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 { ... }
```

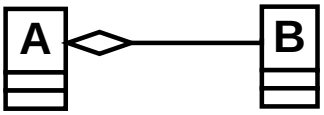
- Inheritance represents the "is-a" relationship

Inheritance

- Roles of inheritance:
  - Represents the *is-a* relationship between classes
  - Describes *specialization*
  - Describes *"being a subset of"*
  - Describes *alternatives* (an Animal is a Dog *or* a Cat *or* a Bird, etc.)
  - A parent class describes the things that all its subclasses have in *common*
  - Allows us to *reuse* definitions by extending classes

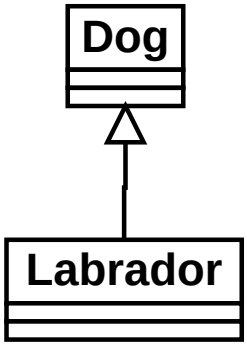
Inheritance

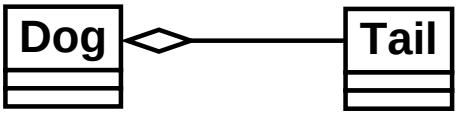
represents:
"every A is a B"
(inheritance)



represents:
"every A has a B"
(aggregation)

For example:





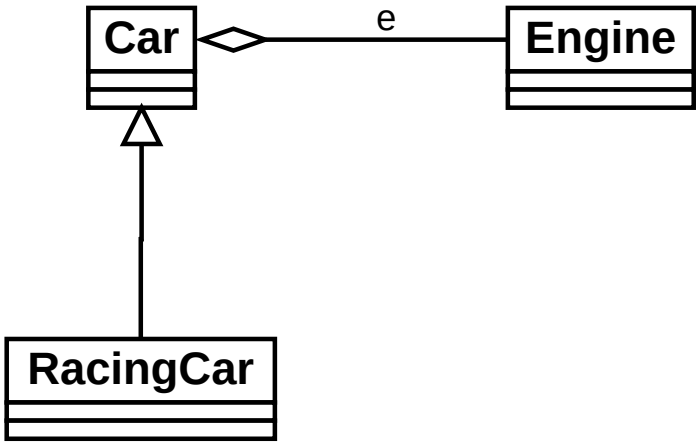
Inheritance

- A subclass *inherits* all of its parent's attributes and methods
- A subclass may add additional features (attributes *and* methods)
- An instance of the subclass has the attributes and methods of the parent in addition to the subclass's own attributes and methods.

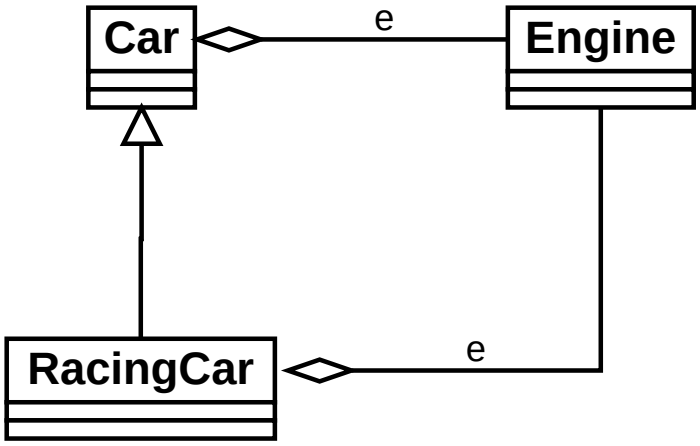
Inheritance

```
class Engine {
  // ...
}

class Car {
  Engine engine;
  // ...
}

class RacingCar extends Car {
  // It implicitly has Engine e;
  // ...
}

// In some client
RacingCar r = new RacingCar();
Engine e1 = r.engine;  // engine is inherited from
```

Inheritance



is the same as



Inheritance

- Inheritance also represents specialization

```
class Engine {
  // ...
}
class Car {
  Engine engine;
  Car() { engine = new Engine(); }
  // ...
}
class RacingCar extends Car {
  Aerofoil aero;
  TurboCharger turbo;
}

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

Inheritance

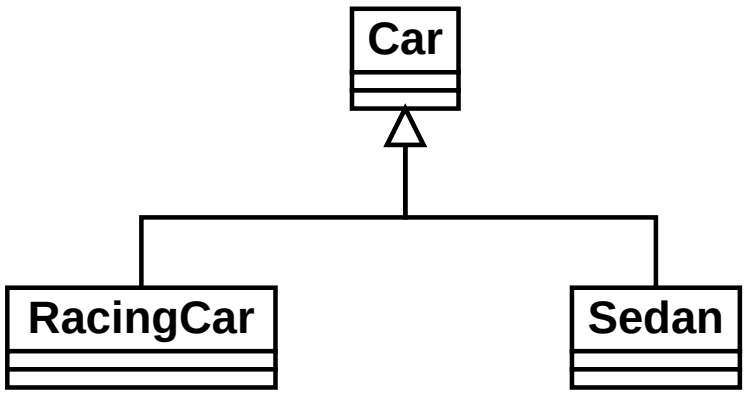
- Methods are inherited too:

```
class Engine {
  void start() { ... }
}
class Car {
  Engine engine;
  double speed;
  Car() { engine = new Engine(); speed = 0.0; }
  void turn_on()
  {
    engine.start();
  }
}
class RacingCar extends Car {
  Aerofoil aero;
  TurboCharger turbo;
}
// In some client
RacingCar r = new RacingCar();
r.turn_on(); // Inherited from Car
```

