
Inheritance

- Inheritance is the mechanism that allows us to describe this *specialization* relationship between classes.

```
class B { ... }  
class A extends B { ... }
```

- *A* is a *subclass* of *B*, or equivalently, *A* is *derived from B*, *A* is a *child of B*, or *B* is a *superclass of A*, or *B* is a *parent of A*.
- Means that the set of *A* objects is a subset of the set of *B* objects.

```
class Labrador extends Dog { ... }
```

Inheritance

- Inheritance represents specialization

```
class Engine {
    // ...
}
class Car {
    Engine e;
    Car() { e = new Engine(); }
    // ...
}
class RacingCar extends Car {
    Aerofoil a;
    TurboCharger t;
}

// In some client
RacingCar r = new RacingCar();
Engine e1 = r.e; // e is inherited from Car
TurboCharger t1 = r.t;
Car c = new Car();
Engine e2 = c.e;
TurboCharger t2 = c.t; // Error
```

Inheritance

- Inheritance serves as a tool for reusability:
- We can write

```
class RacingCar extends Car {  
    Aerofoil a;  
    TurboCharger t;  
}
```

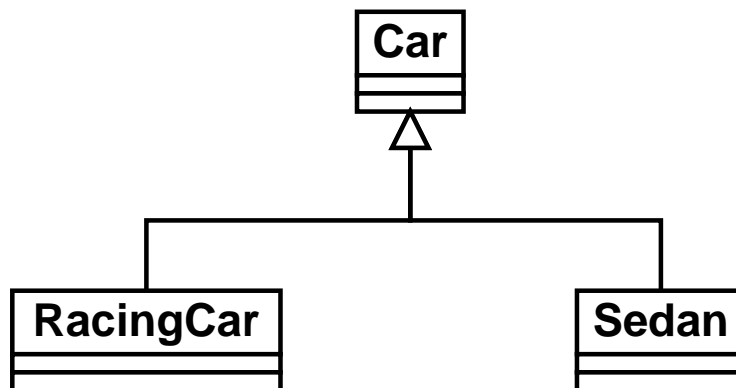
instead of

```
class RacingCar {  
    Engine e;  
    Aerofoil a;  
    TurboCharger t;  
}
```

Inheritance

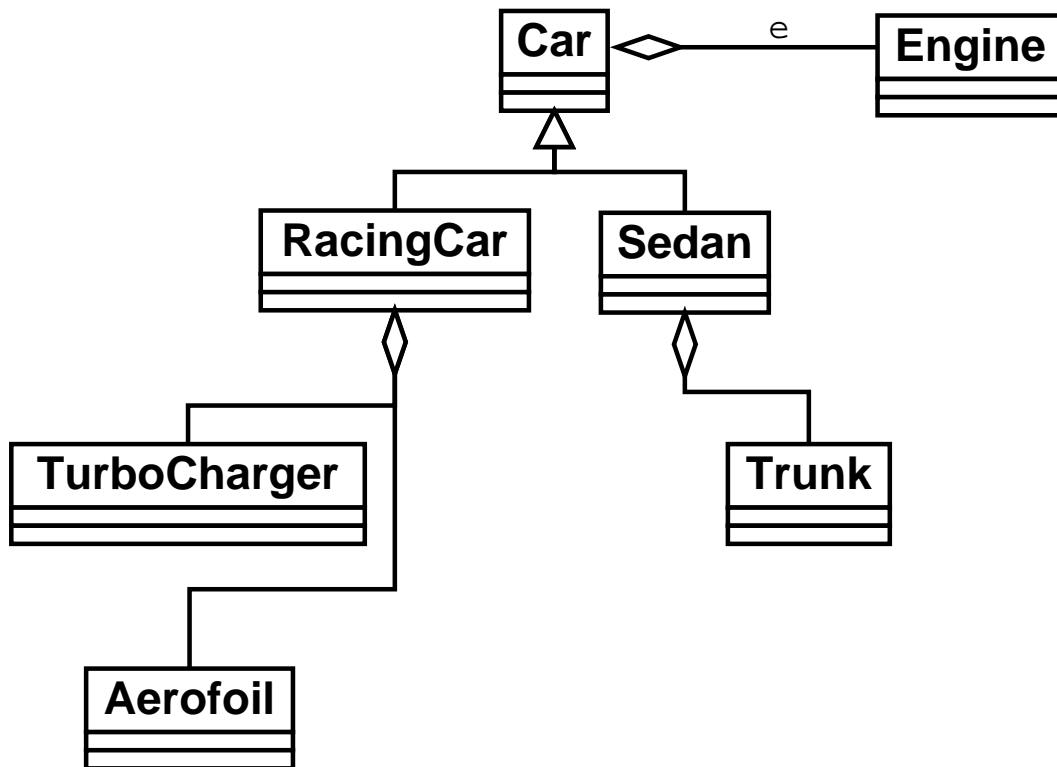
- Classes can have many subclasses

```
class Sedan extends Car {  
    Trunk t;  
    PassengerSeats[] ps;  
}  
  
// In some client  
Sedan s = new Sedan();  
s.turn_on();
```



