



Inheritance

Presented by,
Prof. Rupali Shende

00P Concept

- Another fundamental object-oriented technique is inheritance, used to organize and create reusable classes
- focuses on:
 - deriving new classes from existing classes
 - creating class hierarchies
 - the protected modifier
 - polymorphism via inheritance
 - inheritance hierarchies for interfaces
 - inheritance used in graphical user interfaces




Introduction

- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- When you inherit from an existing class, you can reuse methods and fields of the parent class.
- Moreover, you can add new methods and fields in your current class also.



Why use inheritance

- For Method Overriding (so runtime polymorphism can be achieved).
 - For Code Reusability.
- 

Inheritance

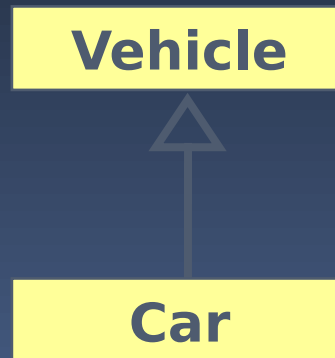
- *Inheritance* allows a software developer to derive a new class from an existing one
- The existing class is called the *parent class*, or *superclass*, or *base class*
- The derived class is called the *child class* or *subclass*.
- As the name implies, the child inherits characteristics of the parent
- That is, the child class inherits the methods and data defined for the parent class

Inheritance

- To use a derived class, the programmer can add new variables or methods, or can modify the inherited ones
- *Software reuse* is at the heart of inheritance
- By using existing software components to create new ones, we capitalize on all the effort that went into the design, implementation, and testing of the existing software

Inheritance

- Inheritance relationships often are shown graphically in a UML class diagram, with an arrow with an open arrowhead pointing to the parent class



Inheritance should create an *is-a* relationship, meaning the child *is a* more specific version of the parent

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Deriving Subclasses

- In Java, we use the reserved word `extends` to establish an inheritance relationship
- The syntax of Java Inheritance
- **class** Subclass-name **extends** Superclass-name
- {
- //methods and fields
- }
- The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

```
class Car extends Vehicle
{
    // class contents
}
```

ACCESS CONTROL AND INHERITENCE contd...

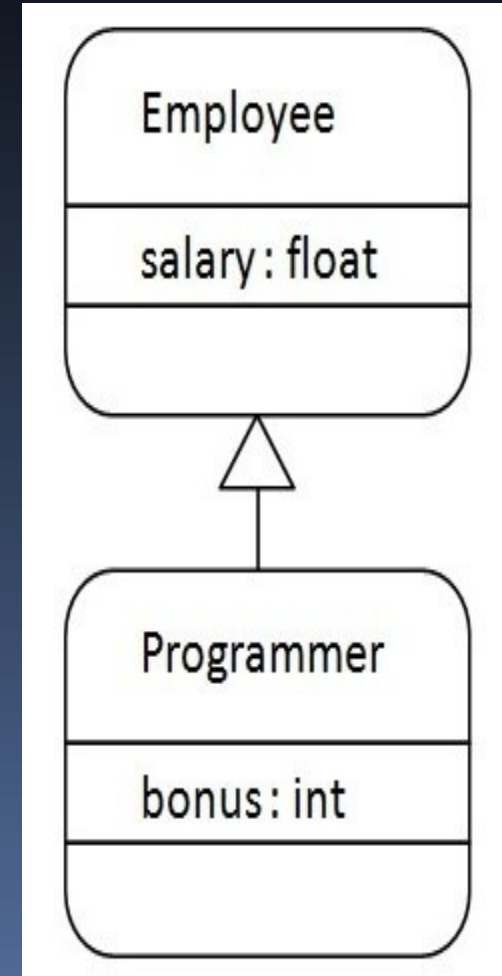
We can summarize the different access types according to who can access them in the following way:

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

NOTE: Constructors and destructors of the base class are never inherited.