Inheritance

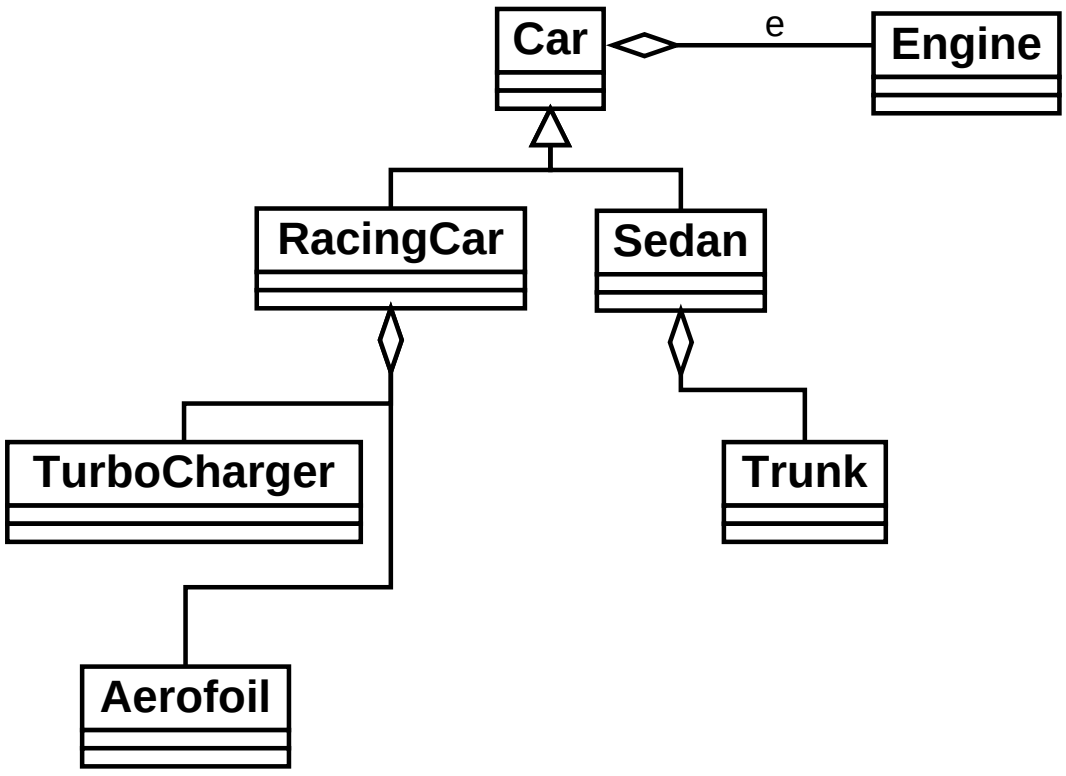
- Classes can have many subclasses

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



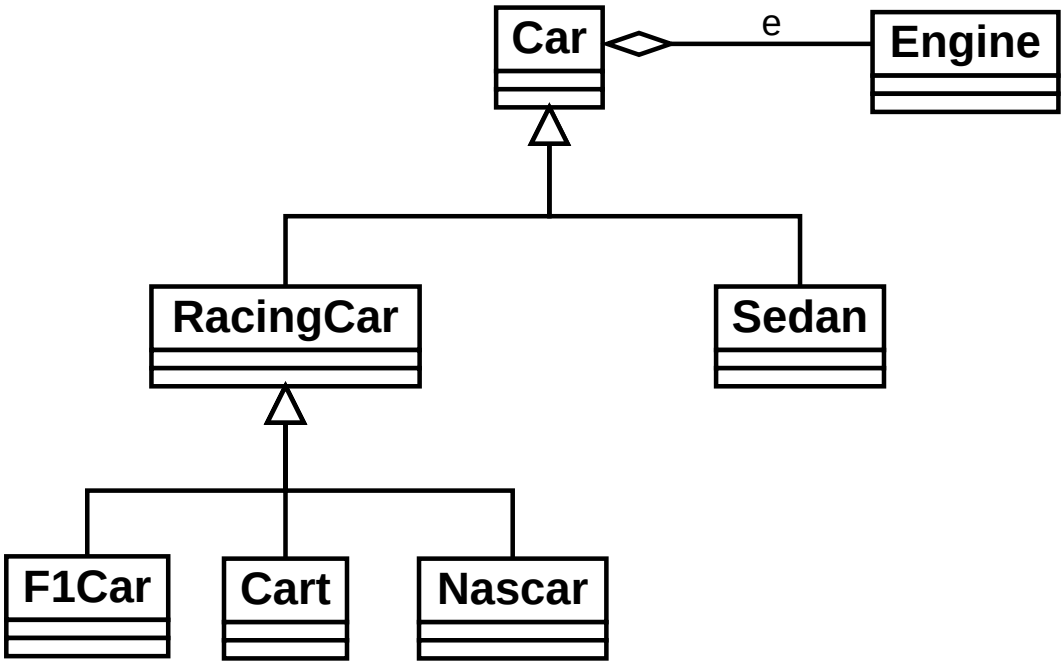
Inheritance

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



Inheritance

- Class hierarchy:



Inheritance

- If in some client we attempt to access an attribute or method in a class, but the attribute or method is not defined in that class, then Java looks for it in the class's parent.

Accessing a method or attribute

- Method (and attribute) lookup:
  - If a method (or attribute) m is applied to an object of type A the method m of class A is executed (or accessed) if it is declared in A.
  - If m is not defined in A and A is a subclass of B then the method m of class B is executed if it is declared in B.
