2、具体内容

面向对象的第二大特征就是继承性，继承性的主要特点在于：可以扩充已有类的功能。

■继承问题的引出

所谓的良好的代码指的是结构性合理、适合于维护、可重用性很高，但是如果现在只是按照之前所学习到的概念进行程序的定义，那么不可避免的要面对重复的问题，下面定义两个类：人类、学生类，如果按照传统定义，则结构如下：

Person.java类

```
class Person {
    private String name ;
    private int age ;
    public void setName(String name) {
        this.name = name ;
    }
    public void setAge(int age) {
        this.age = age ;
    }
    public String getName() {
        return this.name ;
    }
    public int getAge() {
        return this.age ;
    }
}
```

Student.java类

```
class Student {
    private String name ;
    private int age ;
    private String school ;
    public void setName(String name) {
        this.name = name ;
    }
    public void setSchool(String school) {
        this.school = school;
    }
    public void setAge(int age) {
        this.age = age ;
    }
    public String getSchool() {
        return this.school ;
    }
    public String getName() {
        return this.name ;
    }
    public int getAge() {
        return this.age ;
    }
}
```

这个时候可以发现虽然类的概念可以解决结构性的问题，但是对于之前的开发的程序代码总能够发现有一些重复的代码出现在程序之中，并且可以进一步思索以下关系：学生是一个人，人是一个更加广泛的定义范畴，而学生是一个相对狭小的定义范畴。从另一个角度来讲，学生之中应该包含人的所有特点。

如果想要进行代码的重用，那么久必须使用继承的概念来解决，所谓的继承的本质：在已有类的功能上继续进行功能的扩充。

■继承的实现

如果在Java程序之中要想实现继承关系，那么久必须依靠extends关键字来完成，此关键字的具体语法如下：

```
class 子类 extends 父类{}
```

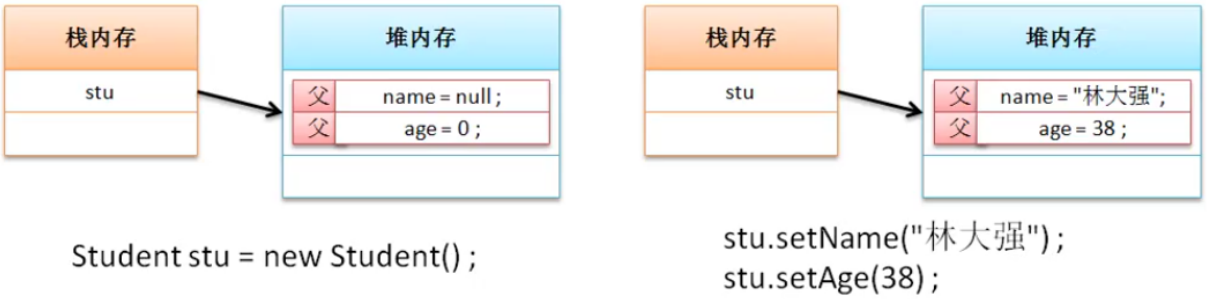
特别需要注意的是，很多情况下会把子类称为派生类，把父类称为超类（SuperClass）。

范例：观察继承的实现

```
class Person {
	private String name ;
	private int age ;
	public void setName(String name) {
		this.name = name ;
	}
	public void setAge(int age) {
		this.age = age ;
	}
	public String getName() {
		return this.name ;
	}
	public int getAge() {
		return this.age ;
	}
}
class Student extends Person {	// Student是子类
	// 在子类之中不定义任何的功能
}
public class JavaDemo {
	public static void main(String args[]) {
		Student stu = new Student() ;
		stu.setName("林大强") ;  // 父类定义
		stu.setAge(38) ; // 父类定义
		System.out.println("姓名：" + stu.getName() + "、年龄：" + stu.getAge()) ;
	}
}
```

由于此时存在有继承关系，所以此时的子类即便没有定义任何的操作，那么也可以直接通过父类继承而来的方法实现相应的功能，而这个时候的内存关系如下。

继承操作



Student stu = new Student() ;

stu.setName("林大强") ;
stu.setAge(38) ;

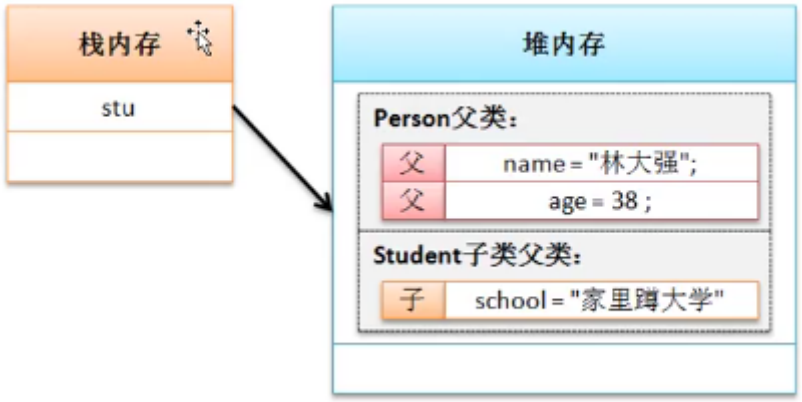
继承实现的主要目的是在于子类可以重用父类中的结构，并且也可以实现功能的扩充，那么同时强调了：子类可以定义更多的内容，并且描述的范围更小。

