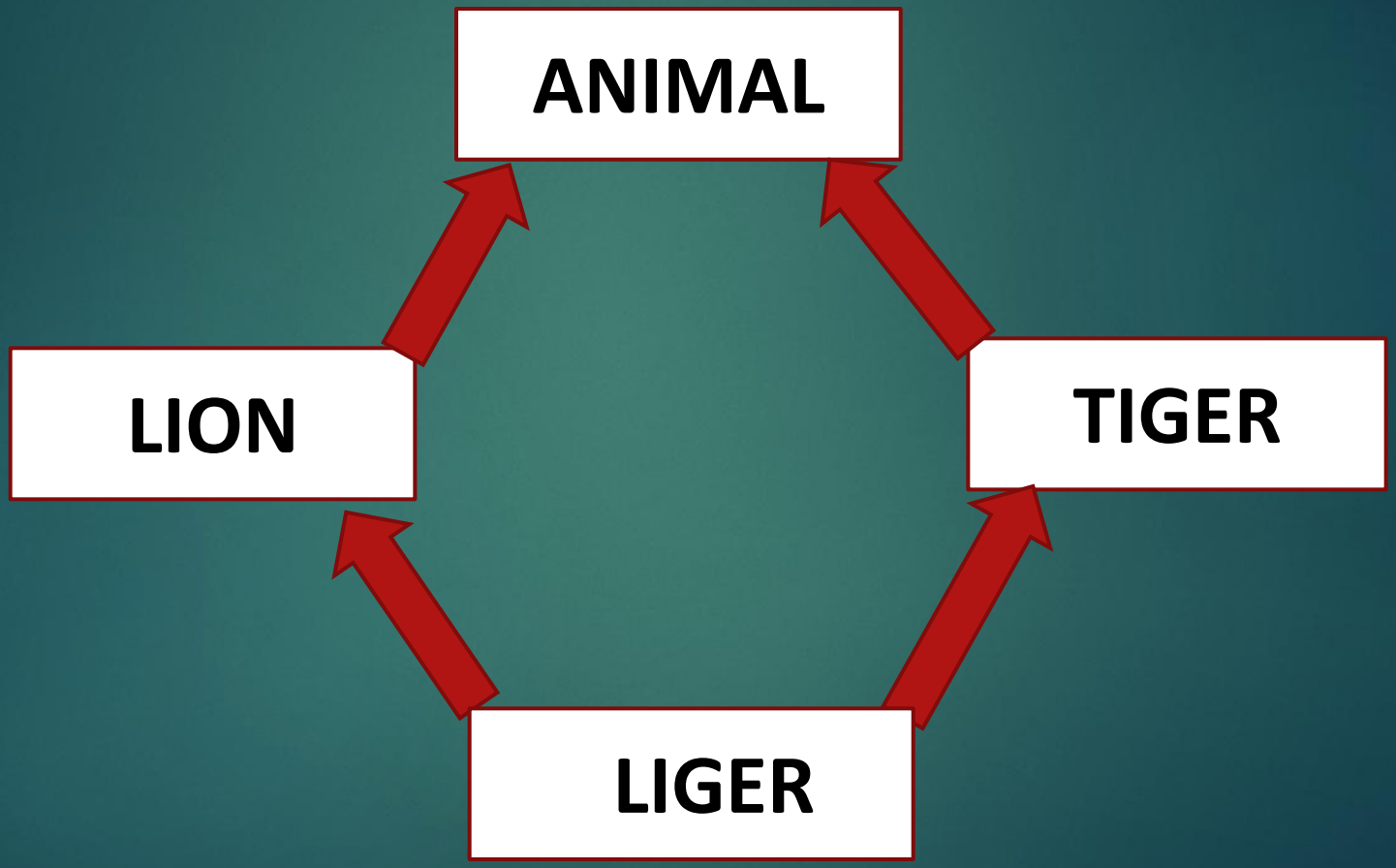
Hierarchical Inheritance



Multiple Inheritance



Hybrid Inheritance



Inheritance Accessibility Mode

- Inheritance syntax

class <Derived class Name> : <AccessibilityMode> <Base class Name>

{ // Class members

}};

