OOP with Java

Yuanbin Wu
cs@ecnu

OOP with Java

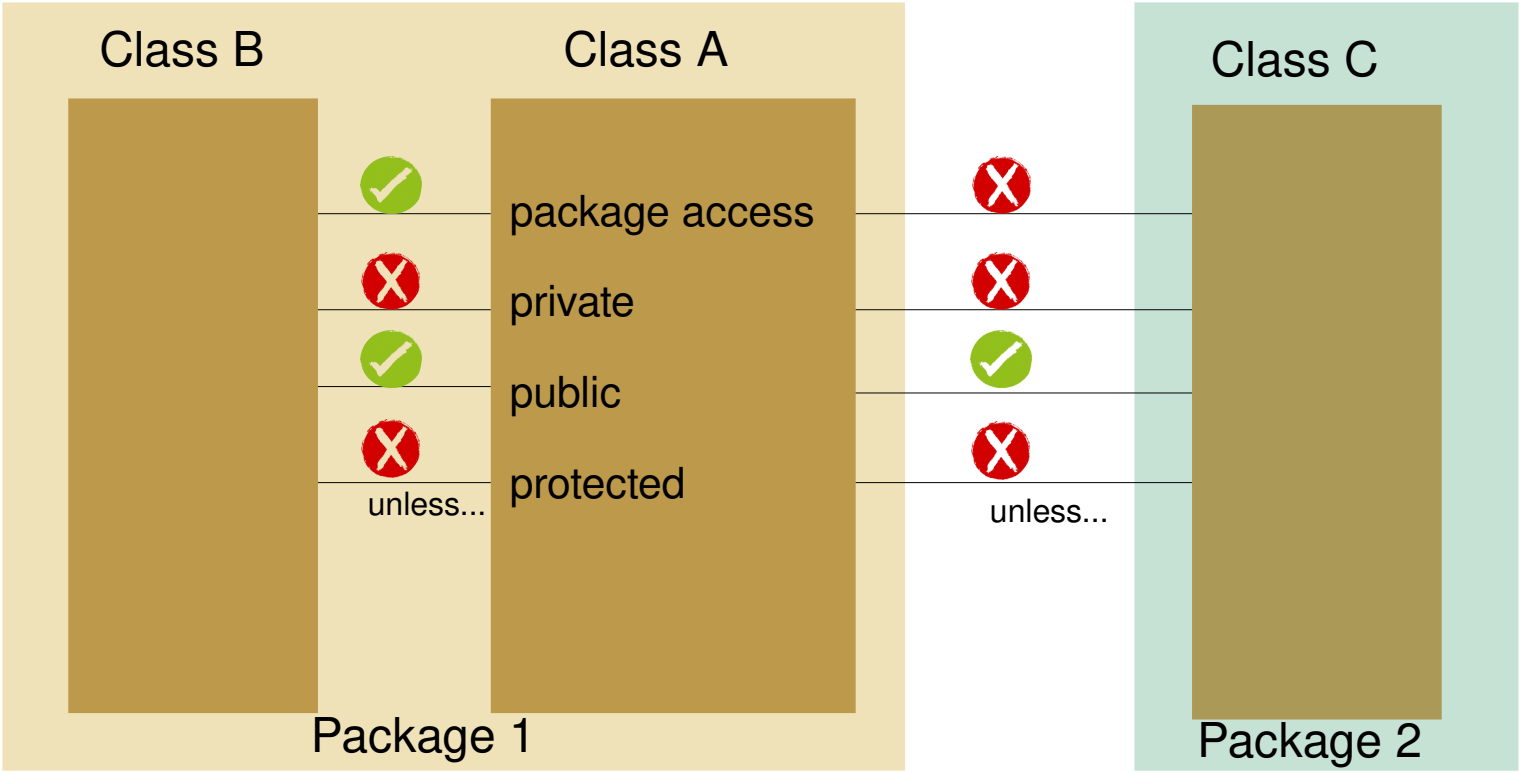
- 通知
  - Project 4: 5 月 2 日晚 9 点

- 复习
  - Java 包
    - 创建包 : package 语句，包结构与目录结构一致
    - 使用包 : import

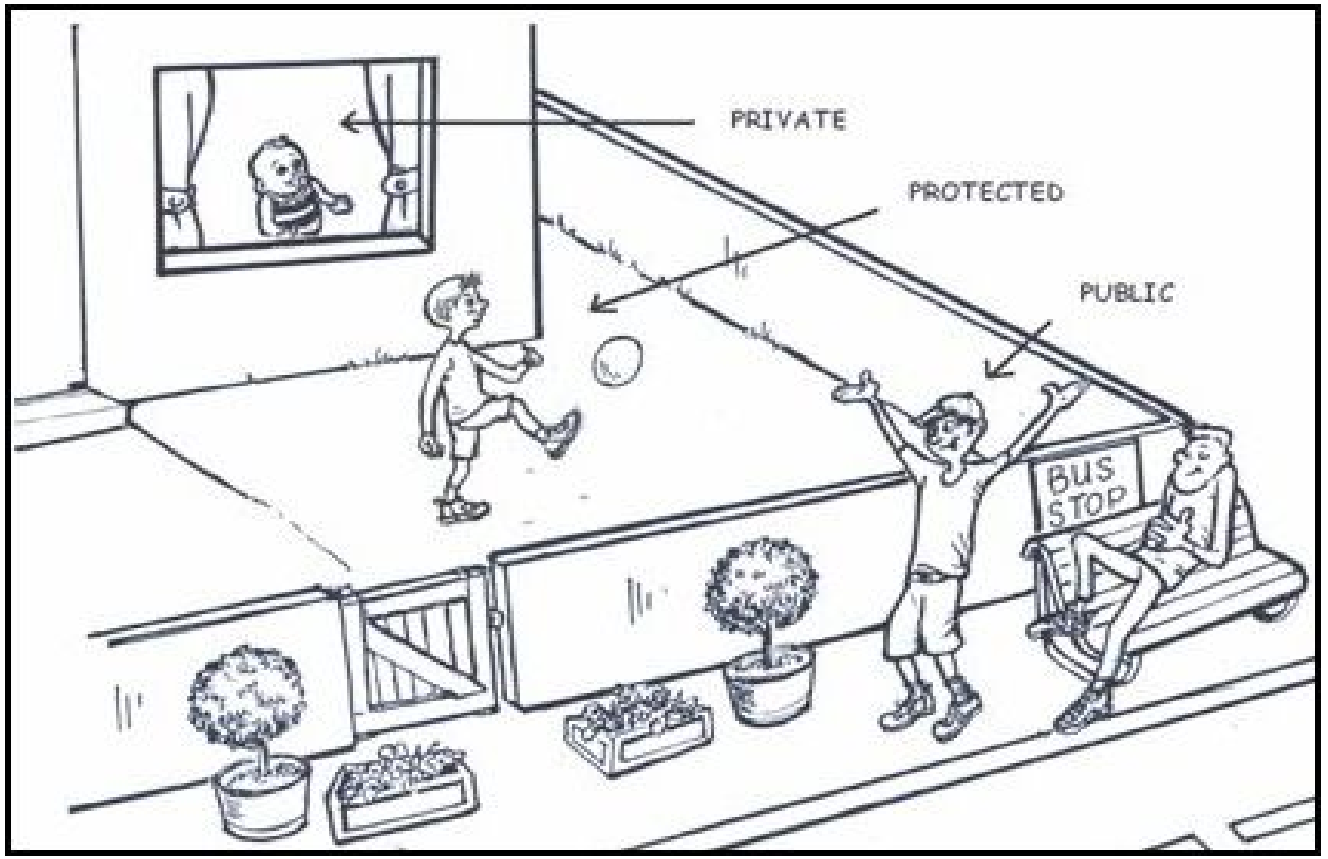
```
restaurant/
    - people/
        - Cook.class
        - Waiter.class
    - tools/
        - Fork.class
        - Table.class
```

```
import restaurant.people.Cook;
import restaurant.tools.Fork;
import restaurant.tools.*;
import restaurant.*;
```