范例：子类扩充定义

```
class Person {
	private String name ;
	private int age ;
	public void setName(String name) {
		this.name = name ;
	}
	public void setAge(int age) {
		this.age = age ;
	}
	public String getName() {
		return this.name ;
	}
	public int getAge() {
		return this.age ;
	}
}
class Student extends Person {	// Student是子类
	private String school ; // 子类扩充的属性
	public void setSchool(String school) {
		this.school = school ;
	}
	public String getSchool() {
		return this.school ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Student stu = new Student() ;
		stu.setName("林大强") ;  // 父类定义
		stu.setAge(38) ; // 父类定义
		stu.setSchool("家里蹲大学") ;
		System.out.println("姓名：" + stu.getName() + "、年龄：" + stu.getAge() + "、学校：" + stu.getSchool()) ;
```

```
    }
}
```

如果此时继续讨论内存关系，则就会出现两个范围的属性了（Person父类范畴、Student子类范畴）



■子类对象实例化流程

现在已经成功的实现了继承关系，并且也已经发现了继承主要特点，但是一旦程序之中提供有继承逻辑，那么对于子类的实例化定义是有要求。从正常的社会逻辑来讲：没有父类一定没有子类，对于继承程序的逻辑也是一样的，在你进行子类对象实例化的时候一定要首先实例化好父类对象。

范例：观察一个程序

```
class Person {
    public Person() {
        System.out.println("【Person父类】一个新的Person父类实例化对象产生了。") ;
    }
}
class Student extends Person {      // Student是子类
    public Student() {    // 构造方法
        System.out.println("【Student子类】一个新的Student实例化对象产生了。") ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        new Student() ; // 实例化子类对象
    }
}
```

```
class Person {
 public Person() {
  System.out.println("【Person父类】一个新的Person父类实例化对象产生了。") ;
 }
}
```

```
class Student extends Person { // Student是子类
 public Student() { // 构造方法
  System.out.println("【Student子类】一个新的Student实例化对象产生了。") ;
 }
}
public class JavaDemo {
 public static void main(String args[]) {
  new Student() ; // 实例化子类对象
 }
```

程序执行结果：	【Person父类】一个新的Person父类实例化对象产生了。 【Student子类】一个新的Student实例化对象产生了。

现在即使没有进行父类对象实例化，也会由系统自动调用父类的构造方法（实例化父类对象），默认情况下的子类对象实例化流程里面会自动实例化父类对象。实际上这个时候就相当于子类的构造方法里面隐含了一个"super()"的形式

范例：修改子类定义

```
class Student extends Person {      // Student是子类
      public Student() {    // 构造方法
            super();//写与不写此语句效果一样
            System.out.println("【Student子类】一个新的Student实例化对象产生了。") ;
      }
}
```

super()表示的就是子类构造调用父类构造的语句，该语句只允许放在子类构造方法的首行。在默认情况下实例化处理，子类只会调用父类的无参构造方法，所以写与不写“super()”区别不大，但是如果说你的父类里面没有提供无参构造，这个时候就必须利用super()明确调用有参构造。

```
class Person {
      private String name ;
      private int age ;
      public Person(String name,int age) {
            this.name = name ;
            this.age = age ;
      }
}
class Student extends Person {      // Student是子类
      private String school ;
      public Student(String name,int age,String school) {      // 构造方法
            super(name,age) ; // 明确调父类构造
            this.school = school ;
      }
}
public class JavaDemo {
      public static void main(String args[]) {
            new Student("林小强",48,"北京大学") ; // 实例化子类对象
      }
}
```

结论：无论如何折腾，在实例化子类对象的同时一定会实例化父类对象，目的是为了所有的属性可以进行空间的分配。

```
class Person {
	private String name ;
	private int age ;
	public Person(String name,int age) {
		this.name = name ;
		this.age = age ;
	}
}
class Student extends Person {	// Student是子类
	private String school ;
	public Student(String name,int age,String school) {	// 构造方法
		this(name,age) ;	// 明确调父类构造
		this.school = school ;
	}
	public Student(String name,int age) {
		this(name,age,"家里蹲大学") ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		new Student("林小强",48,"北京大学") ; // 实例化子类对象
	}
}
```

super与this都可以调用构造方法，super是由子类调用父类的构造，而this是调用本类构造，并且都一定要放在构造方法的首行，所以两个语句不允许使用出现。

■继承的相关限制

现在已经清楚了整个的继承逻辑，那么下面对于继承实际上还有一些要求。

1、Java之中不允许多重继承，只允许多层继承。

• 在实际的生活之中一个人只会有一个父亲，那么在程序的路基之中也是奉行与此标准的。

错误的继承	**正确的继承**
class A {} class B {} class C extends A,B { } // 多重继承	class A {} class B extends B{} class C extends C { } // 多层继承

继承的主要目的是扩展已有类的功能，但是多重继承目的是希望可以同时继承多个类中的方法，而面对于多重继承的要求应该是将范围限定在同一类之中。如果说现在使用了多层继承，这个时候对于C类也同样可以继承多个父类的操作。但是多层继承也应该有一个限度，别整继承关系太长，对于继承关系而言，如果时写的代码，理论上层次不应该超过三层。

2、在进行继承关系定义的时候，实际上子类可以继承父类中的所有的操作结构。但是对于私有操作属于隐式继承，而所有的非私有操作属于显示继承。

```
class Person {
	private String name ;
	public void setName(String name) {
		this.name = name ;
	}
	public String getName() {
		return this.name ;
	}
}
class Student extends Person {
	public Student(String name) {
		setName(name) ;	// 设置name属性内容
	}
	public void fun() {
		// System.out.println(name) ; // 直接访问不可能，因为私有的
		System.out.println(getName()) ;	// 间接访问
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Student stu = new Student("林中强") ;
		stu.fun() ;
	}
}
```

继承一旦发生了，所有的操作就都可以被子类使用了，并且在程序设计里面并没有考虑到现实生活中所谓的“败家子”的概念，子类至少会维持父类的现有功能。