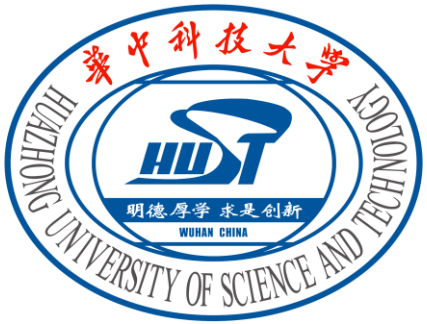
基于Java的面向对象程序设计

陈维亚

weiya_chen@hust.edu.cn

华中科技大学软件学院

第5讲：继承



目录

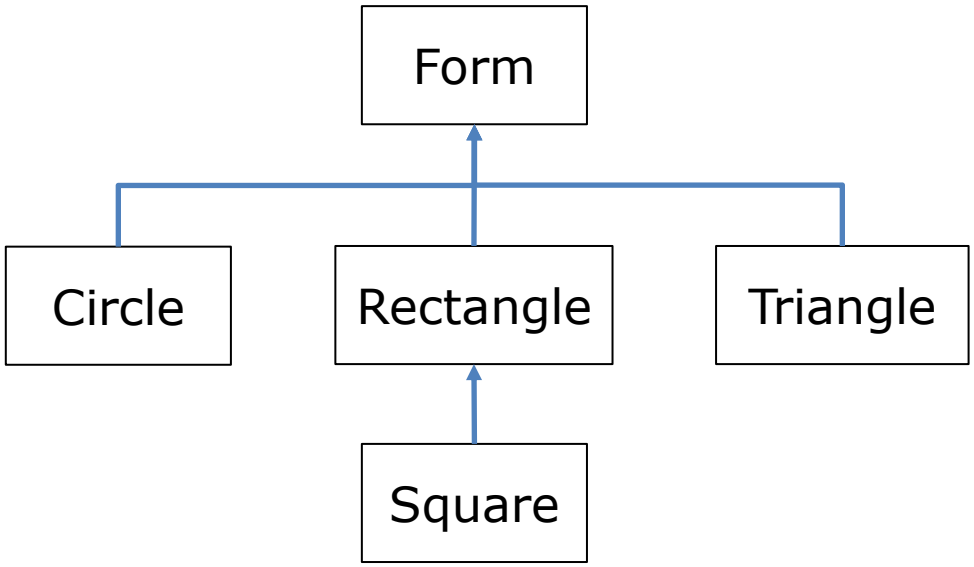
1. 继承的概念

- 继承是如何被引入的？

1. 继承的概念

❑ 继承 Heritage

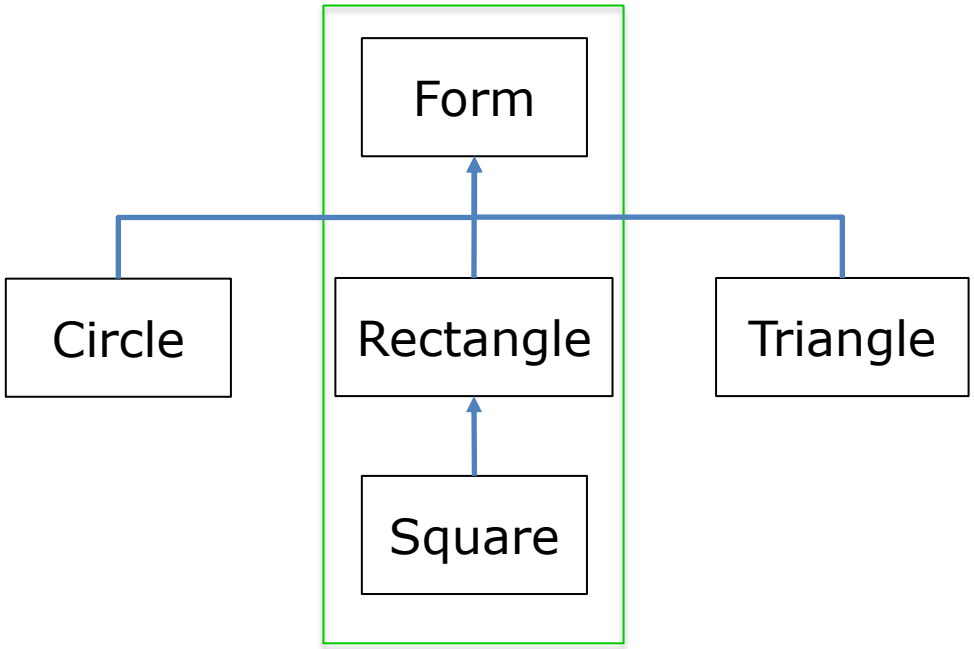
继承是一种类和类之间的关系，是一种“is a kind of”的关系。



1. 继承的概念

❑ 继承 Heritage

继承是一种类和类之间的关系，是一种“is a kind of”的关系。

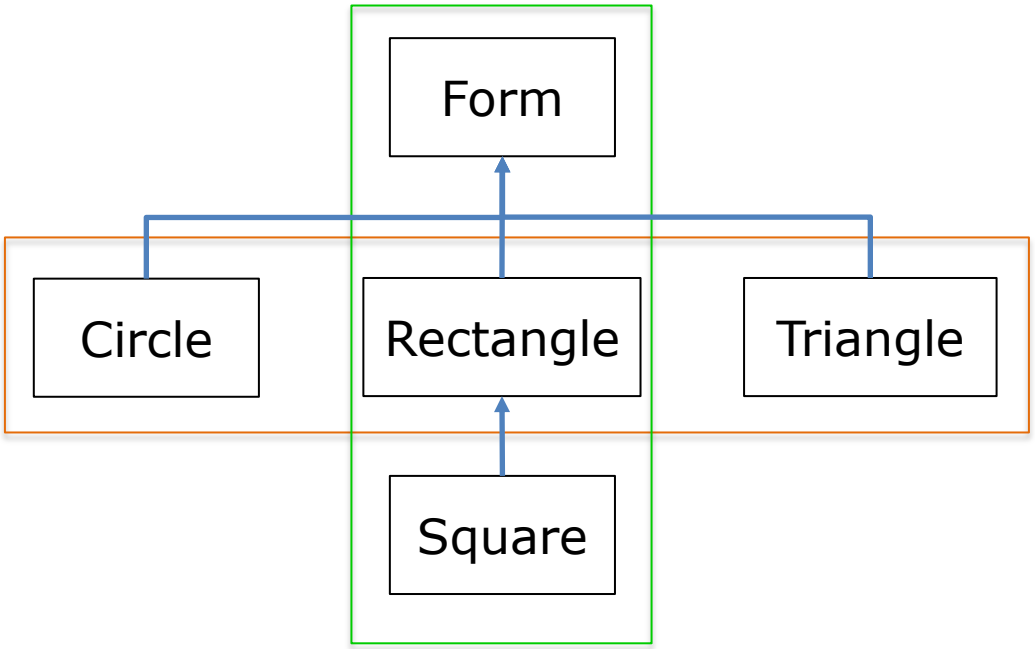


纵向：一个类共享一个或多个类中定义的属性和行为，是一种泛化/特殊化的层次关系。

1. 继承的概念

❑ 继承 Heritage

继承是一种类和类之间的关系，是一种“is a kind of”的关系。



纵向：一个类共享一个或多个类中定义的属性和行为，是一种泛化/特殊化的层次关系。

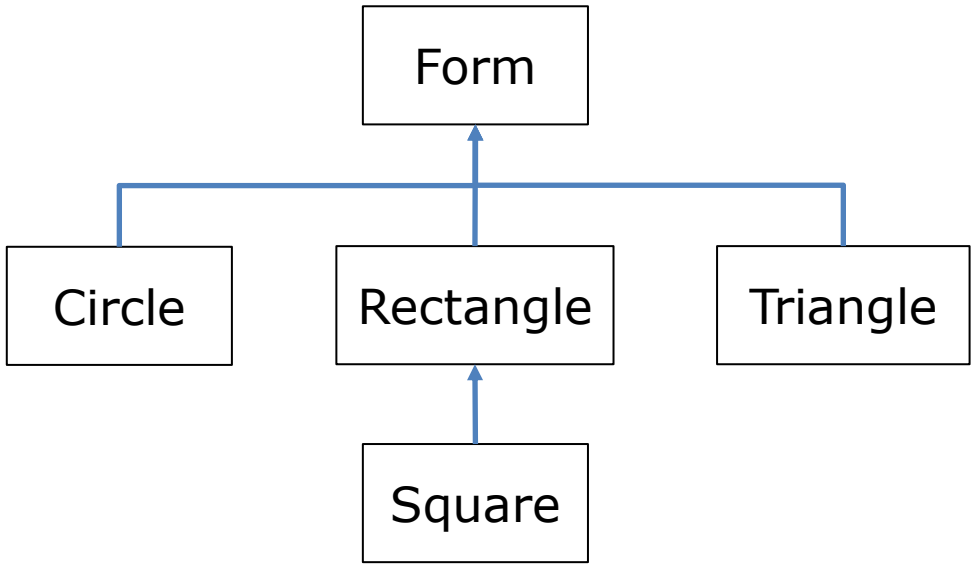
横向：同类别的多个事物进行抽象，抽取了它们的共性，放入父类中共享。

1. 继承的概念

❑ 继承的好处

继承是java面向对象编程技术的**一块基石**，因为：

- 在设计时，它允许创建分等级层次的类，思路更清晰，减少了冗余代码，易于修改；
- 定义新类时，可以以原来的类为基础，借助代码的重用，使系统变得容易扩展，提高了开发效率。
- 继承帮助我们统一了函数的调用接口，在增加新类后减少对已有系统的修改。