- 复习
  - 访问控制
    - 对类的成员 (数据 , 方法) 的一种修饰
    - 对哪些外部对象是可见的 ?
    - package access (default package), public, private, protected



- 复习
  - 为什么需要访问控制？封装
    - 将易变的与稳定的部分区分开
    - 在满足需求的情况下，接口尽量简单



OOP with Java

- 类的复用
- 组合
- 继承
- 组合与继承

类的复用

- 类的复用 (reuse classes)
  - 问题：如何通过已有类来定义新的类
    - 已有类 A，创建类 B
    - B 有部分功能与 A 重合
  - 例子
    - 已有 Car 类，创建 Transformer 类？

类的复用

Copy and Paste

```
class A {

    // data
    // ...

    // methods
    //…

}
```

```
class B {

    // data of A
    // ...

    // methods of A
    // …

    // new things
}
```

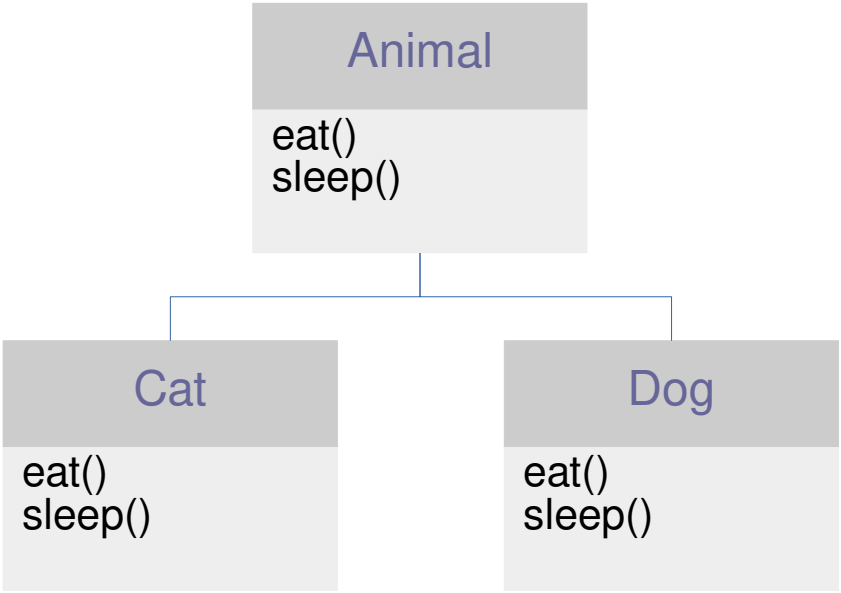
类的复用

- 情况 1
  - class B 中包含 class A 类型的数据成员
  - 例如 :
    - 引擎类 : class Engine
    - 轮胎类 : class Wheel
    - 离合器类 : class Clutch
    - 汽车类 ?
  - “has-a” 关系

组合 (composition)

类的复用

- 情况 2
  - class B 带有 class A 所有的数据和方法成员，同时增加新的成员，或者修改原有的成员
  - 例如：
    - 跑车类具有汽车类的所有方法
  - “is-a” 关系
    - A cat is an animal
    - A dog is an animal



继承 (Inheritance)

类的复用

- 重复使用已有类的两种方式
  - 组合 (composition)
  - 继承 (inheritance)

组合

- 将已有类的对象作为新类的数据成员

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MyCompType {
    private MyType m = new MyType();
    private String s;
    public MyCompType(){
        s = new String("Hello");
    }
}
```

组合

- 初始化 (复习)
  - 默认初始化 (null)
  - 定义时初始化
  - 构造函数初始化
  - 用时初始化
    - 当需要使用该成员时再初始化

组合

- 一种常见的重用方式
- 广义的说，MyType 类可视为对基本类型的重用

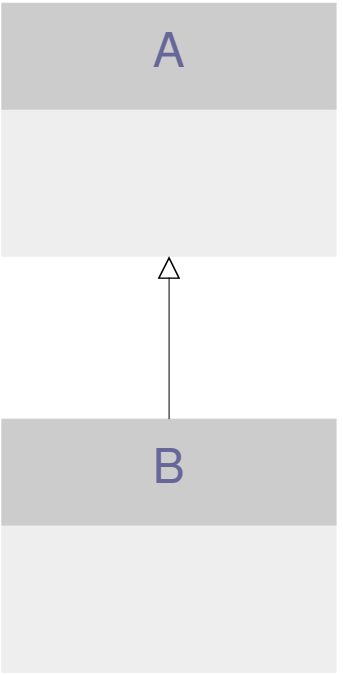
```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
class MyType {
    public Integer i;
    public Double d;
    public Character c;
    private String s;
    private Random r;
    private Picture p;