Inheritance

- Attributes in a class are shared between its subclasses (but not the values of those attributes!)



Inheritance

- Inheritance is a transitive relation: if every A is a B and every B is a C, then every A is a C

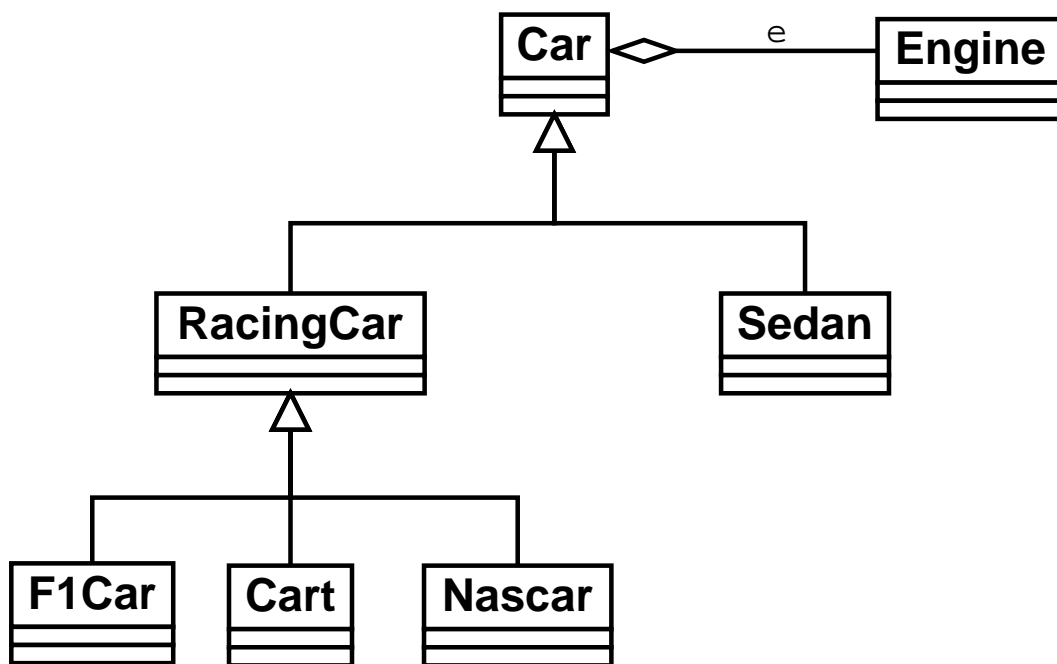
```
class F1Car extends RacingCar {  
    SpeedControlSystem scs;  
}
```

- instead of

```
class F1Car {  
    Engine e;  
    Aerofoil a;  
    TurboCharger t;  
    SpeedControlSystem scs;  
}
```

Inheritance

- Class hierarchy:



Inheritance

- A closer look at inheritance as specialization

```
class Animal {
    boolean tired, hungry;
    void eat()
    {
        get_food();
        hungry = false;
    }
    void get_food() { ... }
    void sleep()
    {
        System.out.println("zzz...");
        tired = false;
    }
}
```