```
public class Canvas{
  private Form[] forms;

  void draw(){
    for(Form f: forms)
      f.draw();
  }
}
```

2. 继承的实现

❑ 语法格式

```
public class Parent{
   …
}

public class Child extends Parent {
   …
}
```

java中的继承是单继承的，一个子类只能继承一个父类，使用extends关键字。

继承是多层级的；

共同的祖先 java.lang.Object

子类继承了父类中所有非private成员方法（除了构造方法）和所有非private的成员变量。

2. 继承的实现

❑ 构造函数 Constructor 和 super

对于构造函数而言，它只能够被调用，而不能被继承。

当我们需要调用父类的构造方法时，只需使用super()即可。

```
public class Rectangle{
  private int a,b;

  public Rectangle(int a, int b){
    this.a = a;
    this.b = b;
  }

}
```

```
public class Square extends Rectangle {
  public Square(int x){
    super(x,x);
  }
}
```

2. 继承的实现

❑ 构造函数 Constructor 和 super

```
public class Rectangle{
  private int a,b;

  public Rectangle(){
    this.a = 0;
    this.b = 0;
  }

  public Rectangle(int a, int b){
    this.a = a;
    this.b = b;
  }
}
```

```
public class Square extends Rectangle {
  public Square(int x){
    super();
    …
  }
}
```

- 创建子类对象实例时，Java默认地首先调用父类的无参数的构造方法，生成父类的对象。接下来完成调用子类的构造方法，生成子类的对象。
- 使用了super显式地调用父类的某个构造方法后，那么在执行的时候就寻找该构造方法，而不再寻找父类的无参数构造方法。
- super必须作为构造方法的第一条执行语句。

2. 继承的实现

❑ 方法的重写（覆盖） override

```
public class Rectangle{
  private int a,b;

  public int getArea(char unit){
    …
  }

  public void showInfo(){
    System.out.println("This is
a rectangle(" + this.a + "," +
this.b + ").");
  }
}
```

```
public class Square extends
Rectangle {

  public int getArea(char unit){
    …
  }

  public void showInfo(){
    System.out.println("This is a
square(" + this.a + ").");
  }
}
```

