Inheritance

Inheritance

- The process by which one class acquires the properties (data members) and functionalities(methods) of another class is called inheritance.
- The class which inherits properties from another class is called subclass or child class or derived class.
- The class whose properties are inherited by subclass is called superclass or parent class or base class.
- subclass/child-class can inherit properties of its superclass/parent-class, as well as it can contain its own properties.

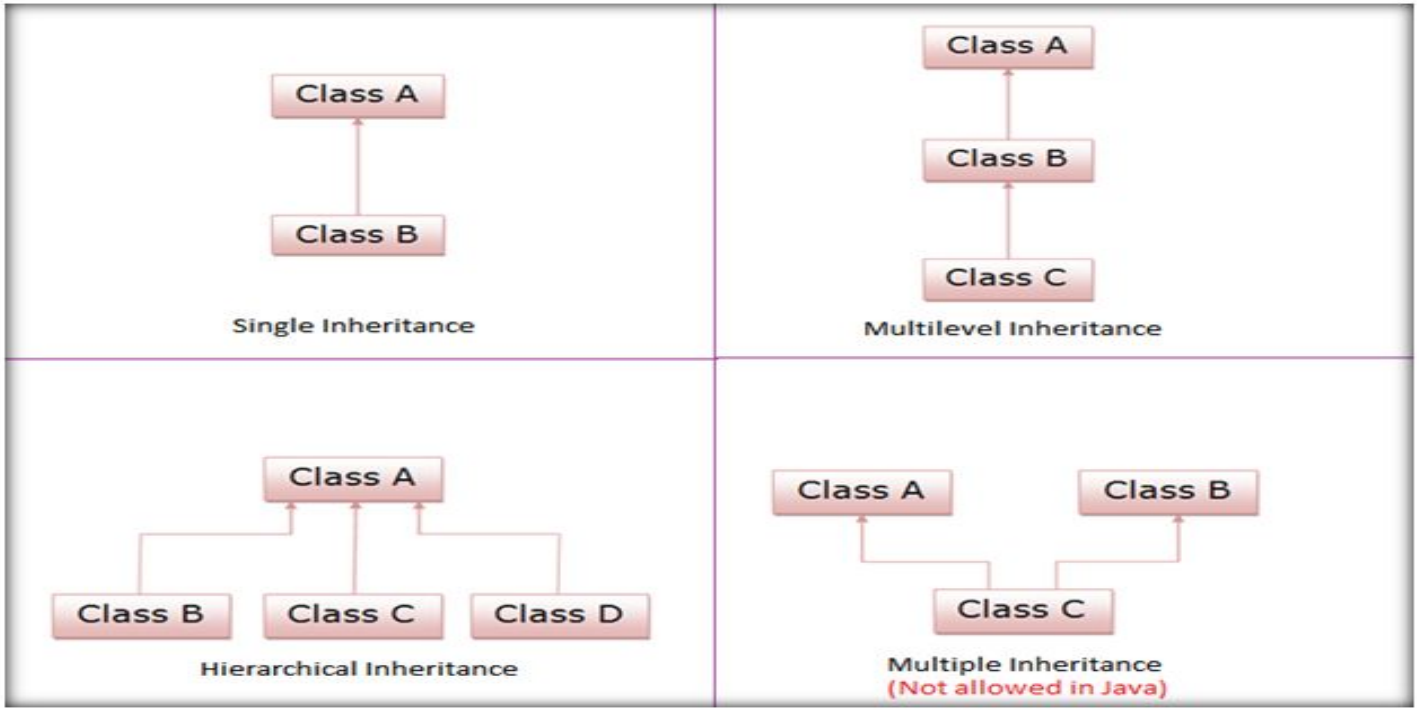
Inheritance

Advantages of Inheritance

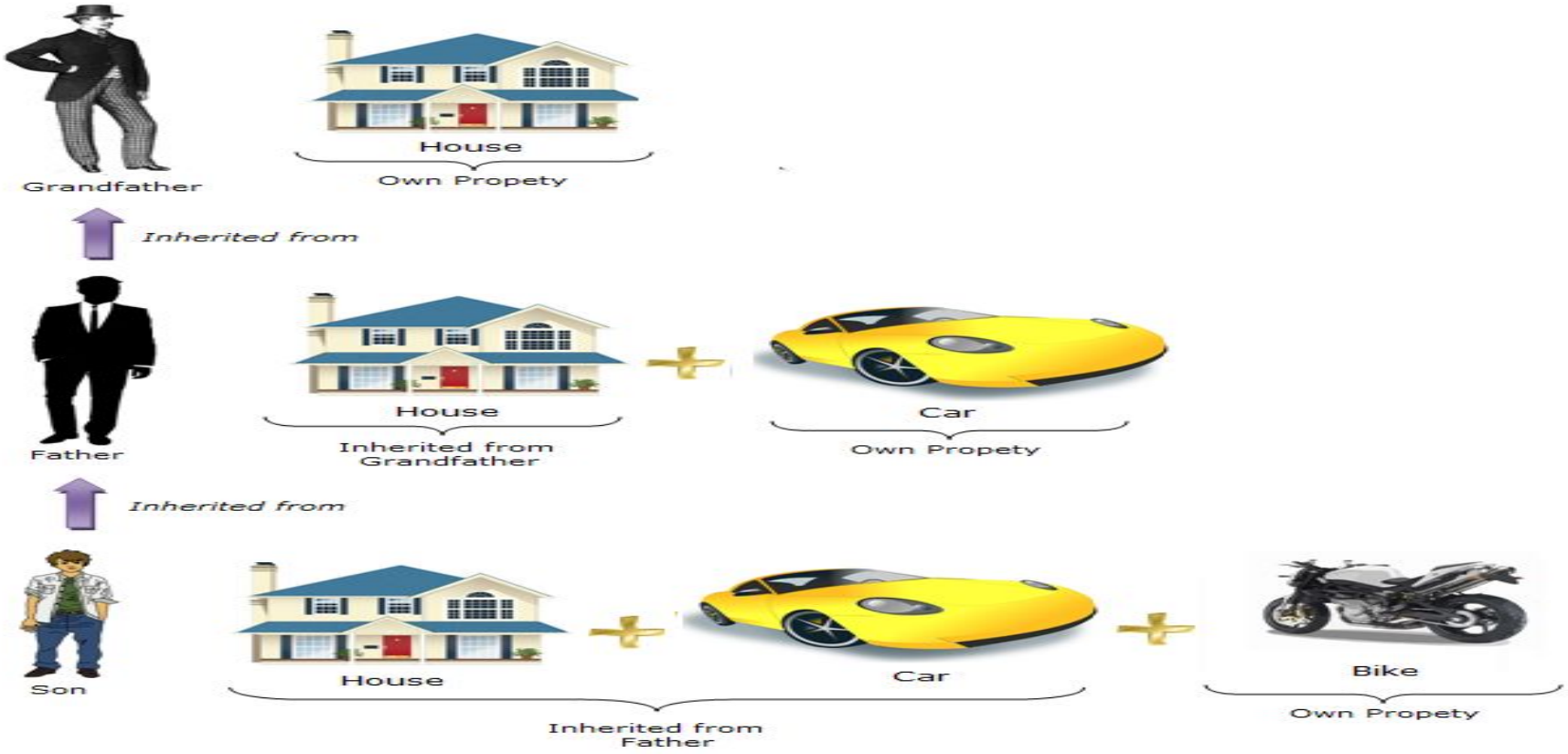
- Code reusability : Reusing the code of parent class.
- Code enhancments : Enhancments in code provided by parent class within child class as per the child class requirement.

Inheritance

Types of Inheritance



Inheritance : Multilevel Inheritance Example



Inheritance

Make a NOTE

Multiple inheritance is **not allowed** in Java to avoid the complexity in programs,

like ambiguity to call fields or methods of same names of different classes.

```
class C extends A, B          // Not allowed
{
    //Code
}
```

Inheritance

Use of super

- For calling Super or Parent class constructor explicitly. super class constructor can be called using super key from derived class constructor only and in that case it should be the first statement within the child class constructor.
- Using super, we can call any method of super class. Ex. super.myfun( );
- In case of method overriding, if we want to call superclass' overridden method.

Inheritance

Order of constructor calling

- Whenever any object of derived class is created, constructors are called from parent to child in the inheritance hierarchy.
- Class A { }

  Class B extends A { }

  Class C extends B { }

  So when we create object of C, first A's constructor will be called, then B's and then C's constructor will be called.

Inheritance

Abstract methods

- Abstract method is a method which is declared without any implementation. Abstract method is preceded by the keyword ***abstract***. Ex.

```
abstract class A

{

// Method without implementation (body), only declaration

  abstract void myfun( ) ;

}
```

Inheritance

Abstract class

- Abstract class is a class which has been declared abstract by using the abstract keyword.
- Ex.