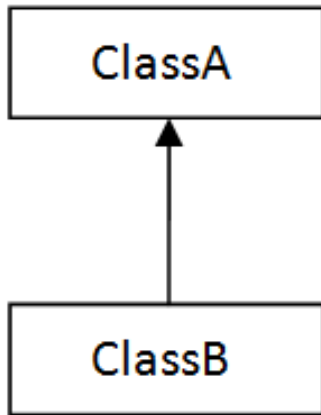
Inheritance Example

```
class Employee{
    float salary=40000;
}
class Programmer extends Employee{
    int bonus=10000;
    public static void main(String args[]){
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
        System.out.println("Bonus of Programmer is:"+p.bonus);
    }
}
```

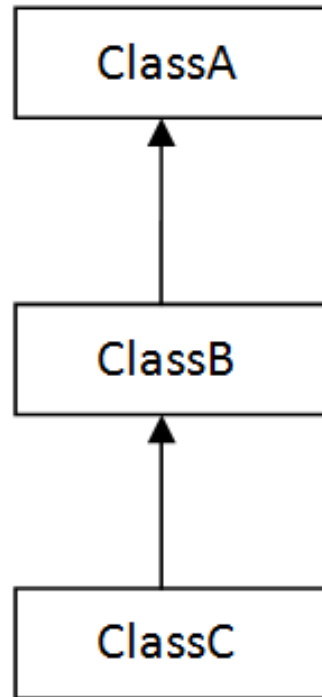


- Programmer salary is:40000.0
- Bonus of programmer is:10000

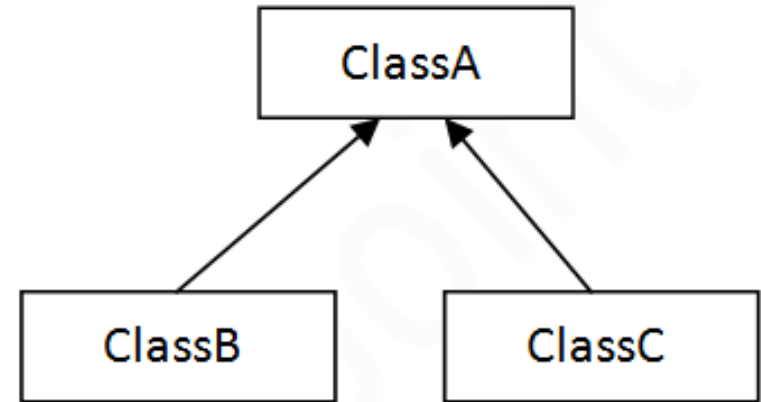
Types of inheritance in java



1) Single

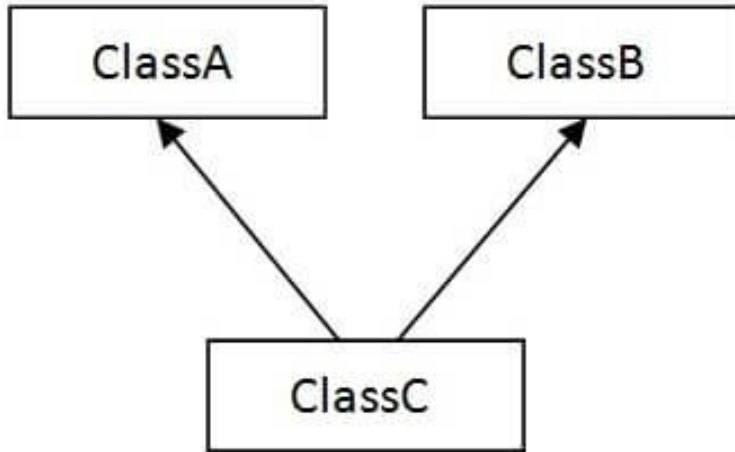


2) Multilevel

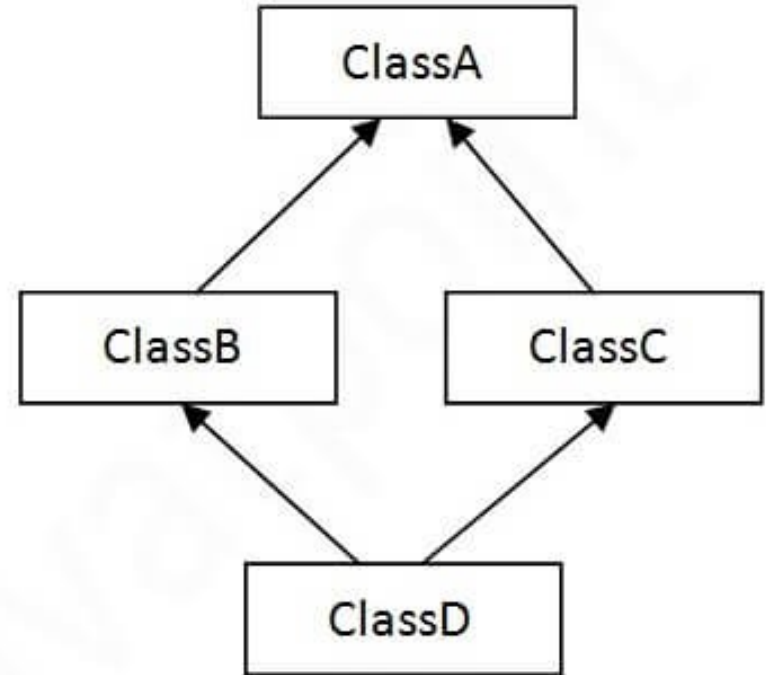


3) Hierarchical

Types



4) Multiple



5) Hybrid

Single Inheritance

- When a class inherits another class, it is known as a *single inheritance*.

Output This is a Vehicle

```
#include<iostream>
using namespace std;

// base class
class Vehicle {
    public:
        Vehicle()
        {
            cout << "This is a Vehicle\n";
        }
};

// sub class derived from a single base classes
class Car : public Vehicle {

};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0;
}
```

Multilevel Inheritance

- When there is a chain of inheritance, it is known as *multilevel inheritance*.

OutputThis is a Vehicle Objects with 4 wheels are vehicles Car has 4 Wheels

```
#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// first sub_class derived from class vehicle
class fourWheeler : public Vehicle {
public:
    fourWheeler()
    {
        cout << "Objects with 4 wheels are vehicles\n";
    }
};

// sub class derived from the derived base class fourWheeler
class Car : public fourWheeler {
public:
    Car() { cout << "Car has 4 Wheels\n"; }
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```

Multiple Inheritance

- In Multiple inheritances, one class can have more than one super-class and inherit features from all parent classes.

Output

```
This is a Vehicle
This is a 4
wheeler Vehicle
```

```
#include <iostream>
using namespace std;

// first base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// second base class
class FourWheeler {
public:
    FourWheeler()
    {
        cout << "This is a 4 wheeler Vehicle\n";
    }
};

// sub class derived from two base classes
class Car : public Vehicle, public FourWheeler {
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```


Hierarchical Inheritance

- When two or more classes inherit a single class, it is known as *hierarchical inheritance*.

```
// C++ program to implement
// Hierarchical Inheritance
#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// first sub class
