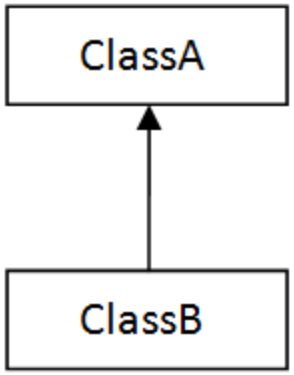
Inheritance

Lecture 14

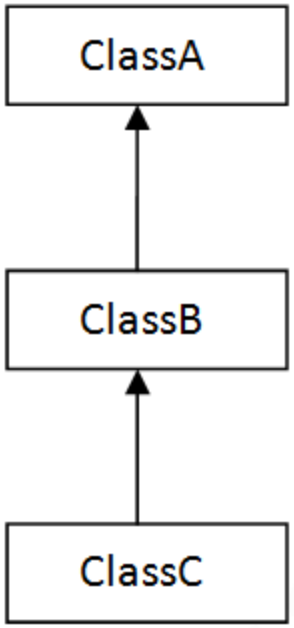
Prof. Anita Agrawal,
BITS –Pilani, K.K.Birla Goa campus

Types of inheritances in Java

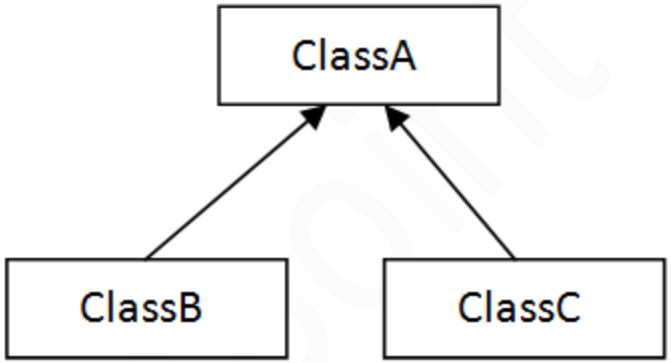
- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Multiple Inheritance (Through Interfaces)



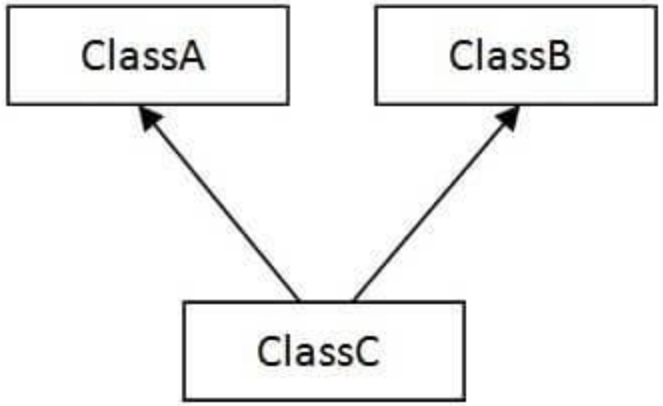
1) Single



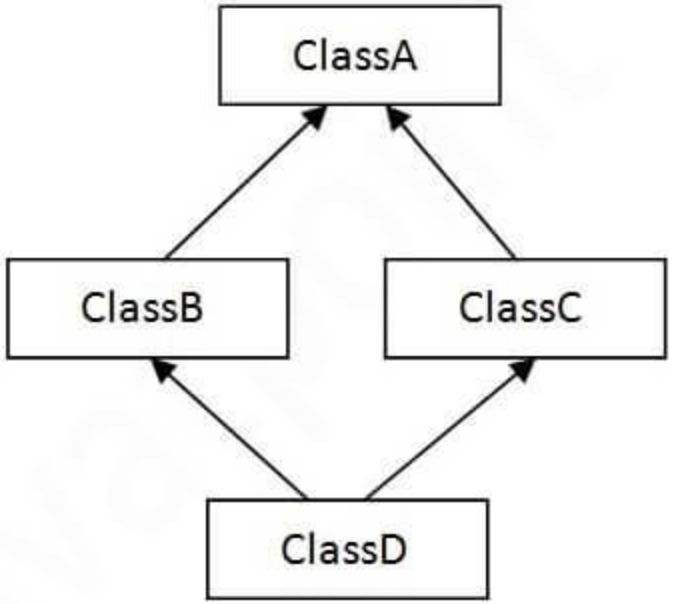
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

- **Single Inheritance**: One class (sub class) extends another class (superclass)

  (one class only)

- **Multilevel Inheritance**: In other words, a derived class will be inheriting a base class and the derived class will act as the base class to other class.

- **Multiple Inheritance**: One class extends more than one class. Java does not support multiple inheritance through classes. It can support this only through interfaces

- **Hierarchical inheritance**: One class is extended by many subclasses.
- **Hybrid Inheritance**: It is a combination of single and multiple inheritance.

Summary:

- In Java, when an "Is-A" relationship exists between two classes, we use Inheritance.
- The keyword '**extend**' is used by the subclass to inherit the features of the superclass.
- Inheritance is important since it leads to **reusability** of the code.

Method overriding

- Overriding occurs *only* when the names and the type signatures of the two methods in the subclass and the superclass are identical.

Dynamic method dispatch

- A mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
  - **Is one of the ways in which java supports run time polymorphism**
- A superclass reference variable can refer to a subclass object. -**upcasting**.
- Java uses this fact to resolve calls to overridden methods at run time.

- If a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

- In Java, we can override methods only, not the variables(data members), so **runtime polymorphism cannot be achieved by data members.**

Final revisited ….

Final classes:

- When a class is declared with ***final*** keyword, it is called a **final** class.
- A final class **cannot be extended(inherited)**.
- Uses of a final class :
  - To prevent inheritance
    - Example: All Wrapper classes like Integer, Float etc. are final classes.
  - To create an immutable class
    - Example: The predefined **String** class.

    You can not make a class immutable without making it final.

Finaldemo.java

Overloading vs. Overriding:

Overriding occurs when both, the **name** and the **type signature** of the two methods are **identical.** Otherwise, the two methods are simply overloaded

Final method

- Declared with a final keyword.
- A final method cannot be overridden by subclasses.
  - Example: the Object class.
- Declare methods with final keyword to follow the same implementation throughout all the derived classes.

Finalmethoddemo.java

- During inheritance, declare methods with final keyword for which we require to follow the same implementation throughout all the derived classes.
- It is not necessary to declare final methods in the initial stage of inheritance(base class always).
- final method can be declared in any subclass for which we want any other class extending this subclass, must follow same implementation of the method as in the that subclass.

- Since private methods are inaccessible, they are implicitly final in Java. So adding *final* specifier to a private method doesn't add any value. It may in-fact cause unnecessary confusion.

What happens if a class is declared final?

- Declaring a class as final implicitly declares all of its methods as final, too.