Inheritance

```
class Dog extends Animal {
    Legs[] l;
    Tail t;
    void run()
    {
        tired = true; // From class Animal
        hungry = true;
    }
    void bark()
    {
        System.out.println("Woof, Woof!");
    }
}
class Labrador extends Dog {
    void say_hello()
    {
        t.wiggle(); // t from class Dog
    }
}
```

Inheritance

```
public class ZooTest {
    public static void main(String[] args)
    {
        Labrador l = new Labrador();
        l.say_hello(); // Will call l.t.wiggle();
        l.run();
        if (l.hungry)
            l.eat(); // from class Animal
        if (l.tired)
            l.sleep();
    }
}
```

Inheritance

- Inheritance represents also a spectrum of possibilities or alternatives, given by the subclasses of a class
- If every B is an A and every C is an A, and nothing else is an A, then an A is either a B or a C
 - (e.g. if every racing car is a car, and every sedan is a car, and nothing else is a car, then a car is either a racing car or a sedan.)

```
class Animal { ... }  
class Dog extends Animal { ... }  
class Cat extends Animal { ... }  
class Bird extends Animal { ... }
```

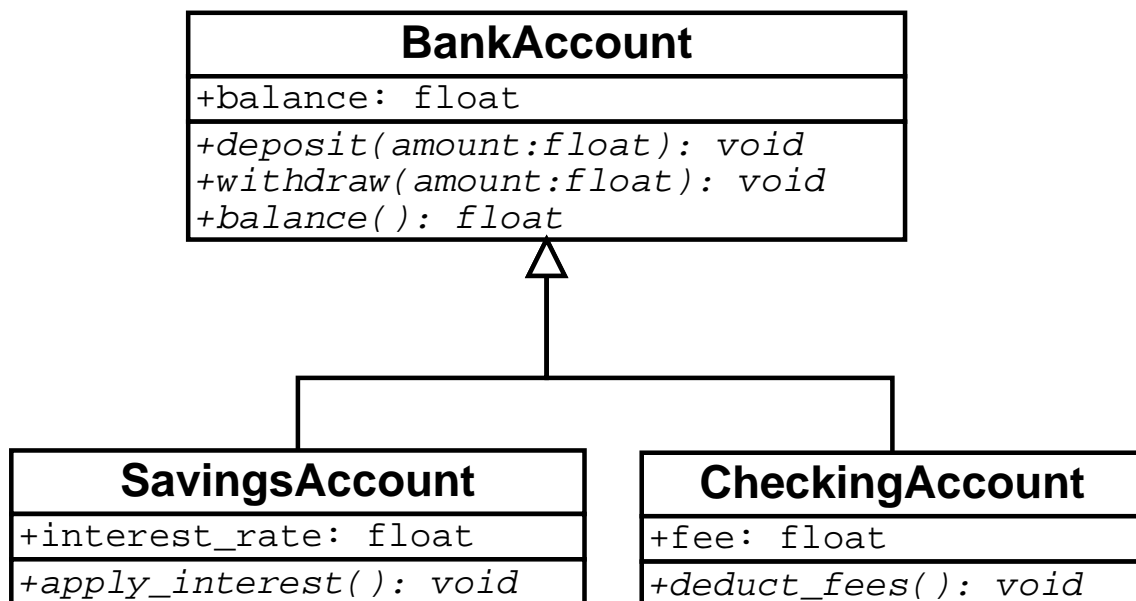
```
// In some client  
Animal a1 = new Dog();  
Animal a2 = new Cat();  
Animal a3 = new Bird();  
Dog d = new Animal(); // Wrong!
```

Inheritance

- Classes as sets of objects:
 - “is-a” between an object and a class is the same as \in
 - “is-a” between two classes is the same as \subseteq
- Let A, B, C be sets
 - If $A \subseteq B$ and $x \in A$ then $x \in B$
 - If $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$
 - If $B \subseteq A$ and $C \subseteq A$, and there is no other set D such that $D \subseteq A$ then $A = B \cup C$

Inheritance

- A bank account is either a savings account or a checking account, then a savings account is a kind of bank account, and a checking account is a kind of bank account.