class Car : public Vehicle {
};

// second sub class
class Bus : public Vehicle {
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base class.
    Car obj1;
    Bus obj2;
    return 0;
}
```

The protected Modifier

- Visibility modifiers determine which class members are inherited and which are not
- Variables and methods declared with `public` visibility are inherited; those with `private` visibility are not
- There is a third visibility modifier that helps in inheritance situations: `protected`

The protected Modifier

- The protected modifier allows a member of a base class to be inherited into a child
- Protected visibility provides more encapsulation than public visibility does
- However, protected visibility is not as tightly encapsulated as private visibility

Hybrid Inheritance

- It is a mix of two or more of the above types of inheritance.

```
#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// base class
class Fare {
public:
    Fare() { cout << "Fare of Vehicle\n"; }
};

// first sub class
class Car : public Vehicle {
};

// second sub class
class Bus : public Vehicle, public Fare {
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base class.
    Bus obj2;
    return 0;
}
```

Important facts about inheritance

- **Default superclass:** Except Object class, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of the Object class.
- **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only **one** superclass. This is because Java does not support multiple inheritances with classes. Although with interfaces, multiple inheritances are supported by Java.

Important facts..

- **Inheriting Constructors:** A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
- **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods (like getters and setters) for accessing its private fields, these can also be used by the subclass.

Benefits of Inheritance

- Inheritance helps in code reuse. The child class may use the code defined in the parent class without re-writing it.
- Inheritance can save time and effort as the main code need not be written again.
- Inheritance provides a clear model structure which is easy to understand.
- An inheritance leads to less development and maintenance costs.
- With inheritance, we will be able to override the methods of the base class so that the meaningful implementation of the base class method can be designed in the derived class. An inheritance leads to less development and maintenance costs.
- In inheritance base class can decide to keep some data private so that it cannot be altered by the derived class.