  abstract class Test {

  ```
      //Code
  ```

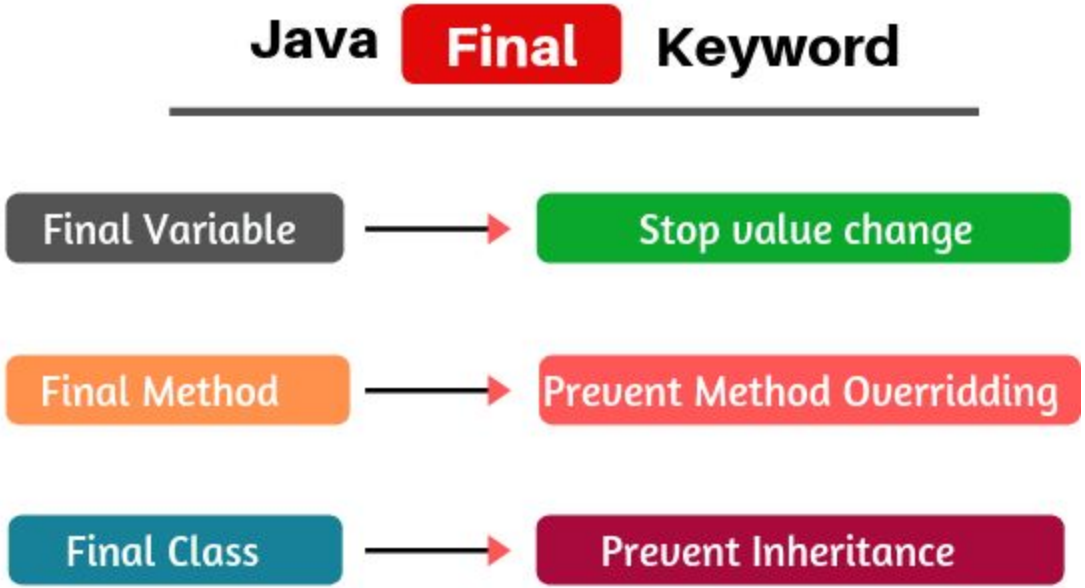
  }
- An abstract class may or may not have any abstract method.
- But if any class has any abstract method then class must be declared as abstract.
- We **can not create object** of an abstract class.

Inheritance

Use of final keyword

- Can be used with variable, method and class
- If variable is declared final, once its' value is initialized, it can not be changed.
- If method is declared final, that method can not be overridden in the subclass.
- If a class is declared final, that class can not be inherited.
- Keyword abstract and final can not be applied simultaneously. Meaning it is not possible to declare a class or method abstract as well as final because these two contradict each other. As abstract says "You must need to override /implement" but final says "This is final, no need to override/implement".

Inheritance



Inheritance

```
void myFun( )
    {
        final int a ;
        a = 5;
       // a = 10; // error: variable a might already have been assigned
       System.out.println(a)  // Print 5.
    }
```

Inheritance

Method overriding in subclass (Make A NOTE)

- Subclass defines a superclass method with same signature (means with same name and parameters)

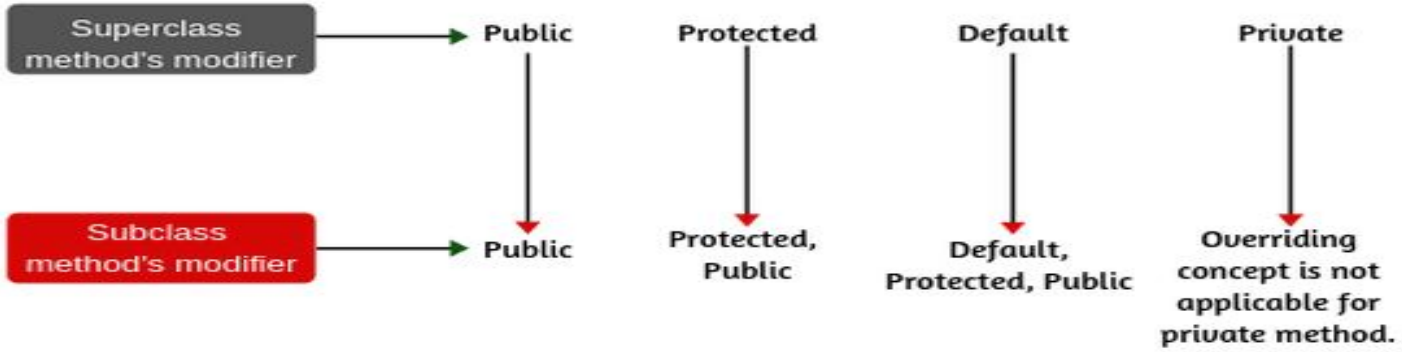
- Return type of subclass' method should be same as parent type but if we want to change the return type then that return type should be the covariant type ie. any return type in child should be a subclass of return type in the parent.

- Overriding concept is not applicable for private, final, static, and main method in Java
- Only the instance method can be overridden in Java.

Inheritance

Method overriding in subclass (Make A NOTE)

- Subclass method's access modifier must be same or higher than superclass method access modifier.