Inheritance

```
class BankAccount {
    private float balance;
    public BankAccount(float initial_balance)
    {
        balance = initial_balance;
    }
    public void deposit(float amount)
    {
        balance = balance + amount;
    }
    public void withdraw(float amount)
    {
        balance = balance - amount;
    }
    public float balance() { return balance; }
}
```

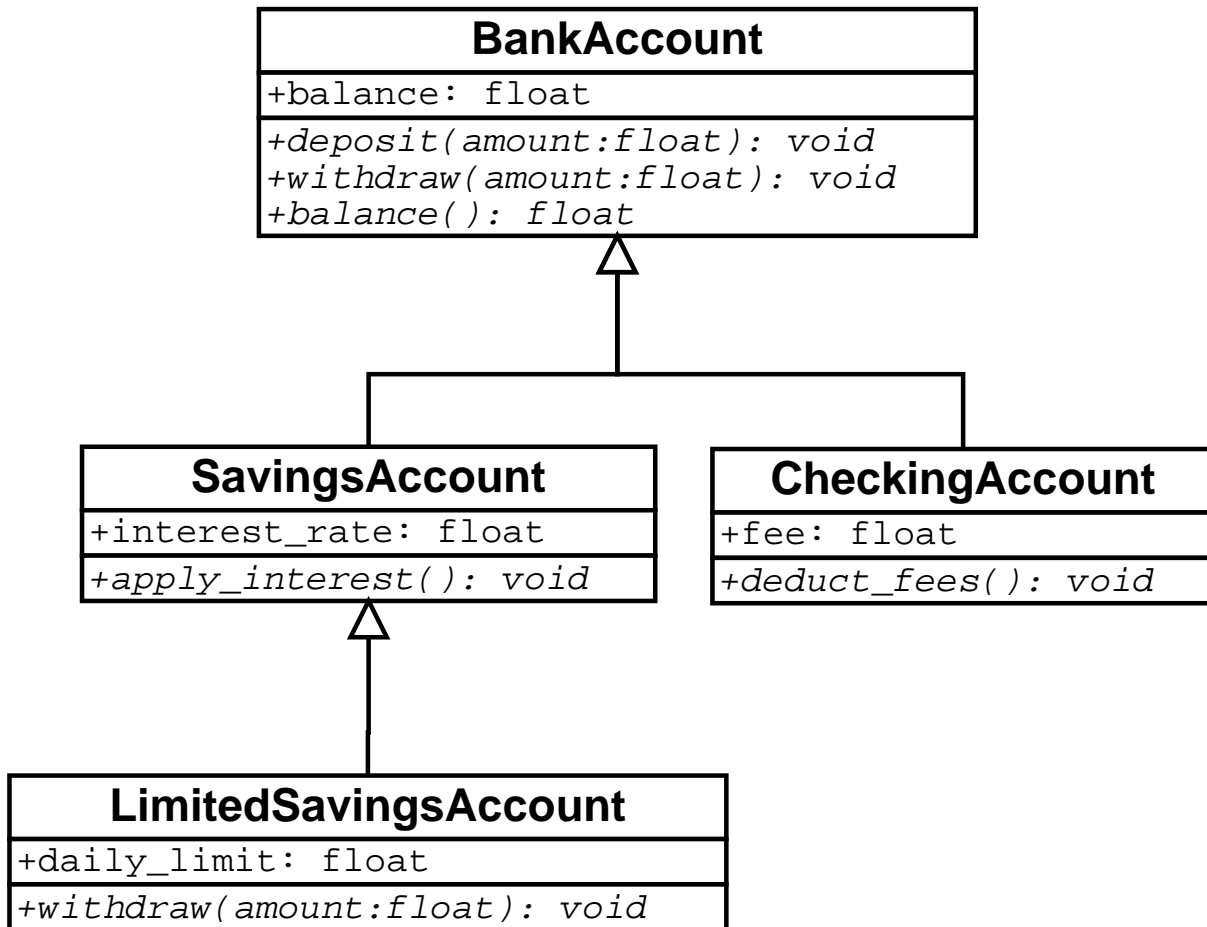
Inheritance

```
class SavingsAccount extends BankAccount {
    private float interest_rate;
    public SavingsAccount(float initial_balance,
                          float rate)
    {
        super(initial_balance); // Calls superclass
                                // constructor
        interest_rate = rate;
    }
    public void apply_interest()
    {
        balance = balance
                + balance * interest_rate/100.0;
    }
}
```