- Accessibility modes can be public, protected and private

Example:

class CricketPlayer : public Player

{ // Class members

}};

Inheritance Accessibility Mode

- According to use of inheritance accessibility mode , inheritance is divided in three kinds
  1. Public Inheritance
  2. Protected Inheritance
  3. Private Inheritance

Inheritance and Accessibility of Fields

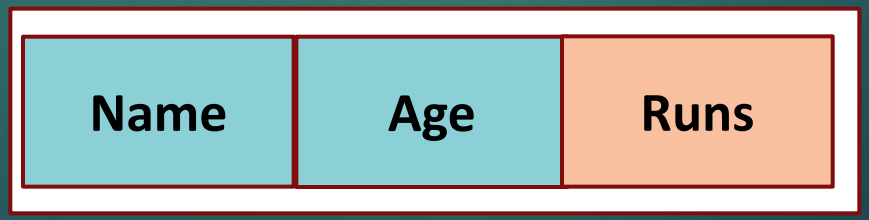
Type of Inheritance	private field	protected field	public field
private	private	private	private
protected	private	protected	protected
public	private	protected	public

Note: private members gets inherited but remains inaccessible.

Implementing Inheritance Example

- Steps to be followed

1. Create class Player with Name, Age .
2. Create class CricketPlayer and inherit Player class publically and add Runs as data members
3. Create object of Player and CricketPlayer class and call member function
4. Object of CricketPlayer can be represented as



Constructor Calls in Inheritance

- If Derived class object is created, first Base class constructor and then Derived class constructor gets called
- If Derived class object is getting destroyed, first Derived class and then Base class Destructor gets called
- If base class has 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)

Constructor Calls in Inheritance

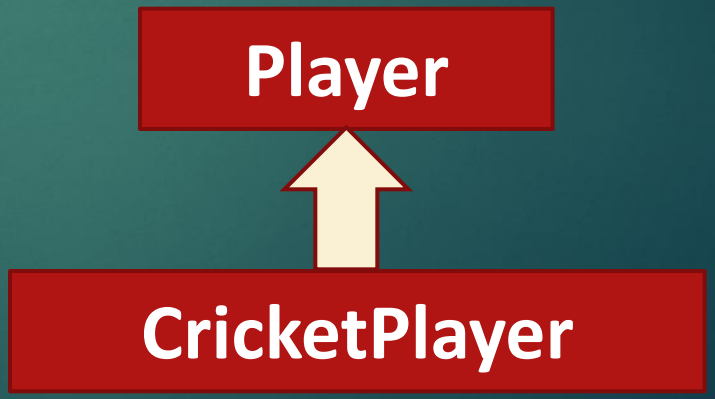
- Why Base class constructor gets called in inheritance?

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

- What does not get inherited?
  - Constructor
  - Destructor
  - Copy Constructor
  - Overloaded Assignment Operator

Up casting

- **Up casting** is using the Super class reference or pointer to refer the **sub-class** object
- Conversion of sub-class reference or pointer into its super class reference is called **Up casting.**
- Base class pointer can point to derived class object provided by Inheritance is **PUBLIC**



Polymorphism

- Polymorphism means many forms of same thing
- There are two type of Polymorphism

1. Static Polymorphism (Early or compile time binding)
   - Achieved through Function Overloading
   - Function calls are resolved at compile time
2. Runtime Polymorphism (Late or runtime binding)
   - Achieved through Function Overriding & late binding
   - Function calls are resolved at runtime

Function Overriding

- When both super class and sub-class have member function with same declaration but different implementation is called as Function Overriding
- Redefinition of super class function in sub class
- Requirement of Function Overriding
  - Inheritance should be there
  - Function that is redefined must have exactly same signature in both super and sub class
  - Same signature means same return type, parameter list and name of function