  - If m is not defined in B and B is a subclass of C then the method m of class C is executed if it is declared in C.
  - ...
  - If no "ancestor" of A has a definition of method m then an error occurs.

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.print(“zzz...”);
    tired = false;
  }
}
```

Inheritance

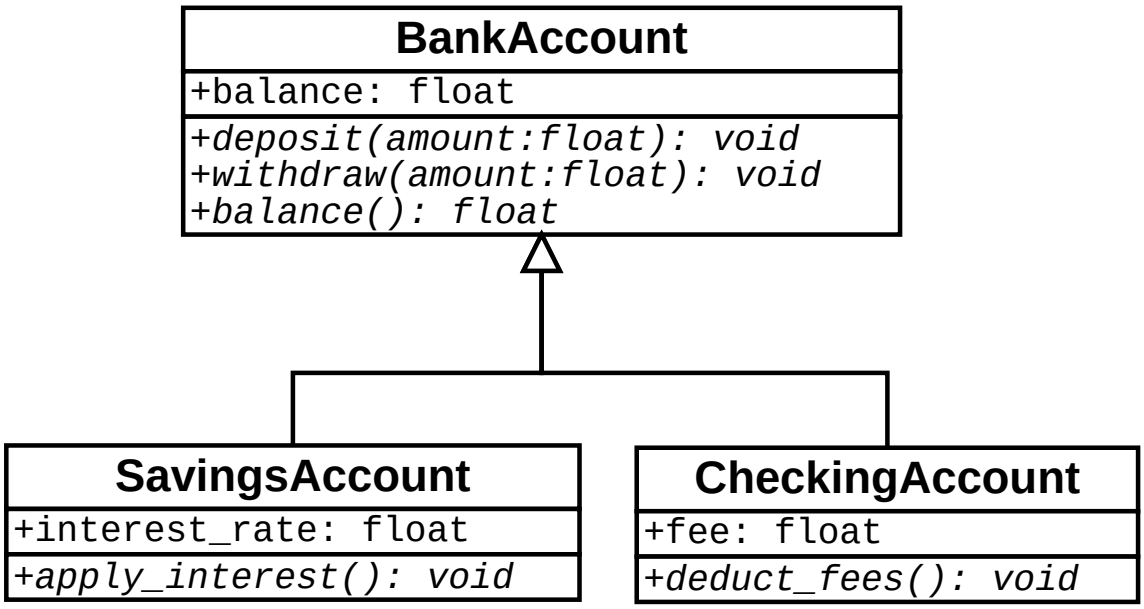
```
class Dog extends Animal {
  Legs[] l;
  Tail t;
  void run()
  {
    tired = true; // From class Animal
    hungry = true;
  }
  void bark()
  {
    System.out.print(“Woof, Woof!”);
  }
}
class Labrador extends Dog {
  void say_hello()
  {
    t.wiggle(); // t from class Dog
  }
}
```

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

- 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; }
}
```