Inheritance

```
class CheckingAccount extends BankAccount {
    private float fee;
    public SavingsAccount(float initial_balance,
                          float fee)
    {
        super(initial_balance);
        this.fee = fee;
    }
    public void deduct_fee()
    {
        balance = balance - fee;
    }
}
```

Overriding methods



Overriding methods

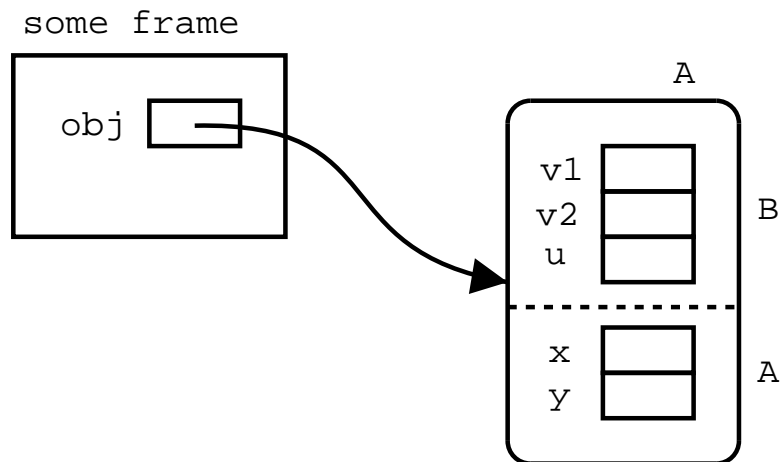
```
class LimitedSavingsAccount
extends SavingsAccount {
    private float daily_limit;
    public LimitedAccount(float initial_balance,
                          float rate, float limit)
    {
        super(initial_balance, rate);
        daily_limit = limit;
    }
    public void withdraw(float amount)
    {
        if (amount < daily_limit)
            balance = balance - amount;
    }
}
```

Inheritance

```
class C { ... }
class D { ... }
class E { ... }
class B {
    C v1, v2;
    D u;
    void m() { ... }
}
class A extends B {
    E x;
    C y;
    void p() { ... }
    void s() { ... }
}
```

Inheritance

```
// In some client
A obj = new A();
obj.p();
obj.m();
// We can refer to ... obj.x ... obj.y ...
// ... obj.u ... obj.v1 ... obj.v2 ...
```