Late Binding & Virtual Functions

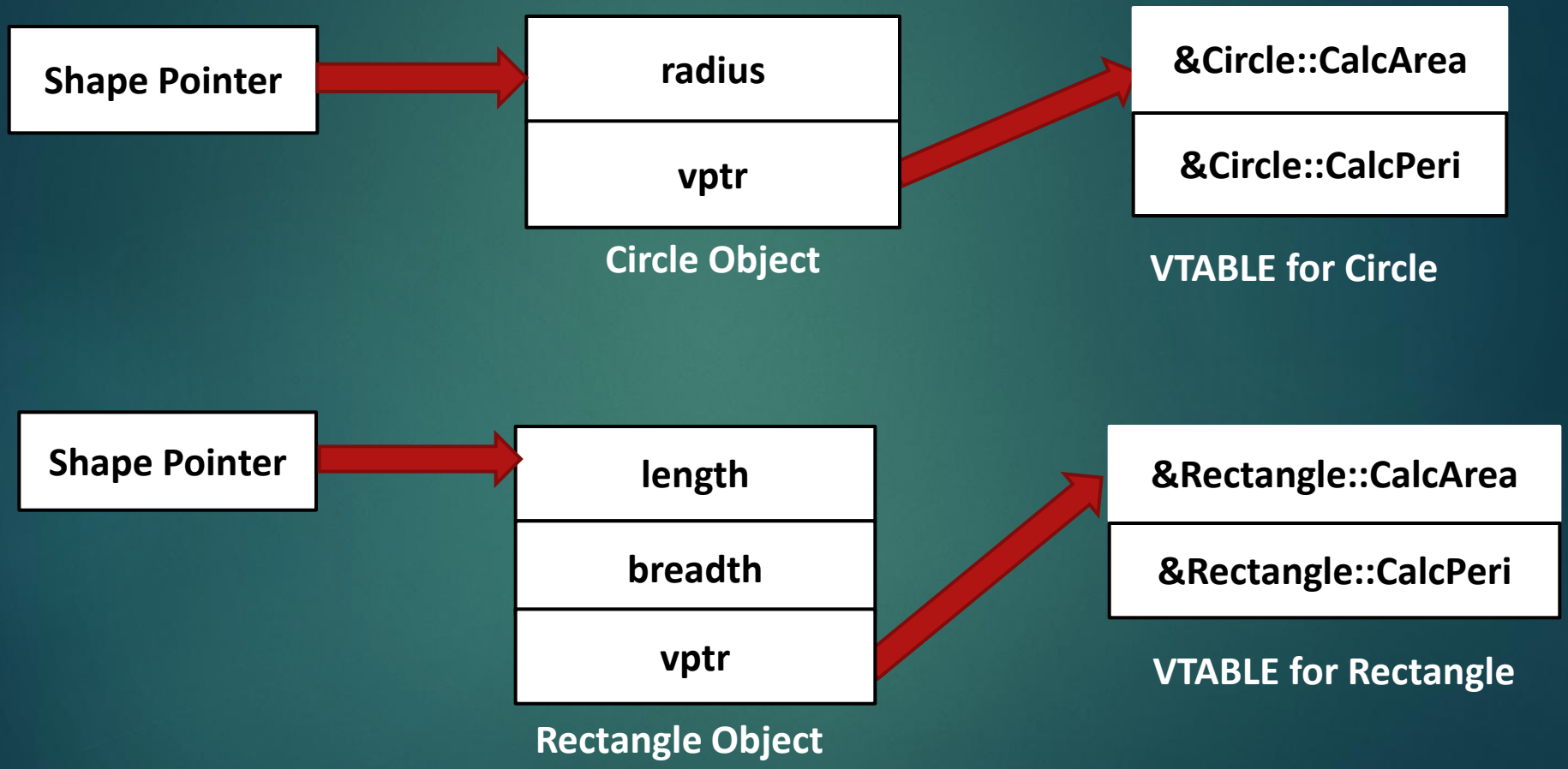
- Late binding is achieved through virtual functions
- **Virtual Functions**
  - Virtual functions are member functions marked with **virtual** keyword
  - Virtual means wait until runtime for resolving function call
  - Once we mark any function as virtual, it remains virtual in entire inheritance hierarchy
  - Virtual function can be made as pure virtual(by assigning zero)

    Ex. virtual Float CalcArea()=0;
  - Pure virtual function can not have body (definition)

Virtual Table and Virtual Pointer

- If a class has virtual function
  - VTABLE will be created for that class.
  - VPTR will be added as data member in class
  - VPTR will be available in every object
  - **VTABLE is per class and VPTR is per object**
  - VPTR will be initialized by constructor hence constructor can not be made **virtual**
  - VTABLE has entries of all **virtual functions**

Virtual Table and Virtual Pointer



Abstract class and Interface

- Abstract class is class which has at least one pure virtual function
- Abstract class is incomplete which has common functionality implemented and some functionality is unimplemented
- A class having all functions as pure virtual is called Interface/contract.
- Object of Abstract class & Interface can not be created.

