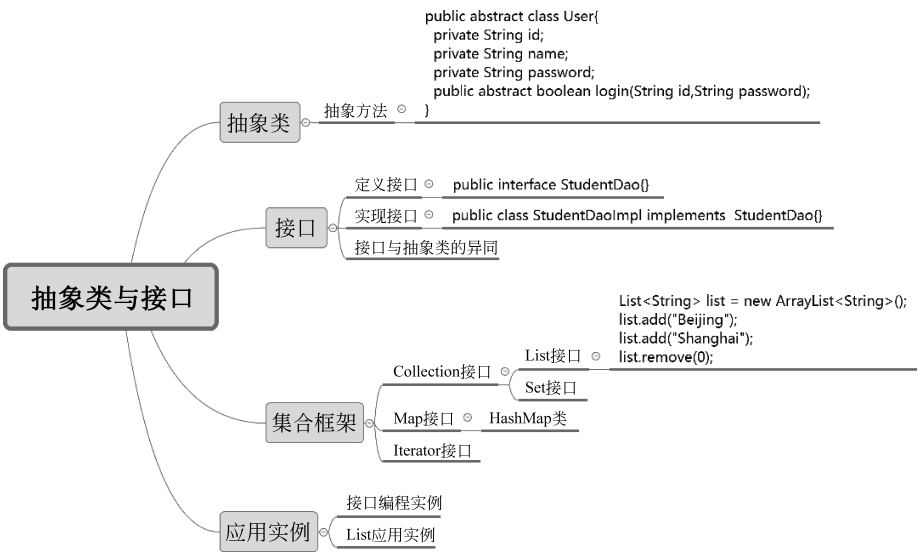
第5章

抽象类与接口

内容导览



学习目标

- 能够根据项目设计需求设计抽象类
- 能够学会定义接口，应用类实现接口
- 理解接口与抽象类的区别，能够使用接口和抽象类解决工程问题
- 掌握集合及集合接口的使用方法，学会使用集合实现对象的增删改查操作

5.1 引　　例

例 5.1　设计一款电子游戏，游戏中有不同类型怪物，如英雄、火龙、天使、僵尸等。这些角色既有共同的行为（如战斗），也有不同行为（如英雄具有战斗、游泳、飞行等功能）。请试着设计游戏，让系统足够灵活，易于扩展，如游戏升级时能方便地增加新的角色；同时有利于编程，能够让多个人一起协同开发。

分析：通过第 4 章中继承的学习，我们可以将战斗的行为抽象为父类的方法，通过继承使子类都具有战斗的行为。但考虑到角色可能具有多个功能，我们可以利用接口的多继承机制，将其他功能定义为接口，在接口中定义抽象方法。在角色设计时，游泳、飞行设计为单独的接口，每个角色根据设计要求通过多继承实现，如设计一个英雄的角色，具有 3 个功能。

```
//文件名为 Jpro5_1.java
1  interface CanSwim{
2      void swim();
3  }
4  interface CanFly{
5      void fly();
6  }
7  class ActionCharacter{
8      public void fight(){
9          System.out.println("can fight!");
10      }
11 }
12 class Hero extends ActionCharacter implements CanSwim,CanFly{
13     public void swim(){
14        System.out.println("can swim!");
15     }
16     public void fly(){
17         System.out.println("can fly!");
18     }
19 }
20 public class Jpro5_1{
21     public static void u(CanSwim x){
22         x.swim();
23     }
24     public static void v(CanFly x){
25         x.fly();
26     }
27     public static void w(ActionCharacter x){
28         x.fight();
29     }
30     public static void main(String args[]){
31         Hero h=new Hero();
32         u(h);
33         v(h);
34         w(h);
35     }
36 }
```

程序运行结果为：

```
can swim!
can fly!
can fight!
```

程序分析：在上述 Java 源程序 Jpro5_1.java 中定义了两个接口和一个类。第 1 行定义接口 CanSwim，其定义了抽象方法 swim()；第 4 行定义接口 CanFly，其定义了抽象方法 fly()；第 7 行定义类 ActionCharacter，第 12 行定义 Hero 类从 ActionCharacter 类扩展，利用 implements 将 CanSwim 和 CanFly 接口合并。在 Jpro5_1 类中有 3 个方法：u()、v()和 w()，它们接受不同接口/类参数。一旦 Hero 对象生成，可被传送到这 3 个方法中的任意一个。