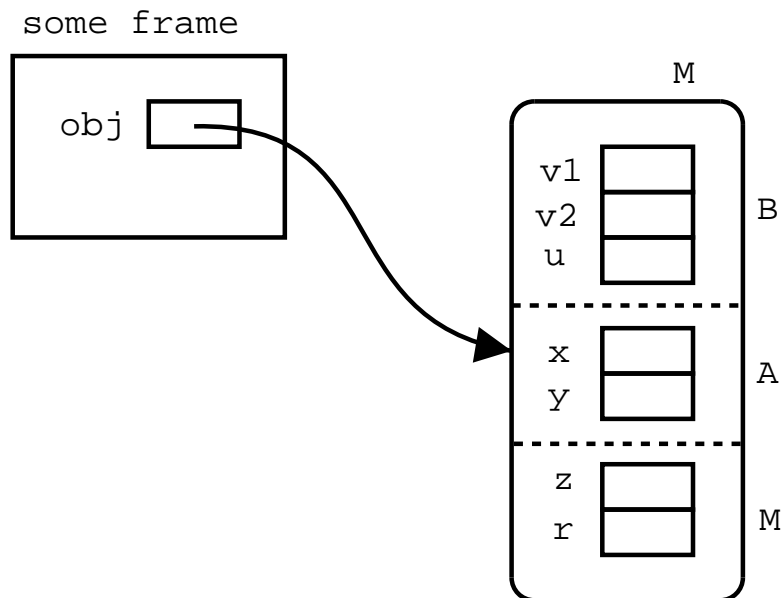
Inheritance

- A method in a subclass can access the attributes and methods of its super class.

```
class C { ... }
class D { ... }
class E { ... }
class B {
    C v1, v2;
    D u;
    void m() { ... v1 ... v2 ... u ... m() ... }
}
class A extends B {
    E x;
    C y;
    void p()
    {
        ... x ... y ... p() ... v1 ...
        ... v2 ... u ... m() ...
    }
    void s() { ... }
}
```

Inheritance

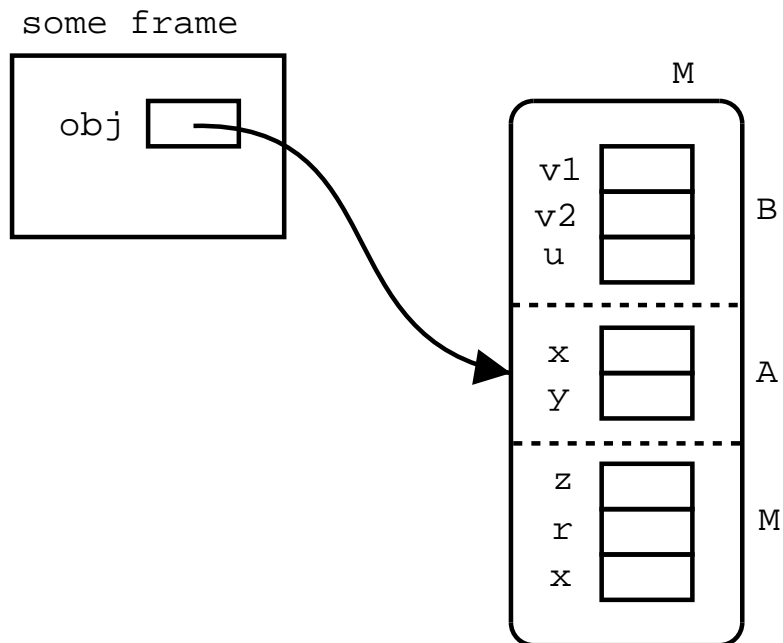
```
class M extends A {  
    E z;  
    D r;  
    void q() { ... }  
}  
// Somewhere else  
M obj2 = new M();
```



Shadowing variables

- An attribute or instance variable can be redefined in a subclass. In this case we say that the variable in the subclass *shadows* the variable in the parent class.

```
class M extends A {  
    E z;  
    D r, x;  
    void q() { ... }  
}
```



Accessing variables from the super class

- The `super` reference is used to access an attribute or method in a parent class.

```
class M extends A {  
    E z;  
    D r, x;  
    void q()  
    {  
        ... this.x ... super.x ...  
    }  
}
```

Overriding methods

- A method can be redefined in a subclass. This is called *overriding* the method.

```
class M extends A {
    E z;
    D r, x;
    void q()
    {
        ... this.x ... super.x ...
    }
    void p()
    {
        ...
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    void m()
    {
        System.out.println("1 ");
    }
}
class B extends A {
    void p()
    {
        System.out.println("2 ");
        m();
    }
}
```

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        A obj1 = new A();
        B obj2 = new B();
        obj1.m();
        obj2.m();
        obj2.p();
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    int x = 3;
    void m()
    {
        System.out.println(x);
    }
}
class B extends A {
    void p()
    {
        System.out.println(x);
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    int x = 3;
    void m()
    {
        System.out.println(x);
    }
}
class B extends A {
    int x = 5;
    void p()
    {
        System.out.println(x);
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    int x = 3;
    void m()
    {
        System.out.println(x);
    }
}
class B extends A {
    int x = 5;
    void p()
    {
        System.out.println(super.x);
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    void m()
    {
        System.out.println("1 ");
    }
}
class B extends A {
    void m()
    {
        System.out.println("3 ");
    }
    void p()
    {
        System.out.println("2 ");
        m();
    }
}
```

Inheritance

- A method in a subclass can access the attributes and methods of a superclass.