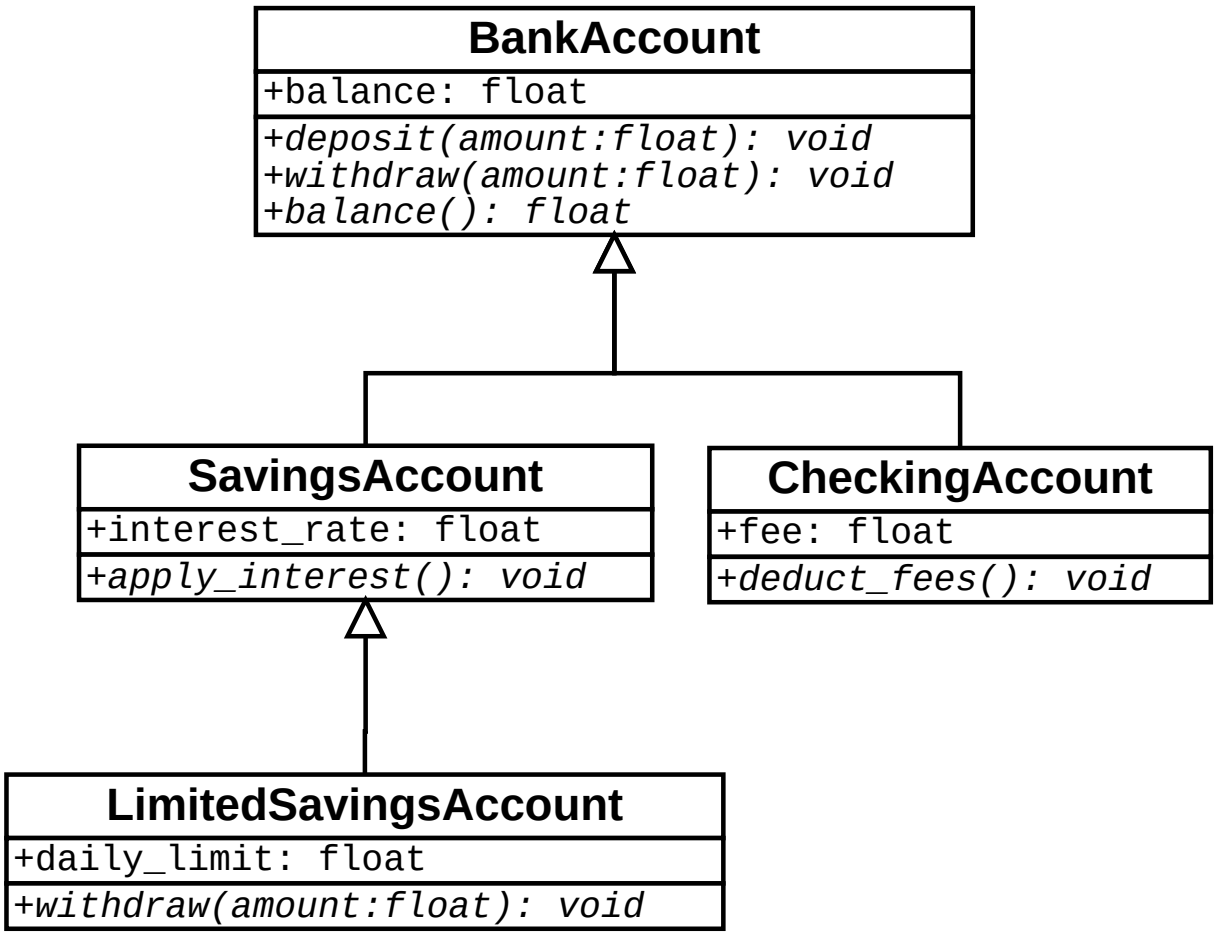
Inheritance

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

Overriding methods

```
public class BankApplication {
  public static void main(String[] args)
  {
    LimitedSavingsAccount a1;
    CheckingAccount a2;
    a1 = new LimitedSavingsAccount(1000.0, 0.2, 20
    a2 = new CheckingAccount(300.0, 3.50);
    a1.withdraw(400.0);
    a1.apply_interest();
    a1.deposit(200.0);
    a2.deduct_fee();
    a2.withdraw(400.0);
  }
}
```

Inheritance

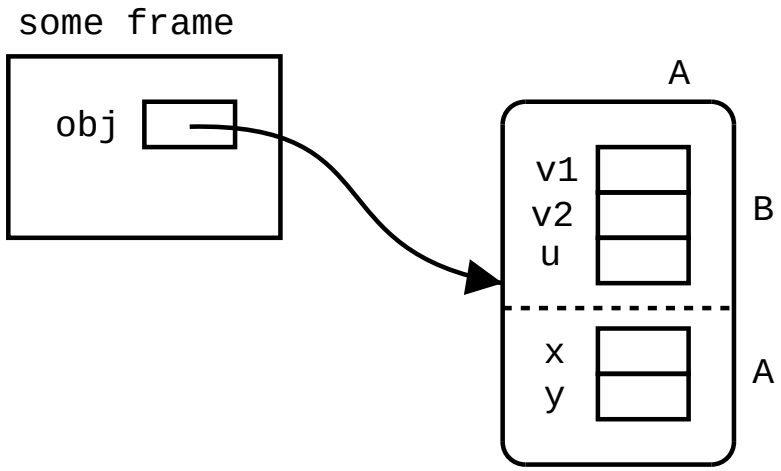
- When a subclass is instantiated, the object created will contain all inherited attributes in addition to its own

Inheritance

```
class B {
  int v1, v2;
  String u;
  void m() { ... }
}
class A extends B {
  double x;
  boolean y;
  void p() { ... }
  void s() { ... }
}
```