- 子类中有和父类中可访问（可继承到子类）的同名&&同返回类型&&同参数表的方法，就会重写（覆盖）从父类继承来的方法。

外壳不变，核心重写！

- 可以在子类中通过super关键字调用父类被重写的方法。

2. 继承的实现

❑ 方法的重载 overload

重载方法名字相同，而参数不同。
返回类型可以相同也可以不同。

```
public class Rectangle{
  private int a,b;

  public Rectangle(){
    this.a = 0;
    this.b = 0;
  }

  public Rectangle(int a, int b){
    this.a = a;
    this.b = b;
  }
}
```

重载规则

- 被重载的方法必须改变参数列表(参数个数或类型或顺序不一样)；
- 被重载的方法可以改变返回类型；
- 被重载的方法可以改变访问修饰符；
- 方法能够在同一个类中或者在其子类中被重载。

2. 继承的实现

❑ final关键字

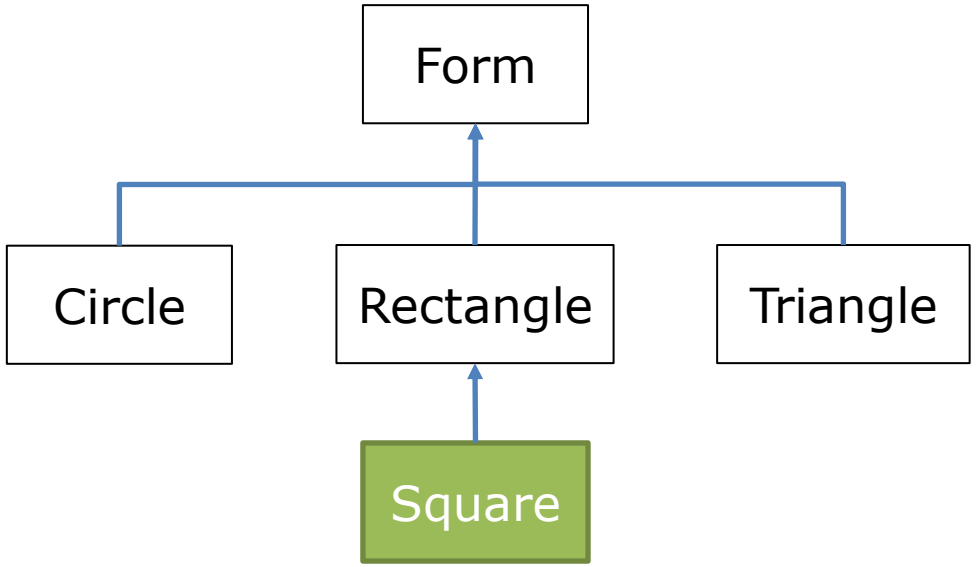
final 关键字声明类可以把类定义为不能继承的，即最终类；
或者用于修饰方法，该方法不能被子类重写：

```
public final class Square{
  private int a,b;

  public Square(){
    this.a = 0;
    this.b = 0;
  }

  public Square(a, int b){
    this.a = a;
    this.b = b;
  }

  public int getArea(){
    return this.a*this.b;
  }
}
```



2. 继承的实现

❑ 向上转型

```
class Test1 {
  …
  A a;
  a = new B();

  a.show();
  a.hello();
}
```

```
class Test2 {
  …
  A a;
  B b = new B();
  a = b;

  a.show();
  b.show();
  a.hello();
  b.hello();
}
```

```
class A {
  void show(){
    print("A");
  }
}

class B extends A {
  void show(){
    print("B");
  }

  void hello(){
    print("Hello");
  }
}
```