```
class A {
    void m()
    {
        System.out.println("1 ");
    }
}
class B extends A {
    void m()
    {
        System.out.println("3 ");
    }
    void p()
    {
        System.out.println("2 ");
        super.m();
    }
}
```

Inheritance

- A method in a superclass can access *indirectly* the attributes and methods of a subclass.

```
class A {
    void m()
    {
        System.out.println("1 ");
    }
    void p()
    {
        System.out.print("2 ");
        m();
    }
}
class B extends A {
    void m()
    {
        System.out.println("3 ");
    }
}
```

Inheritance

- A method in a superclass can access *indirectly* the attributes and methods of a subclass.

```
public class Inh1
{
    public static void main(String[] args)
    {
        A obj1 = new A();
        B obj2 = new B();
        obj1.m();
        obj2.m();
        obj1.p();
        obj2.p();
    }
}
```

Inheritance

- A method in a superclass can access *indirectly* the attributes and methods of a subclass.

```
class A {
    int x = 3;
    void p()
    {
        System.out.println(x);
    }
}
class B extends A {
    int x = 5;
    void q()
    {
        System.out.println(x);
    }
}
```

Inheritance

- A method in a superclass can access *indirectly* the attributes and methods of a subclass.

```
public class Inh1
{
    public static void main(String[] args)
    {
        A obj1 = new A();
        B obj2 = new B();
        obj1.p();
        obj2.p();
        obj2.q();
    }
}
```

Accessing a method or attribute

- When we try to access a method or attribute of an object, it is looked up by the Java runtime system in the class of the object first. If it is not found there, it is looked up in the parent class. If it is not found there, it is looked up in the grand-parent, etc...

```
M obj3 = M();  
obj3.q(); // From class M  
obj3.m(); // From class B  
obj3.p(); // From class M  
obj3.s(); // From class A
```

- Attributes and methods declared as `private` cannot be accessed directly by the subclasses, even though they are present in the object. They can be accessed only indirectly by public accessor methods in the class that declared them as `private`.

Accessing a method or attribute