    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

继承

- 新类包含已有类的方法和数据，并可修改 / 增添
- 语法 : extends
  - A 称为父类 (super class) 或基类 (base class)
  - B 称为子类 (sub-class)

```
class A{
  …
}

public class B extends A {
  …
}
```



继承

1. 子类有父类的所有方法和数据 .

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
    }
}
```

继承

2. 子类可以定义新的方法和数据 .

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{
    String s = new String("Hello");
    public double add(double d){
        return this.d + d;
    }
    public double add(String s){
        return this.s + s;
    }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        System.out.println(ms.get());
        System.out.println(ms.add(1.0));
        System.out.println(ms.add("World"));
    }
}
```

继承

3. 子类可以更新父类的方法，称为**重写 (overriding)**

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{
    public void set(double x){ i = (int)x; }
    public double get() { return i; }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
        System.out.println(ms.d);
    }
}
```

继承

- 继承的基本功能
  - 子类有父类的所有方法和数据
  - 子类可以定义新的方法和数据
  - 子类可以重写 (override) 父类的方法

继承

- 当定义一个子类时发生了什么？
  - 可能性 1：copy&paste 父类的接口和数据，创建一个新的类

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MySubType {
    /*
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
    */

    public string s;
    public childMethods() {...}

}
```

继承

- 当定义一个子类时发生了什么？
  - 可能性 2: 创建一个新的类，包含一个父类的对象作为数据成员 (组合 !)

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

COMPOSITION

```
public class MySubType {
    /*
     public MyType m;
    */

    public string s;
    public childMethods() {...}

}
```

继承

- super 关键字
  - 子类的对象包含一个隐藏的父类对象
  - 在子类中 , super 为该父类对象的引用
  - 复习 : this 关键字

```
public class MySubType extends MyType{
    /*
     MyType _super;
     MySubType _this;
    */

    public string s;
    public childMethods() {...}
}
```

- 作用
  - 当方法被重写时，可以通过 super 调用父类的方法

继承

- 构造函数
  - 在子类构造函数调用前，首先调用父类构造函数

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
    public MyType(){
        System.out.println("In base class");
    }
}
```

```
public class MySubType extends MyType{
    public MySubType (){
        System.out.println("In sub class");
    }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
    }
}
```

```
class MySubSubType extends MySubType{
    public MySubSubType (){
        System.out.println("In sub sub class");
    }
}
```

继承

- 构造函数
  - 调用父类带参数的构造函数
  - 必须出现在子类构造函数的首行

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
    public MyType(){
        System.out.println(“In base class”);
    }
    public MyType(double d){
        this.d = d;
    }
}
```

```
public class MySubType extends MyType{
    public MySubType (){
        super(1.0);
        System.out.println(“In sub class”);
    }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
    }
}
```

- 例子

```
public class Circle {
   public double radius;
   public String color;

   public Circle() {
      this.radius = 1.0;
      this.color = "red";
   }
   public Circle(double radius) {
      this.radius = radius;
      this.color = "red";
   }
   public Circle(double radius, String color) {
      this.radius = radius;
      this.color = color;
   }

   // Return the area of this Circle
   public double getArea() {
      return radius * radius * Math.PI;
   }

   public String toString() {
      return "This is a Circle";
   }
}
```

```
public class Cylinder extends Circle {

   public double height;

   public Cylinder() {
      super();
      this.height = 1.0;
   }
   public Cylinder(double height) {
      super();
      this.height = height;
   }
   public Cylinder(double height, double radius) {
      super(radius);
      this.height = height;
   }
   public Cylinder(double height,
                   double radius, String color) {
      super(radius, color);
      this.height = height;
   }

   // Return the volume of this Cylinder
   public double getVolume() {
      return getArea()*height;   // Use Circle's getArea()
   }

   // overriding
   public double getArea() {
      return 2 * radius * Math.PI*height + 2*super.getArea();
   }

   public String toString() {
      return "This is a Cylinder";
   }
}
```