对象a是子类对象b的**上转型对象**

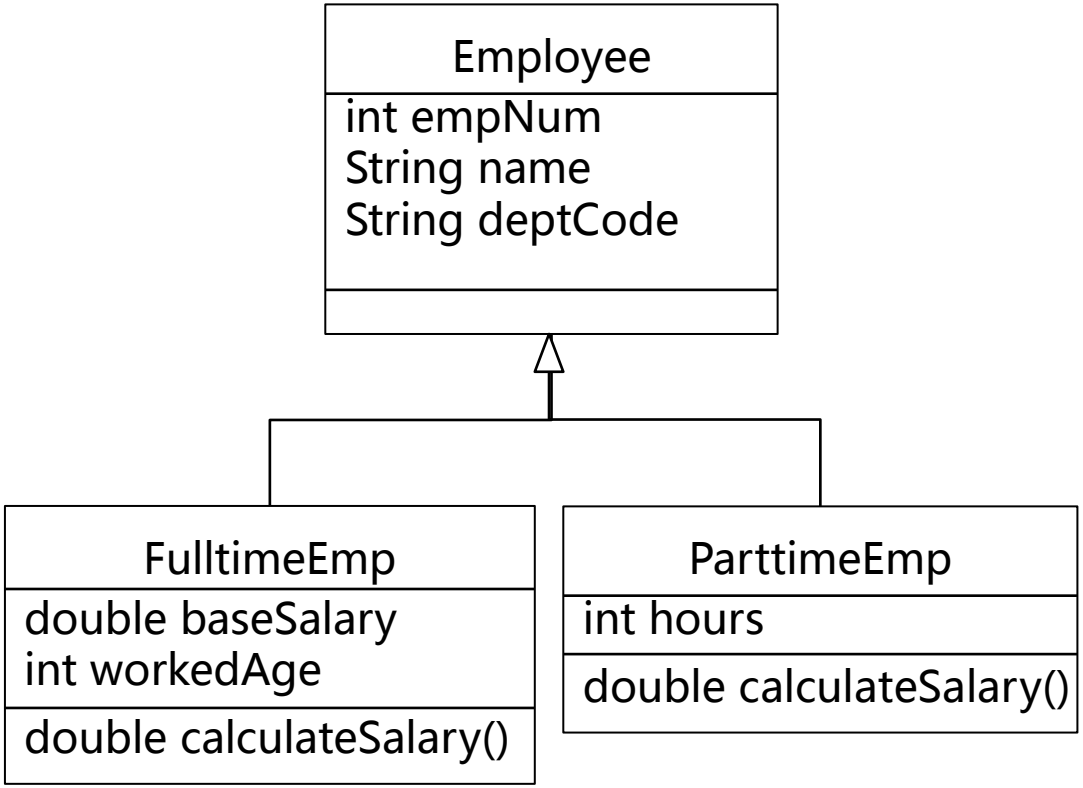
上转型对象可以操作被子类继承和重写的方法，

但不能操作子类新增的成员变量和方法。

【练习】某公司人事管理系统的问题域中有Employee（员工）类、FulltimeEmp（全职员工）类和ParttimeEmp（兼职员工）类，它们的关系如图所示：
请实现

1）输出员工的基本信息；

2）为不同类型员工计算每月工资（全职：基础工资+工龄*500；兼职：每小时50元）。



4. 章节小结

❑ 使用继承的注意事项

继承是一种强耦合关系，在一定程度上破坏了封装， 父类变化了，子类也随之变化；

继承可能造成笨拙的类结构，大量的覆盖（重写）带来不必要的复杂度；

继承很有用，但请慎用继承

何时选用继承呢？

明显存在is kind of的关系时，可选用继承

问一问自己是否需要从子类向父类进行向上转型。如果必须向上转型，则继承是必要的，但是如果不需要，则应当好好考虑自己是否需要继承。

少覆盖原则：既子类应当尽量少的覆盖父类方法，如果覆盖了父类的大多数方法，那就应当考虑是否应当有继承关系

4. 章节小结

继承的概念

继承的实现

继承的应用举例

下节预告

抽象类和方法