```
class A extends B {
    private E x, y;
    void p() { }
    void s() { }
    public E get_x() { return x; }
}
class M extends A {
    E z;
    D r, x;
    void q()
    {
        ... this.x ...
        // instead of super.x ...
        ... get_x() ... or ... super.get_x() ...
    }
}
```

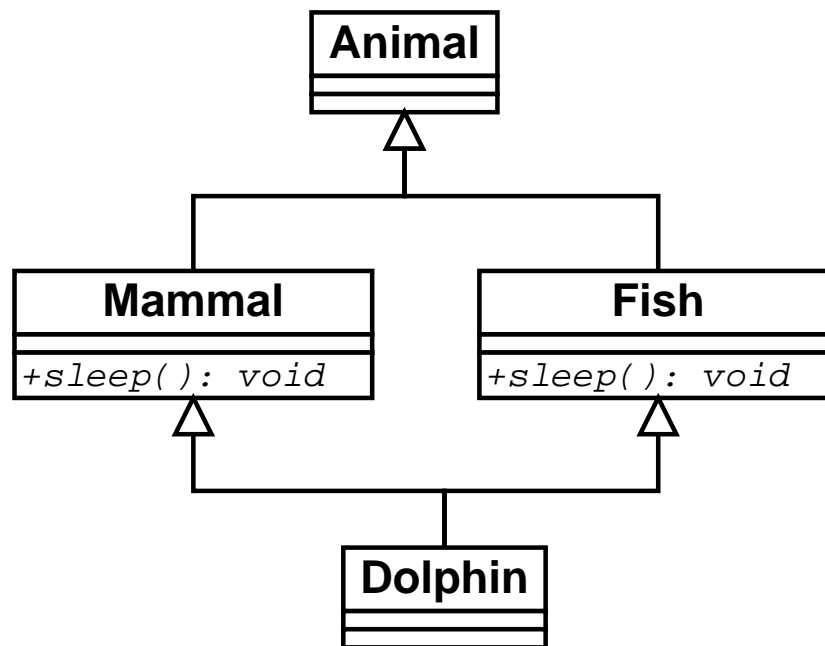
Accessing a method or attribute

- An attribute or method declared as `protected` can be accessed by any subclass, even if it is in a different package.
- An attribute or method declared as `final`, is not inherited at all, i.e. it forbids overriding.
- A class declared as `final`, cannot have subclasses.

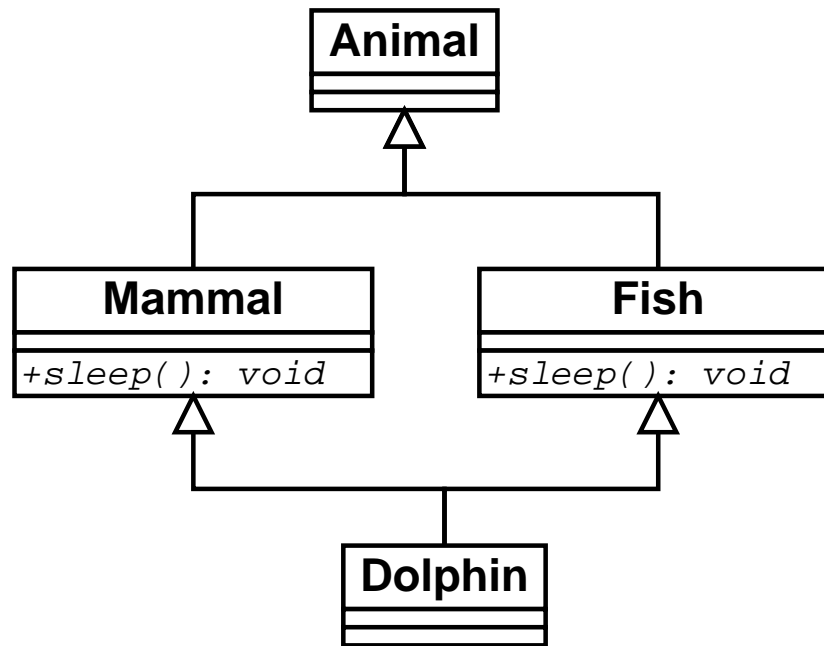
Multiple inheritance

- Multiple inheritance: a class with more than one superclass

Multiple inheritance



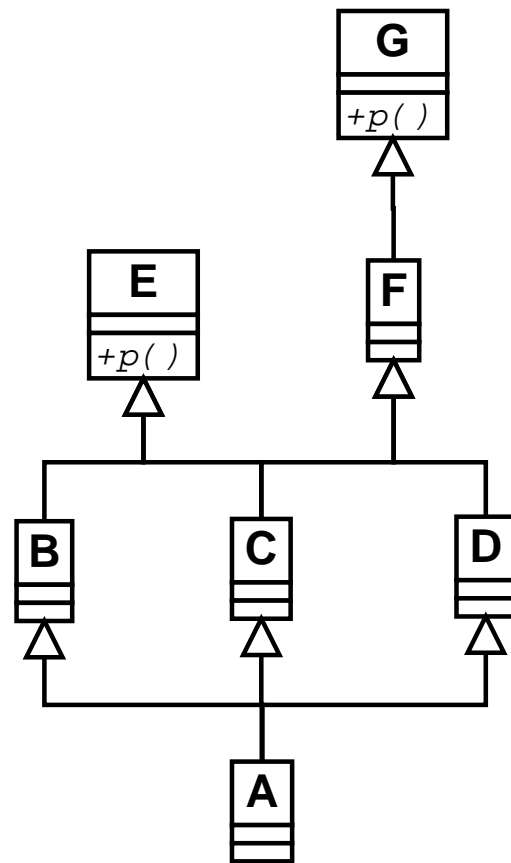
Multiple inheritance



`class A extends B, C { ... } // Error`

- Java does not support multiple inheritance

Multiple inheritance



The end