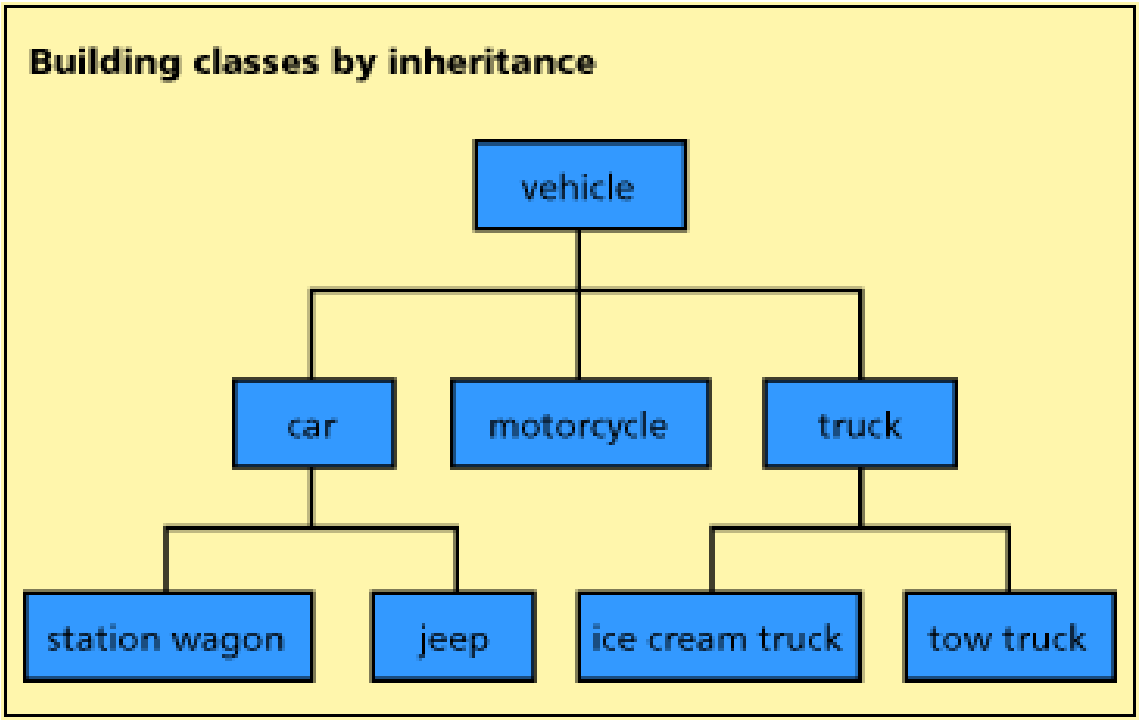
- 例子

```
public class Bicycle {

    public int cadence;
    public int gear;
    public int speed;

    // constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public void setGear(int newValue) {
        gear = newValue;
    }

    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }

}
```

```
public class MountainBike extends Bicycle {

    // the MountainBike subclass adds one field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int startHeight,
                        int startCadence,
                        int startSpeed,
                        int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

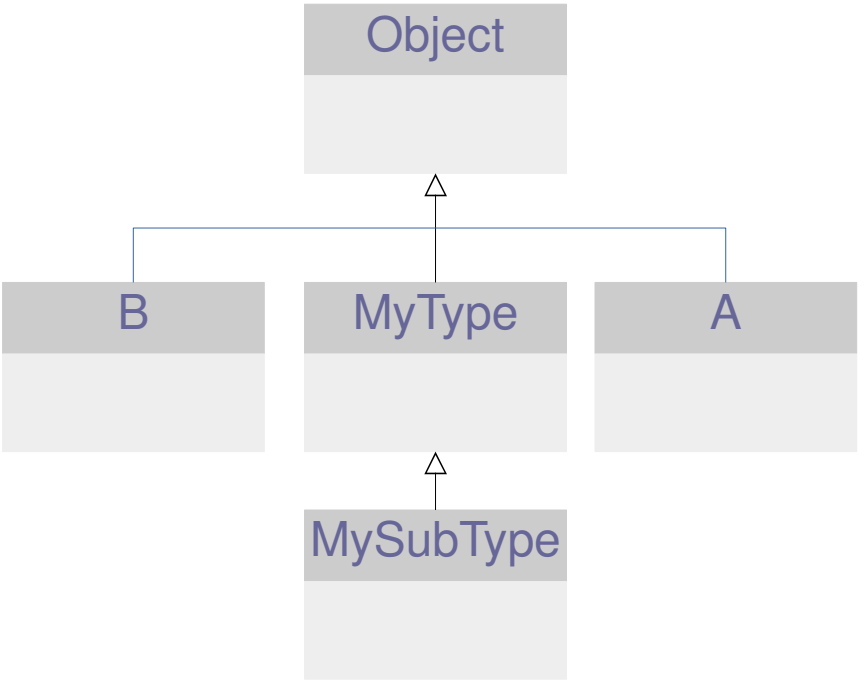
继承

- 不同类之间通过 **父类-子类** 关系构成一棵树



继承

- Object class
  - 每个类都是 Object class 的子类
  - Single root class hierarchy tree
  - toString(), equals(),..
  - Let's try



继承

- 重写 (override)
  - 子类重新实现父类的方法 (同一个函数)
- 重载 (overload)
  - 相同函数名，不同参数列表

继承

- 例子

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public void set(int y) {i = y;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{
    public void set(double x){ i = (int)x; }
    public void set(char z) {c = z; }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
        System.out.println(ms.d);
    }
}
```