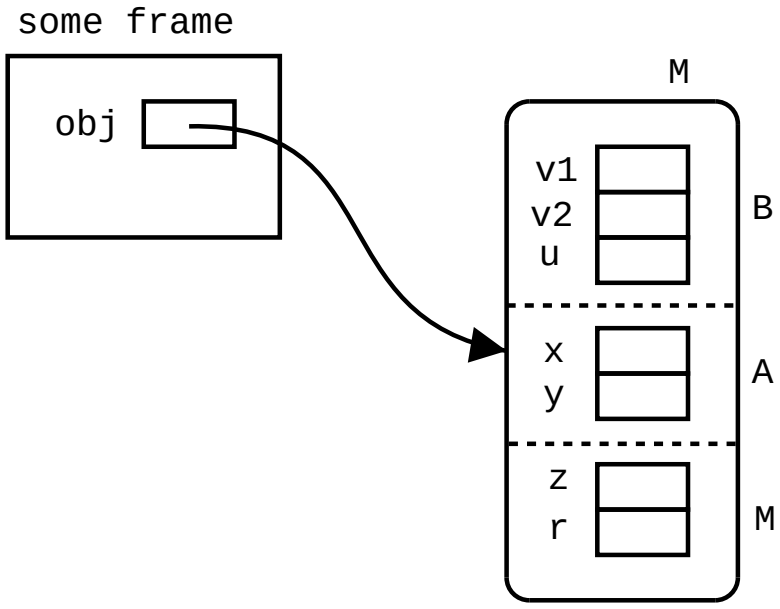
Inheritance

```
// 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

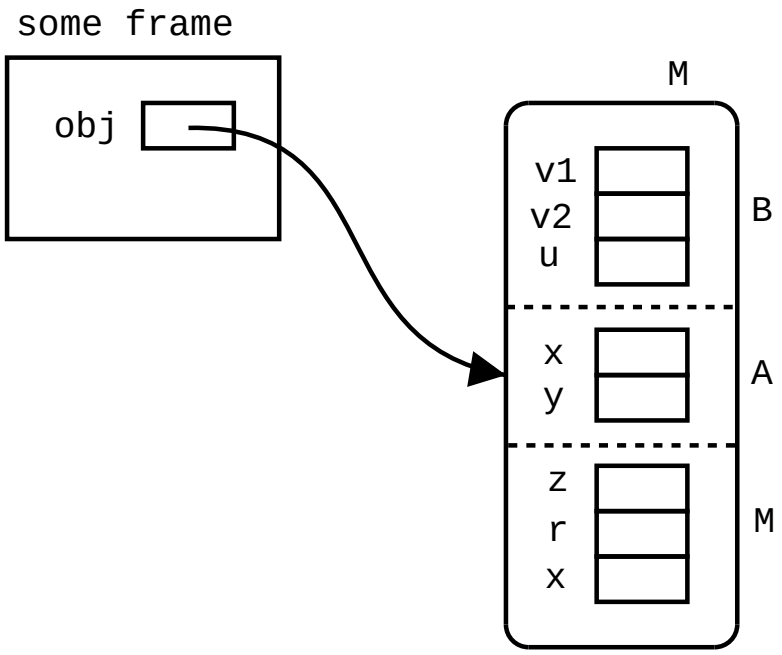
```
class M extends A {
  String z;
  char r;
  void q() { ... }
}
// Somewhere else
M obj2 = new M();
```



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 {
  String z;
  char r, x;
  void q() { ... }
}
```



Inheritance

- Methods in a parent class can be applied to instances of its subclasses

```
class Parent {
    void m()
    {
        System.out.print("1 ");
    }
}
class Child extends Parent {
    void p()
    {
        System.out.print("2 ");
    }
}
```

Inheritance

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

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        Parent obj1 = new Parent();
        Child obj2 = new Child();
        obj1.m();   // Prints "1"
        obj2.m();   // Prints "1"
        obj2.p();   // Prints "2"
    }
}
```

Inheritance

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

```
class Parent {
    void m()
    {
        System.out.print("1 ");
    }
}
class Child extends Parent {
    void p()
    {
        System.out.print("2 ");
        m();
    }
}
```

Inheritance

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

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        Parent obj1 = new Parent();
        Child obj2 = new Child();
        obj1.m();   // Prints "1"
        obj2.m();   // Prints "1"
        obj2.p();   // Prints "2 1"
    }
}
```

Inheritance

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

```
class Parent {
    int t = 3;
    void m()
    {
        System.out.print(t);
    }
}
class Child extends Parent {
    void p()
    {
        System.out.print(t);
    }
}
```

Inheritance

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

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        Parent obj1 = new Parent();
        Child obj2 = new Child();
        obj1.m();   // Prints "3"
        obj2.m();   // Prints "3"
        obj2.p();   // Prints "3"
    }
}
```

Inheritance

- *Shadowing a variable*: if class A has an attribute n and a subclass B of A also declares an attribute n, then n of B *shadows* n of A.