在解决这个问题时，我们提到了抽象方法、接口等，下面将一一介绍。

5.2

5.2 抽 象 类

在面向对象程序设计中，父类抽取并定义了其子类共同拥有的属性和方法。在这些属性和方法中，有些是已经明确并可以在父类中实现的，有些则与具体的子类有关且无法在父类中实现。对于无法确定的方法，我们可以将其在父类中定义为抽象方法，相应地该父类也被定义为抽象类。例如，在学生成绩管理系统中，用户有三个属性：id(可用作登录用户名)、password(密码)、name(姓名)；有一个登录方法：boolean login(String id, String password)。系统中设有管理员类、教师类和学生类三类用户，可以继承用户类的 3 个属性并重写登录方法。此时，用户类的登录方法可以定义为抽象方法，该用户类也应该定义为抽象类。

抽象类的定义与一般类一样，都有数据和方法，定义格式与一般类也非常类似，只是在定义类的 class 前增加一个关键字 abstract 来表示定义一个抽象类，即用 abstract 修饰的类称为抽象类。

抽象方法是抽象类中的一种特殊方法，用 abstract 关键字来修饰。在抽象类中，抽象方法仅有方法头，而没有方法体，它的方法体放在子类中，因此抽象方法必须在子类中重写，否则没有意义。含有抽象方法的类一定是抽象类，抽象类中除了抽象方法，还可以包括其他普通的成员方法。

抽象类和抽象方法的声明格式如下：

```
abstract class 类名{
    成员变量;
    成员方法(){方法体}
    abstract 成员方法();
}
```

例如，定义一个抽象类 User。

```
public abstract class User{
  private String id;
  private String name;
  private String password;
  public abstract boolean login(String id,String password);
}
```

抽象类不能创建对象，它只能作为其他类的父类，它的存在仅是为了继承而用。User 类是抽象类，不能用 new 创建它的实例，但它可以被继承。抽象方法 login()只有方法声明部分和“空”实现，它的具体实现由其子类完成。例如，第 7 章介绍的 InputStream 类和 OutputStream 类都属于抽象类，均包含抽象方法和其他方法。

注意：如果一个类没有显式地被 abstract 修饰，但是类体中包含抽象方法，则该类也为抽象类。

例 5.2 下面定义了一个抽象类，然后通过继承实现该抽象类。

```
//文件名为 Jpro5_2.java
1   abstract class Animal{                  //抽象类
2       String name;
3       public abstract void go();          //抽象方法
4   }
5   class Cat extends Animal{
6       public Cat(String name){
7           this.name=name;
8       }
9       public void go(){                   //方法的覆盖
10          System.out.println(name+" can run.");
11      }
12  }
13  class Bird extends Animal{
14      Bird(String name){
15          this.name=name;
16      }
17      public void go(){                   //方法的覆盖
18          System.out.println(name+" can fly.");
19      }
20  }
21  public class Jpro5_2{
22      public static void main(String args[]){
23          Cat c=new Cat("Cady");
24          Bird b=new Bird("Bird");
25          c.go();
26          b.go();
27      }
28  }
```

程序分析：程序首先定义了一个抽象类 Animal，该抽象类包含一个成员变量和一个抽象方法。随后定义了两个类 Cat 和 Bird，它们都继承了抽象类 Animal。在主方法 main()中定义了一个 Cat 对象和一个 Bird 对象，并且通过对象调用相应的 go()方法。

【练习 5.1】

1. [单选题]下列关于抽象类的描述，错误的是(　　)。

A. 抽象类中有抽象方法　　B. 用 abstract 修饰的类是抽象类

C. 抽象方法没有方法体　　D. 抽象类可以用来实例化对象

2. [单选题]下列关于抽象类定义，正确的是(　　)。

A.
```
abstract AbstractClass{
    abstract void method();
    }
```

B.
```
class abstract AbstractClass{
    abstract void method();
    }
```

C.
```
abstract class AbstractClass{
    abstract void method();
    }
```

D.
```
abstract class AbstractClass{
    abstract void method(){
    System.out.println("method");
      }
    }
```

3. [程序填空]请在程序的每条横线处填写一个语句，使程序能实现功能。

```
__①__class Vehicle{
  abstract  void go();                          //抽象方法
}
class Car__②__ Vehicle{                         //继承抽象类
    public void go(){                           //方法的重写
        System.out.println("小汽车启动");
    }
}
public class Main{
    public static void main(String[] args){
        Car c=new Car();                        //创建一个 Car 对象
        c.go();
    }
}
```

4. [单选题]下列程序的运行结果是(　　)。

```
abstract class MineBase{
    abstract void amethod();
    static int i;
}

public class Mine extends MineBase{
    public static void main(String argv[]){
        int[] ar=new int[5];
        for (i=0; i<ar.length; i++)
            System.out.println(ar[i]);
    }
}
```