继承

- 例子

```
class MyType {
    public int i;
    public double d;
    public char c;
    private void set(double x) { d = x;}
    private void set(int y) {i = y;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{
    public void set(double x){ i = (int)x; }
    public void set(char z) {c = z; }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
        System.out.println(ms.d);
    }
}
```

组合与继承

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MyCompType {
    private MyType m = new MyType();
    private String s;
    public MyCompType(){
        s = new String(“Hello”);
    }
}
```

```
public class MySubType extends MyType{

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
    }
}
```

组合与继承

- 同时使用组合与继承

```
public class MySubType extends MyType{
    String s = new String("Hello");
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
    }
}
```

组合与继承

- 比较
  - B, C 对象都包含一个 A 的对象
  - 访问方式不同
    - b.a.get(); b.a.set(1);
    - c.get(); c.set(1);
  - 设计角度：类间关系不同
    - has-a 关系
    - is-a 关系

```
class A{
  …
  public get(){}
  public set(int i){}
}

class B{
  public A a = new A();
}

class C extends A {
  …
}
```

组合与继承

- 没有 is-a 关系，但需能调用另一类的所有方法

```
class SpaceShipControls{
  void up(int v) {}
  void down(int v) {}
  void left(int v) {}
  void right(int v) {}
  void forward(int v) {}
  void backward(int v) {}
}
```

```
class SpaceShip extends SpaceShipControls{
 …
  Static public void main(String []args){
      SpaceShip s = new SpaceShip();
      s.up(); s.forward();
  }
}
```

代理 (Delegation)
介于组合与继承之间

```
class SpaceShip {
  Private SpaceShipControls s;
  public void up() {s.up();}
  public void down() {s.down();}
  public void left() {s.left();}
  public void up() {s.right();}
  public void forward() {s.forward();}
  public void backward() {s.backward();}
}
```

总结

- 组合
  - 类 B 包含类 A 作为数据成员
  - has-a
- 继承
  - 类 B 具有类 A 的所有数据与方法，并能增添修改
  - Is-a
  - 方法重写 (override)