```
class Parent {
    int t = 3;
}
class Child extends Parent {
    int t = 5;
}
```

- If an instance of B is created it will contain both variables. Shadowed variables are also inherited, but can be accessed only by using the special reference `super`.

Inheritance

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

```
class Parent {
    int t = 3;
    void m()
    {
        System.out.print(t);
    }
}
class Child extends Parent {
    int t = 5;
    void p()
    {
        System.out.print(t);
    }
}
```

Inheritance

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

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        Parent obj1 = new Parent();
        Child obj2 = new Child();
        obj1.m();   // Prints "3"
        obj2.m();   // Prints "3"
        obj2.p();   // Prints "5"
    }
}
```

Inheritance

- When a variable is shadowed, we can use `super` to access it

```
class Parent {
    int t = 3;
    void m()
    {
        System.out.print(t);
    }
}
class Child extends Parent {
    int t = 5;
    void p()
    {
        System.out.print(super.t);
    }
}
```

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        Parent obj1 = new Parent();
        Child obj2 = new Child();
        obj1.m();   // Prints "3"
        obj2.m();   // Prints "3"
        obj2.p();   // Prints "3"
    }
}
```

Inheritance

- *Overriding a method*: if class A has a method m and a subclass B of A also declares a method called m, then m of B *overrides* m of A.

```
class Parent {
    void m()
    {
        System.out.print("1 ");
    }
}
class Child extends Parent {
    void m()
    {
        System.out.print("2 ");
    }
}
```

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        Parent obj1 = new Parent();
        Child obj2 = new Child();
        obj1.m();   // Prints "1"
        obj2.m();   // Prints "2"
    }
}
```

Inheritance

- A method can be redefined in a subclass

```
class Parent {
    void m()
    {
        System.out.print("1 ");
    }
}
class Child extends Parent {
    void m()
    {
        System.out.print("2 ");
    }
    void p()
    {
        System.out.print("3 ");
        m();
    }
}
```

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        Parent obj1 = new Parent();
        Child obj2 = new Child();
        obj1.m();   // Prints "1"
        obj2.m();   // Prints "2"
        obj2.p();   // Prints "3 2"
    }
}
```

Inheritance

- A method can be redefined in a subclass

```
class Parent {
    void m()
    {
        System.out.print("1 ");
    }
}
class Child extends Parent {
    void m()
    {
        System.out.print("2 ");
    }
    void p()
    {
        System.out.print("3 ");
        super.m();
    }
}
```

Inheritance

```
public class Inh0
{
    public static void main(String[] args)
    {
        Parent obj1 = new Parent();
        Child obj2 = new Child();
        obj1.m();   // Prints "1"
        obj2.m();   // Prints "2"
        obj2.p();   // Prints "3 1"
    }
}
```

Inheritance

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

```
class Parent {
    void m()
    {
        System.out.print("1 ");
    }
    void q()
    {
        System.out.print("2 ");
        m();
    }
}
class Child extends Parent {
    void m()
    {
        System.out.print("3 ");
    }
}
```

Inheritance

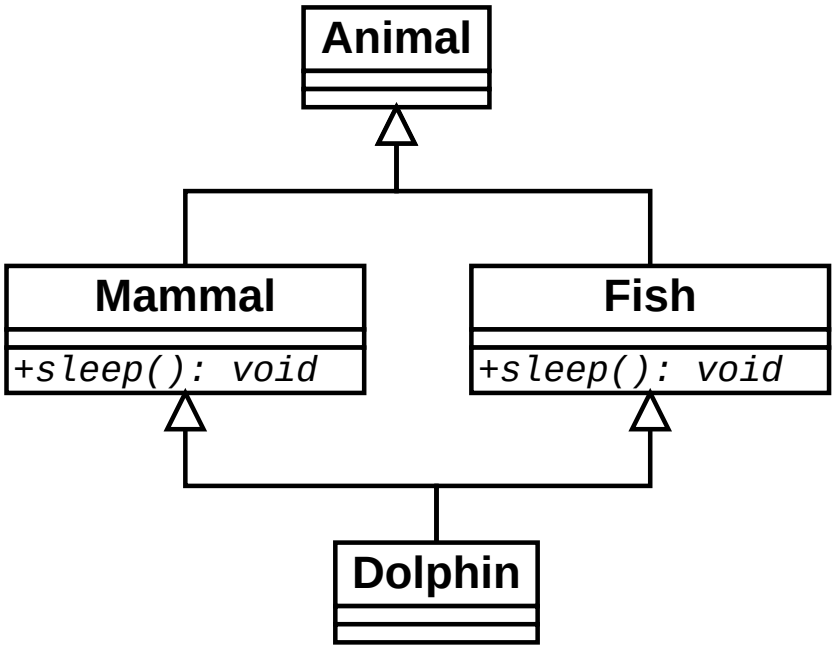
- A method in a superclass can access *indirectly* the attributes and methods of a subclass.