Costs of Inheritance

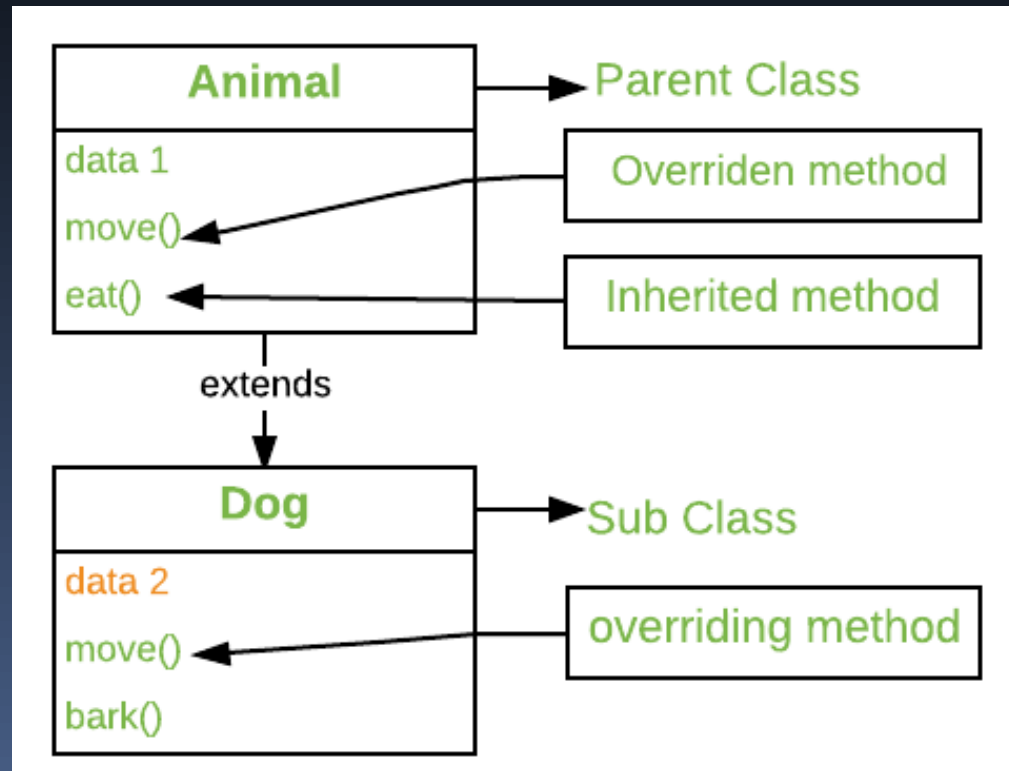
- Inheritance decreases the execution speed due to the increased time and effort it takes, the program to jump through all the levels of overloaded classes.
- Inheritance makes the two classes (base and inherited class) get tightly coupled. This means one cannot be used independently of each other.
- The changes made in the parent class will affect the behavior of child class too.
- The overuse of inheritance makes the program more complex.

Method Overriding

- In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.
- When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to *override* the method in the super-class.

Method Overriding

- If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. In other words, *it is the type of the object being referred to* (not the type of the reference variable) that determines which version of an overridden method will be executed.



Example

- The purpose of Method Overriding is clear here. Child class wants to give its own implementation

```
class BaseClass
{
public:
    virtual void Display()
    {
        cout << "\nThis is Display() method"
              << " of BaseClass";
    }
    void Show()
    {
        cout << "\nThis is Show() method "
              << "of BaseClass";
    }
};

class DerivedClass : public BaseClass
{
public:
    // Overriding method - new working of
    // base class's display method
    void Display()
    {
        cout << "\nThis is Display() method"
              << " of DerivedClass";
    }
};

// Driver code
int main()
{
    DerivedClass dr;
    BaseClass &bs = dr;
    bs.Display();
    dr.Show();
}
```

Advantage of method overriding

- The main advantage of method overriding is that the class can give its own specific implementation to a inherited method **without even modifying the parent class code.**
- This is helpful when a class has several child classes, so if a child class needs to use the parent class method, it can use it and the other classes that want to have different implementation can use overriding feature to make changes without touching the parent class code.

Rules of method overriding

- Argument list: The argument list of overriding method (method of child class) must match the Overridden method(the method of parent class). The data types of the arguments and their sequence should exactly match.
- **Access Modifier** of the overriding method (method of subclass) cannot be more restrictive than the overridden method of parent class. For e.g. if the Access Modifier of parent class method is public then the overriding method (child class method) cannot have private, protected and default Access modifier,because all of these three access modifiers are more restrictive than public.

Rules

- private, static and final methods cannot be overridden as they are local to the class. However static methods can be re-declared in the sub class, in this case the sub-class method would act differently and will have nothing to do with the same static method of parent class.



Thank You!