The access modifier of the overriding method cannot be more restrictive than overridden method of superclass.

Private > Default > Protected > Public

More restrictive → Less restrictive

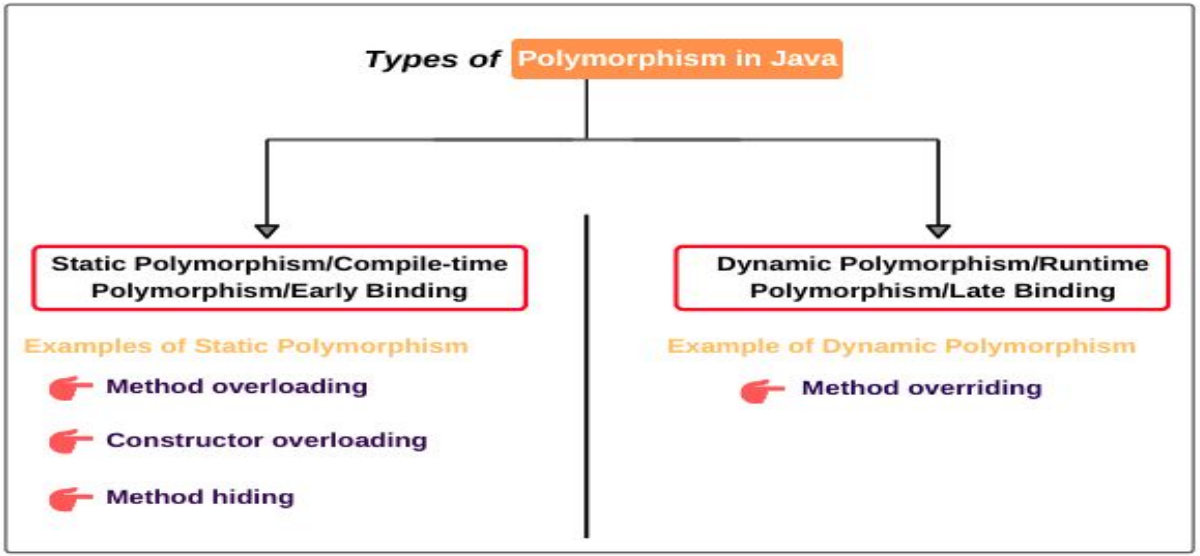
Fig: Applicable access modifiers to the overriding method

Polymorphism

- The ability to exist in different form is called polymorphism. Ex. a+b
- Polymorphism means “one thing different form” or “one interface multiple methods”
- Polymorphism in OOP is the ability of an entity to take several forms. In other words, it refers to the ability of an object (or a reference to an object) to take different forms of objects.

Polymorphism

- Types of Polymorphism : Static Polymorphism & Dynamic Polymorphism



- Binding refers to the association(linking) of method definition with the method call.

Polymorphism

- The binding of static, private and final methods is compile-time. The reason is that the these method cannot be overridden and the type of the class is determined at the compile time.
- When the compiler is able to figure out the correct version of something (data member and member functions) during compilation, that's known as static binding.
- In java member variables have static binding.

Polymorphism

Static Polymorphism

- A polymorphism that exhibited during compilation is called static polymorphism.
- In the static polymorphism, the behavior of a method is decided at compile-time. Hence, Java compiler binds method calls with method definition/body during compilation.

Dynamic Polymorphism

- A polymorphism that is exhibited at runtime is called dynamic polymorphism.
- In dynamic polymorphism, the behavior of a method is decided at runtime, therefore, the JVM (Java Virtual Machine) binds the method call with method definition/body at runtime and invokes the relevant method during runtime when the method is called.

Polymorphism

Interface in Java

- Creating an interface means defining a contract for *what a class can do,* without saying anything about *how the class will do it.*
- There can be only abstract methods in an interface.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.

Syntax to create an interface

```
interface interface-name{

    //abstract methods

}
```

Polymorphism

Properties of Interface in Java

- An interface can contain only abstract methods.
- All the member functions in interface are implicitly public and abstract, it does not matter whether we include these modifiers with the function declaration or not.
- All the data members in interface are implicitly public, static and final, it does not matter whether we include these modifiers with the data member declaration or not.
- An interface can extend (inherit) one or more than one interfaces. It means, interface exhibits the properties of multiple inheritance.
- A class can implement one or more than one interfaces.

Polymorphism

Properties of Interface in Java

- We can not instantiate an interface. It means we can not create any object of interface.
- Interface does not contain any constructor.
- If a class implements an interface, all the methods from the interface within the class must be public.

AccessModifier

Visibility	Public	Protected	*Default*	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, *through inheritance*	No	No
From any non-subclass class outside the package	Yes	No	No	No