2、具体内容

类继承的主要作用在于可以扩充已有类的功能，但是对于之前的继承操作而言会发现，子类可以由自己的选择任意来决定是否要覆写某一个方法，这个时候父类无法对子类做出强制性约定（强制性必须覆写某些方法），这种情况下往往不会采用类的继承（在实际的开发之中很少会继承一个已经完善的类，可以直接使用的类）而是必须要继承抽象类，在以后进行父类（SuperClass）设计的时候优先考虑的一定是抽象类。

■抽象类的基本定义

抽象类的主要作用在于对子类中覆写方法进行约定，在抽象类里面可以去定义一些抽象方法以实现这样的约定，抽象方法指的是使用了abstract关键字定义的并且没有提供方法体的方法，而抽象方法所在的类必须为抽象类，抽象类必须使用abstract关键字来进行定义（在普通类的基础上追加抽象方法就是抽象类）。

范例：定义一个抽象类

```
abstract class Message {        // 定义抽象类
        private String type ; // 消息类型
        public abstract String getConnectInfo() ;   // 抽象方法
        public void setType(String type) {  // 普通方法
                this.type = type ;
        }
        public String getType() { // 普通方法
                return this.type ;
        }
}
public class JavaDemo {
        public static void main(String args[]) {
                Message msg = new Message() ;
                System.out.println(msg.getConnectInfo()) ;
        }
}
```

当一个抽象类定义完成之后（切记：“抽象类不是完整的类”），如果想要去使用抽象类则必须按照如下原则进行：

- 抽象类必须提供有子类，子类使用extends继承一个抽象类；
- 抽象类的子类（不是抽象类）一定要覆写抽象类中的全部抽象方法；
- 抽象类的对象实例化可以利用对象多态性通过子类向上转型的方法完成；

范例：使用抽象类

```
abstract class Message { // 定义抽象类
	private String type ; // 消息类型
	public abstract String getConnectInfo() ;	// 抽象方法
	public void setType(String type) {  // 普通方法
		this.type = type ;
	}
	public String getType() { // 普通方法
		return this.type ;
	}
}
class DatabaseMessage extends Message {	// 类的继承关系
	public String getConnectInfo() {	// 方法覆写
		return "Oracle数据库连接信息。" ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Message msg = new DatabaseMessage() ;
		msg.setType("客户消息") ;
		System.out.println(msg.getConnectInfo()) ;
		System.out.println(msg.getType()) ;
	}
}
```

从整体上来讲，抽象类只是比普通类增加了抽象方法以及对子类的强制性的覆写要求而已，其它的使用过程和传统的类继承是完全相同的。

对于抽象类使用的几点意见：

- 抽象类使用很大程度上有一个核心的问题：**抽象类自己无法直接实例化**；
- 抽象类之中主要的目的是进行过渡操作使用，所以当你要使用抽象类进行开发的时候，往往都是在你设计中需要解决类继承问题时所带来的的代码重复处理；

■抽象类的相关说明

抽象类是一个重要的面向对象设计的结构，对于抽象类使用的时候需要注意以下几点问题：

1、在定义抽象类的时候绝对不能够使用final关键字来进行定义，因为抽象类必须有子类，而final定义的类是不能够有子类；

2、抽象类时作为一个普通类的加强版出现的（抽象类的组成就是在普通类的基础上扩展而来的，只是追加了抽象方法）既然是在普通类基础上扩展的，那么普通类之中就可以定义属性和方法，那么这些属性一定是要求进行内存空间开辟的，所以抽象类一定可以提供有构造方法，并且子类也一定会按照子类对象的实例化原则进行父类构造调用。

```
abstract class Message { // 定义抽象类
	private String type ; // 消息类型
	public Message(String type) { // 类中没有提供有无参构造
		this.type = type ;
	}
	public abstract String getConnectInfo() ;	// 抽象方法
	public void setType(String type) {  // 普通方法
		this.type = type ;
	}
	public String getType() { // 普通方法
		return this.type ;
	}
}
class DatabaseMessage extends Message {   // 类的继承关系
	public DatabaseMessage(String str) {
		super(str) ;
	}
	public String getConnectInfo() {    // 方法覆写
		return "Oracle数据库连接信息。" ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Message msg = new DatabaseMessage("客户消息") ;
		System.out.println(msg.getConnectInfo()) ;
		System.out.println(msg.getType()) ;
	}
}
```

3、抽象类之中允许没有抽象方法，但是即便没有抽象方法，也无法直接使用关键字new直接实例化抽象对象。

```
abstract class Message { // 定义抽象类
}
class DatabaseMessage extends Message {   // 类的继承关系
}
public class JavaDemo {
	public static void main(String args[]) {
		Message msg = new DatabaseMessage() ;
	}
}
```

即便抽象类没有实例化对象，那么也无法直接使用关键字new获取抽象类的对象，必须依靠子类对象完成。

4、抽象类可以提供有staitc方法，并且该方法不会受到抽象类对象的局限。

```
abstract class Message { // 定义抽象类
```

```
    public abstract String getInfo() ;    // 抽象方法
    public static Message getInstance() {
        return new DatabaseMessage() ;
    }
}
class DatabaseMessage extends Message {   // 类的继承关系
    public String getInfo() {
        return "数据库连接信息。" ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Message msg = Message.getInstance() ;
        System.out.println(msg.getInfo()) ;
    }
}
```