A. 打印 5 个 0　　B. 编译出错，数组 ar[]必须初始化

C. 编译出错，Mine 应声明为 abstract　　D. 出现 IndexOutOfBounds 的异常

5.3 接　　口

在 Java 面向对象程序设计中，有时需要定义一些“标准规范”并让相关类共同遵守，为此需要用到 Java 中的接口类型。实际上，接口是一组抽象方法的集合，这些方法没有具体的实现形式。如果一个类实现了某个接口，即继承并实现了该接口的所有抽象方法，那就表明该类遵守了该接口定义的“标准规范”。

Java 接口是另一种引用类型，也是面向对象的一个重要机制。接口又称界面，引入它的目的是为了克服 Java 单继承机制带来的缺陷，实现类的多继承功能。

5.3.1 定义接口

Java 接口在语法上类似于类的一种结构，是一种特殊的类，只定义了类中方法的原型，而没有直接定义方法的内容。接口只定义常量和抽象方法，没有变量和方法的实现。

接口的定义包括接口声明和接口体两部分，格式如下：

```
[public|abstract] interface 接口名 [extends 接口列表]{
    常量声明;
    方法声明;
}
```

接口的修饰符可以是 public 或 abstract，其中 public 或 abstract 可以缺省。public 的含义与类修饰符是一致的。但是缺省 public 或 abstract 修饰符时，定义的接口只能被同一个包中的其他类和接口使用。

注意：一个 Java 源文件中最多只能有一个 public 类或接口。当存在 public 类或接口时，Java 源文件名必须与这个类或接口同名。

如同 class 是定义类的关键字，interface 是定义接口的关键字。其后的接口名应符合 Java 对标识符的规定。

接口中所有变量的修饰符只能是 public、final 及 static，所以在定义时可以不用显式地使用修饰符。也正是由于这一点，因此在接口中定义的属性都是常量，在定义时必须给定初值。接口可以用来实现不同类之间的常量共享。

接口中定义的方法只有方法头而不能有方法体，public、abstract 缺省也有效。与抽象类一样，接口不需要构造方法。

例 5.3 组装电脑：通过 PCI 接口将主板与显卡和声卡连接。

分析：很多教材在介绍接口时使用组装计算机作为案例，本章也选择大家熟悉的组装计算机作为引例。实际上，无论是显卡还是声卡，都是插在 PCI 槽上的。所以，首先定义一个 PCI 接口，然后定义显卡类和声卡类实现该接口，从而使它们的对象能直接传递给 PCI 接口的对象，在参数传递过程中实现接口回调。

源代码如下：

```
//文件名为 Jpro5_3.java
1  interface PCI{
2      void setName(String s);
3      void run();
4  }
5  class VideoCard implements PCI{
6     String name="微星";
7      public void setName(String s){
8          name=s;
9      }
10     public void run(){
11         System.out.println(name+"显卡已开始工作!");
12     }
13 }
14 class SoundCard implements PCI{
15     String name="AC";
16     public void setName(String s){
17         name=s;
18     }
19     public void run(){
20         System.out.println(name+"声卡已开始工作!");
21     }
22 }
23 class Mainboard{
24     public void interfacePCI(PCI p){
25         p.run();
26     }
27     public void run(){
```

```
28          System.out.println("主板已开始工作!");
29      }
30  }
31  public class Jpro5_3{
32      public static void main(String[] args){
33          Mainboard mb=new Mainboard();
34          VideoCard vc=new VideoCard();
35          vc.setName("HuaWei");
36          SoundCard sc=new SoundCard();
37          mb.interfacePCI(vc);
38          mb.interfacePCI(sc);
39          mb.run();
40      }
41  }
```

程序分析：程序的第 1～4 行定义了一个接口 PCI，然后定义了 VideoCard 和 SoundCard 两个类，且实现了前面定义的接口。第 23～30 行定义了一个类 Mainboard。最后定义了一个主类 Jpro5_3，在 main()方法中模拟电脑的组装。

下面是在学生信息管理系统中使用接口。

```
1   public interface StudentDao{
2     //将学生对象添加到数据表 student 中
3     int add(Student s);
4     //根据学号删除表中对应的记录
5     int delete(String studentId);
6     //用 Student 对象更新表 student 中对应的记录
7     int update(Student s);
8     //查询 student 表中的所有学生
9     List<Student>searchAll();
10    //根据学号查询学生
11    Student findStudentById(String studentId);
12  }
```

上面的程序段定义了一个名为 StudentDao 的接口，其中有五个抽象方法：add()、delete()、update()、searchAll()和 findStudentById()，分别用于学生信息的添加、删除、修改、查找全部学生、按学号查找学生。

与类一样，接口也具有继承性。定义接口时可以使用 extends 关键字定义该新接口是某个已经存在的父接口的派生接口，它将继承父接口的所有属性和方法。与类的单继承不同，一个接口可以有多个父接口，接口名称之间用“,”分隔。新的接口将继承所有父接口的属性和方法。