Virtual Destructor

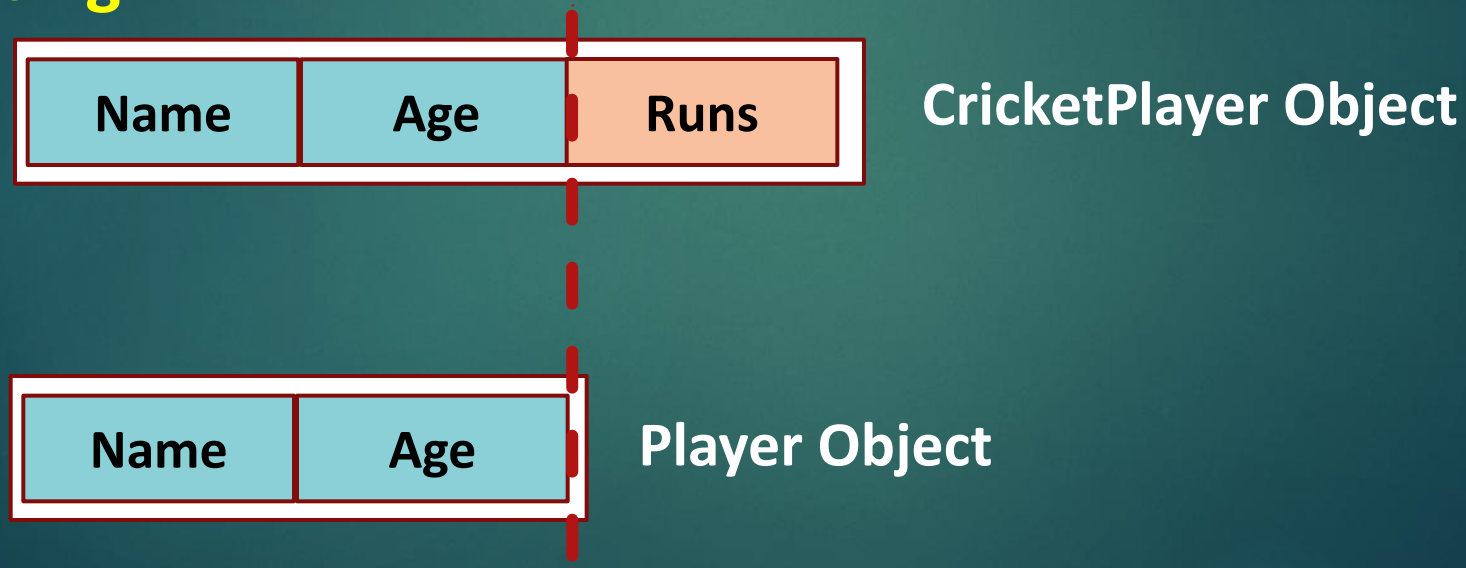
- Virtual destructor is needed in case memory is allocated for **object dynamically** within Inheritance hierarchy
- To ensure complete clean up and keeping destructor calling sequence as Derived to Base
- **Constructor can not be virtual**

Virtual Base classes & Multiple Inheritance

- In case of Multiple Inheritance from Sibling classes, properties and behaviors get inherited multiple time and there is conflict while using these duplicate properties and function. This problem is termed as **DIMOND problem.**
- To resolve diamond problem, virtual base classes can be used.

Object Slicing

- If derived class object is assigned to base class object then base class content/fields in derived class object get copied in to base class object but derived class special field will not be copied. This process is called **object Slicing.**



Thank You

Acquire Properties and Behaviours easily!!!!!

Extend and enhance only when required !!!!!

…….Inheritance