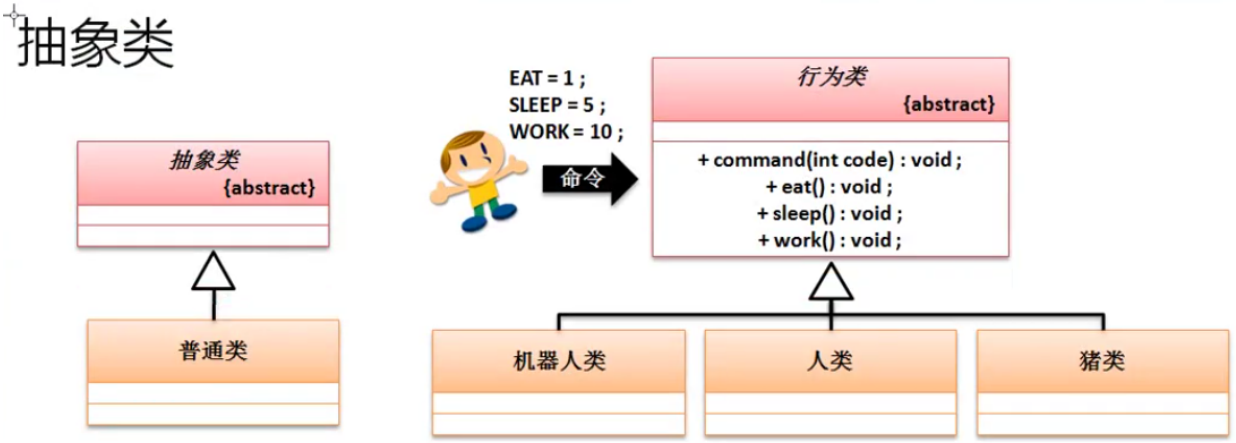
static方法永远不受到实例化对象或结构的限制，永远可以直接通过类名称进行调用。

■抽象类的应用

抽象类的本质上就属于一个加强型的类，那么对于类已经清楚了，可以描述一切的有关的现实事物，但是通过分析可以发现，抽象类的设计应该是比类更高一层的定义。那么下面来研究一下抽象类的实际应用。

假如说现在要描述有三类的事物：

- 机器人：不休息，只知道补充能量和工作；
- 人类：需要休息、需要吃饭以及努力的工作；
- 猪：需要休息、不需要工作只需要吃饭；



范例：实现代码

```
abstract class Action {
    public static final int EAT = 1 ;
    public static final int SLEEP = 5 ;
    public static final int WORK = 10 ;
    public void command(int code) {
        switch(code) {
```

```
                case EAT : {
                        this.eat() ;
                        break ;
                }
                case SLEEP : {
                        this.sleep() ;
                        break ;
                }
                case WORK : {
                        this.work() ;
                        break ;
                }
                case EAT + SLEEP + WORK : {
                        this.eat() ;
                        this.sleep() ;
                        this.work() ;
                        break ;
                }
        }
    }
    public abstract void eat() ;
    public abstract void sleep() ;
    public abstract void work() ;
}
class Robot extends Action {
    public void eat() {
        System.out.println("机器人需要接通电源充电。") ;
    }
    public void sleep() {}
    public void work() {
        System.out.println("机器人按照固定的套路进行工作。") ;
    }
}
class Person extends Action {
    public void eat() {
        System.out.println("饿的时候安静的坐下吃饭。") ;
    }
    public void sleep() {
        System.out.println("安静的躺下，慢慢的睡着，而后做着美丽的春梦。") ;
    }
    public void work() {
        System.out.println("人类是高级脑类动物，所有要有想法的工作。") ;
    }
}
class Pig extends Action {
    public void eat() {
        System.out.println("吃食槽中的人类的剩饭。") ;
    }
    public void sleep() {
        System.out.println("倒地就睡。") ;
    }
    public void work() {}
```

```
}
public class JavaDemo {
	public static void main(String args[]) {
		Action robotAction = new Robot() ;
		Action personAction = new Person() ;
		Action pigAction = new Pig() ;
		System.out.println("------------ 机器人行为 -------------") ;
		robotAction.command(Action.SLEEP) ;
		robotAction.command(Action.WORK) ;
		System.out.println("------------ 人类的行为 -------------") ;
		personAction.command(Action.SLEEP + Action.EAT + Action.WORK) ;
		System.out.println("------------ 猪类的行为 -------------") ;
		pigAction.work() ;
		pigAction.eat() ;
	}
}
```

现在的程序以及完整的实现了一个行为的抽象处理，但是也需要作出一点点思考。现在定义的Action父类主要的目的：对所有行为规范进行统一处理。

姓名：	林小强	**生日：**	2009-10-10
籍贯：	广州	**民族：**	汉

抽象类最大的好处一是对子类方法的统一管理，二是可以自身提供一些普通方法并且这些普通方法可以调用抽象方法（这些抽象方法必须在有子类提供实现的时候才会生效）。

```
abstract class Action {
	public static final int EAT = 1 ;
	public static final int SLEEP = 5 ;
	public static final int WORK = 10 ;
	public void command(int code) {
		switch(code) {
			case EAT : {
				this.eat() ;
				break ;
			}
			case SLEEP : {
				this.sleep() ;
				break ;
			}
			case WORK : {
				this.work() ;
				break ;
			}
			case EAT + SLEEP + WORK : {
				this.eat() ;
				this.sleep() ;
				this.work() ;
				break ;
			}
		}
```

```
    }
    public abstract void eat() ;
    public abstract void sleep() ;
    public abstract void work() ;
}
class Robot extends Action {
    public void eat() {
        System.out.println("机器人需要接通电源充电。") ;
    }
    public void add(){ // 这个方法父类不知道

    }
    public void sleep() {}
    public void work() {
        System.out.println("机器人按照固定的套路进行工作。") ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Action robotAction = new Robot() ;
        System.out.println("------------ 机器人行为 -------------") ;
        robotAction.command(Action.SLEEP) ;
        robotAction.command(Action.WORK) ;
    }
}
```