```
public class Inh1
{
    public static void main(String[] args)
    {
        Parent obj1 = new Parent();
        Child obj2 = new Child();
        obj1.m();   // Prints "1"
        obj2.m();   // Prints "3"
        obj1.q();   // Prints "2 1"
        obj2.q();   // Prints "2 3"
    }
}
```

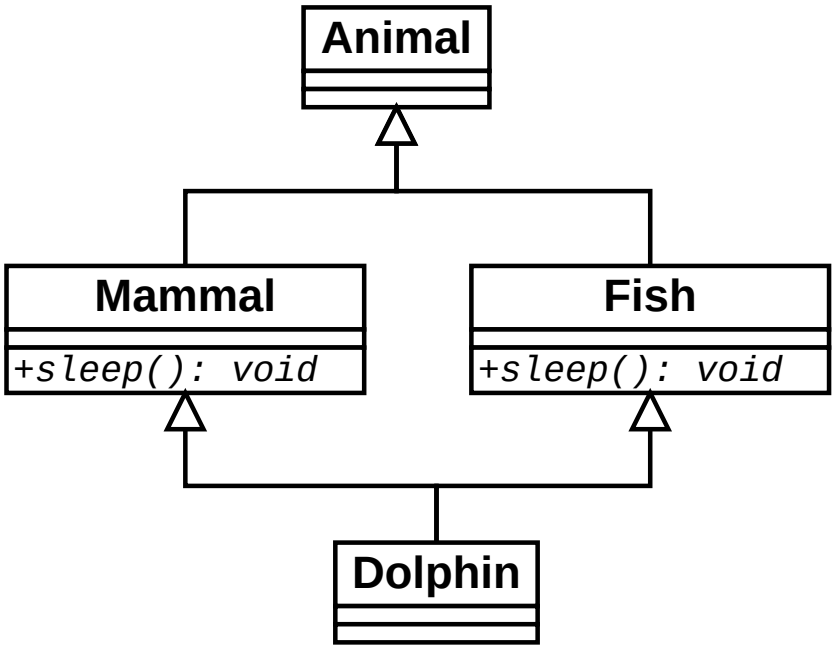
Multiple inheritance

- Multiple inheritance: a class with more than one superclass

Multiple inheritance

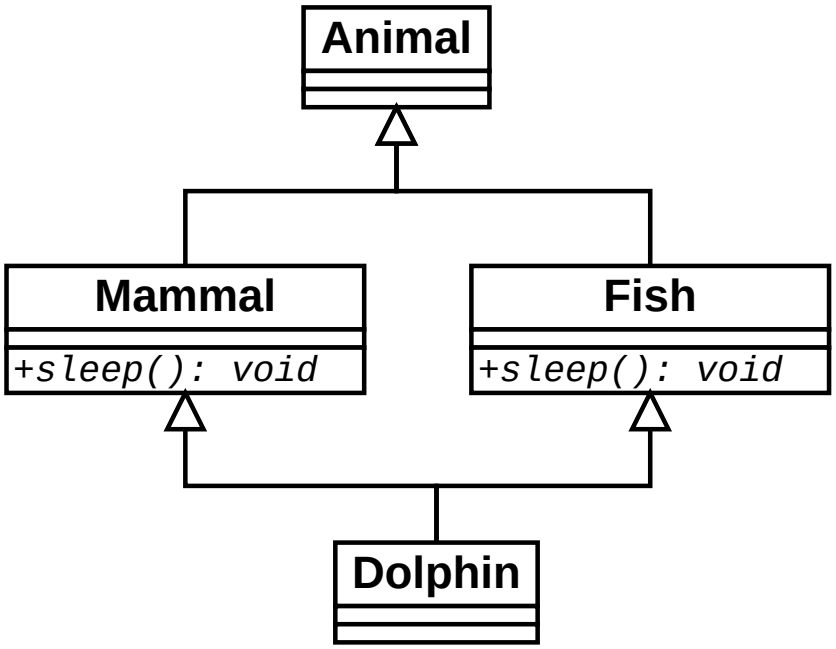


Multiple inheritance



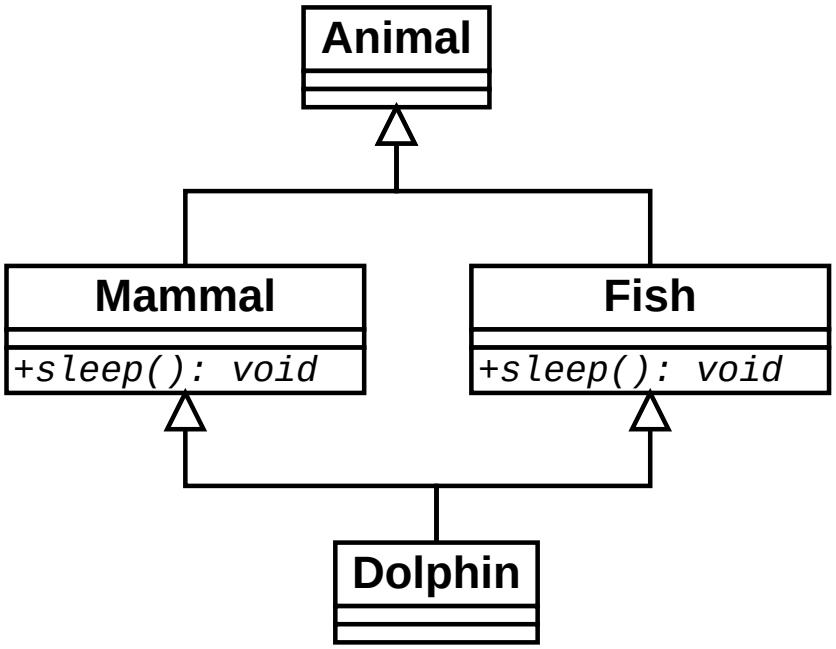