5.3.2 接口实现

5.3.2

接口中只是声明了提供的功能和服务，而功能和服务的具体实现需要在实现接口的

类中定义。在类中实现接口的格式如下：

```
[类修饰符] class 类名 [extends 父类名] [implements 接口名列表]
```

其中，接口名列表包括多个接口名称，各接口间用逗号分隔。implements 是实现接口的关键字。实现接口的类如果不是抽象类，就必须实现接口中定义的所有方法并给出具体的实现代码，当然还可以使用接口中定义的任何常量。

例 5.4 接口的实现示例。

```
//文件名为 Jpro5_4.java
1  interface Shape{                    //定义接口 Shape
2     double PI=3.14;
3     void print();
4  }
5  class Circle implements Shape{  //实现接口 Shape
6      public void print(){
7          System.out.println("我实现了接口"+PI);
8      }
9  }
10 public class Jpro5_4{
11     public static void main(String args[]){
12         Circle circle=new Circle();
13         circle.print();
14     }
15 }
```

请读者自行运行结果。

程序分析：程序中定义了一个接口 Shape，其包含一个常量 PI 和方法 print()，然后定义了一个类 Circle 实现接口 Shape。注意在实现方法 print()时，其修饰符必须为 public。在 main()方法中创建了一个 Circle 类对象 circle，然后通过 circle 调用方法 print()。

在学生成绩管理系统中，接口 StudentDao 将由 StudentDaoImpl 来实现，其实现语句为

```
public class StudentDaoImpl implements StudentDao
```

具体实现方法详见第 10 章。

实现接口时，需要注意以下问题：

- 如果实现接口的类不是 abstract 修饰的抽象类，那么在类的定义部分必须实现接口中定义的所有方法并给出具体的实现代码，这是因为非抽象类中不可以存在抽象方法。
- 如果实现接口的类是 abstract 修饰的抽象类，那么它可以不实现该接口的所有抽象方法。
- 在实现接口方法时，必须将方法声明为公共方法，而且还要求方法的参数列表、名称和返回值与接口中定义的完全一致。

- 如果在实现接口的类中所实现的方法与抽象方法有相同的方法名称和不同的参数列表，则只是重载一个新的方法，并没有实现接口中的抽象方法。
- 如果接口的抽象方法的访问修饰符规定为 public，则类在实现这些抽象方法时，必须显式地使用 public 修饰符，否则系统会提示出错警告。
- 如果接口中的方法的返回类型不是 void，则在类中实现该方法时，方法体中至少要有一条 return 语句。

5.3.3 抽象类与接口的区别


5.3.3

Java 中的抽象类和接口有很多相似之处，都含有抽象方法，抽象方法的实现都放在其他类中实现。但两者却是两种完全不同的类型，它们之间的主要区别如表 5-1 所示。

表 5-1 抽象类与接口的区别

比较项目	抽 象 类	接 口
方法及实现	包含未实现的抽象方法，也可以包含已给出具体实现的方法	接口是完全抽象的，所有方法均不需要给出具体的实现形式
继承与实现	子类通过 extends 关键字来继承抽象类。若子类不是抽象类，其需要实现抽象类中所有的抽象方法	子类使用 implements 关键字实现接口，且需要实现接口中声明的所有方法
构造器	抽象类可以有构造器	接口不能有构造器
与普通 Java 类的区别	除了不能被实例化之外，关键字和普通 Java 类没有任何区别	接口是完全不同的类型
访问修饰符	抽象方法可以由 public、protected 和 default 进行修饰	接口中方法的默认修饰符是 public，通常不使用其他修饰符
多继承	一个 Java 类只能继承一个抽象类	一个 Java 类能实现多个接口
扩展新方法	在抽象类中添加新方法时，可以给出该方法的默认实现，故不需要修改其他代码	若往接口中添加新方法，必须改变实现该接口的其他所有类

在一般的应用程序设计中，往往将抽象类和接口结合起来应用，这样代码结构更加清晰，容易扩展。采用何种技术要根据应用场景来决定，通常可以遵循的规则如下：

- 定义接口。接口是系统的核心，定义了要完成的功能，包含主要的方法声明，因此系统最顶层采用接口。
- 定义抽象类。如果某些类实现的方法有相似性，则可以抽象出一个抽象类包含方法声明；如果顶层接口有共同部分要实现，则定义抽象类实现接口。
- 由抽象类的具体类各自实现个性化的方法。

Java API 中的集合框架很好地体现了这种设计原则，我们将在 5.4 节中详细讲解它。

【练习 5.2】

1. [单选题]下列关于接口的描述，错误的是(　　)。