

OOP with Java

Yuanbin Wu
cs@ecnu

OOP with Java

- 通知
 - Project 4: 4 月 18 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.*;
```

- 访问控制

- package access (default package)
 - public, private, protected

- 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 (reusing classes)
 - 如何通过已有类来定义新的类
 - copy and paste?

类的复用

- 情况 1
 - class B 中包含 class A 类型的数据成员
 - class B 复用了 class A
 - 例如：汽车类包含一个轮胎类的数据成员
 - “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，同时在此基础上增加一些新的成员 / 修改原有的成员
- class B 复用了 class A
- 例如：跑车类具有汽车类的所有方法
- “is-a” 关系

继承 (Inheritance)

组合

- 将已有的类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");  
    }  
}
```

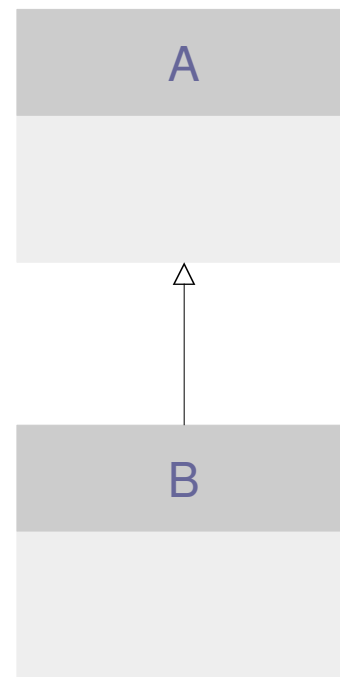

组合

- 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

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 称为父类或基类 (base class)
 - B 称为子类 (sub-class)

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



继承

- 例子

```
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);  
    }  
}
```

1. 子类有父类的所有方法和数据。

继承

- 例子

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

2. 子类可以定义新的方法和数据。

```
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

- 例子

```
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);  
    }  
}
```

3. 子类可以更新父类的方法，称为**重写 (overriding)**

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

- **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** 用来指代父类对象的引用
 - **this** 关键字

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

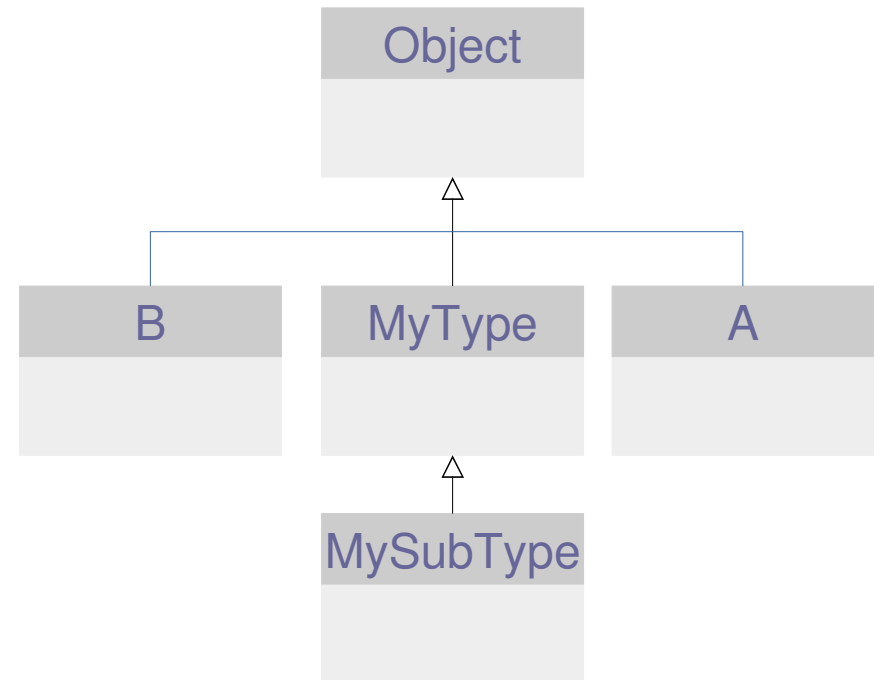
- 构造函数
 - 带参数的构造函数

```
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 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();  
    }  
}
```

继承

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



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();
  }
}
```

```
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();}
}
```

代理 (Delegation)

介于组合与继承之间

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)