```
class Dolphin extends Mammal, Fish {
  void sleep() { ... }
}
// elsewhere
Dolphin flipper = new Dolphin();
flipper.sleep();
```

Multiple inheritance




```
class Dolphin extends Mammal, Fish {
  void sleep() { ... }
}
// elsewhere
Dolphin flipper = new Dolphin();
flipper.sleep();  // Which one?
```

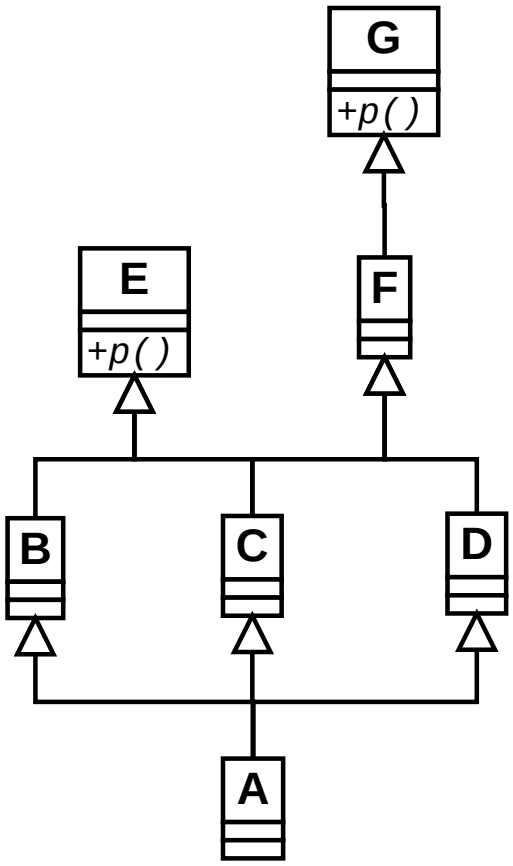
Multiple inheritance




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

- Java does not support multiple inheritance

Multiple inheritance



Accessing the superclass' constructor

In a constructor we can invoke the constructor of the parent class:

```
class Parent {
  int x;
  Parent(int n) { x = n; }
}
class Child extends Parent {
  Child()
  {
    super(5);
  }
}
```

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

```
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;
  }
}
```

Remarks on constructors

```
class A {
    String s;
    A(String q)
    {
        s = "hello "+q;
    }
}

public class ConsTest {
    public static void main(String[] args)
    {
        A x = new A();  // Error
        System.out.println(x.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A(String q)
    {
        s = "hello "+q;
    }
}

public class ConsTest {
    public static void main(String[] args)
    {
        A x = new A(“bye”);
        System.out.println(x.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A() { s = "bonjour "; }
    A(String q)
    {
        s = "hello "+q;
    }
}

public class ConsTest {
    public static void main(String[] args)
    {
        A x = new A();
        System.out.println(x.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A()
    {
        s = "hello ";
    }
}
class B extends A {
    int n;
}
public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B();
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A(String q)
    {
        s = "ask "+q;
    }
}
class B extends A {
    int n;
}
public class ConsTest
{
    public static void main(String[] args)
    {
        B b1 = new B(); //Error
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A() { s = "hello "; }
}

class B extends A {
    int n;
    B(int i)
    {
        n = i;
    }
}

public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A(String q) { s = "hello "+q; }
}
class B extends A {
    int n;
    B(int i)
    {   // Error: no A()
        n = i;
    }
}
public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A(String q) { s = "hello "+q; }
}
class B extends A {
    int n;
    B(int i)
    {
        super(“bye”);
        n = i;
    }
}
public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A()  { s = "bye "; }
    A(String q) { s = "hello "+q; }
}
class B extends A {
    int n;
    B(int i)
    {
        super("salut");
        n = i;
    }
}
public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

- If a class `A` does not have a constructor, then it implicitly has a default constructor with no parameters

  ```
  A() { super(); }
  ```

- If a class `A` has a constructor with parameters, then it does not implicitely have a default constructor `A()`

- Constructors are not inherited

- All constructors have an implicit call to the superclass's default constructor, unless it explicitely calls a non-default constructor from the parent.

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.

The end