# Chp8 接口

## Key Point

- 接口的基本语法
- 接口的作用
- 解耦合

## 练习

1. 代码改错：

```
interface IA{
    void m1();
    int a = 100;
}
class MyClass implements IA{
    void m1(){}
}
public class TestInterface{
    public static void main(String args[]){
        IA ia = new MyClass();
        ia.m1();
        System.out.println(IA.a);
    }
}
```

2. 代码填空：

```
interface IA{
    void m1();
    void m2();
}

_____ class MyClassA implements IA{
    public void m1(){}
}

class MyClassB extends MyClassA{
    _________________ {}
}
```

3. 有如下代码：

```
interface IA{
   void ma();
}

interface IB extends IA{
   void mb();
}

interface IC{
   void mc();
}

interface ID extends IB, IC{
   void md();
}
```

1) 如果有一个类 ClassE 实现 ID 接口，如果不希望 ClassE 是抽象的，则需要实现哪些方法？

2) 把下面的代码补充完整

```
public class TestClassE{
   public static void main(String args[]){
      IC ic = new ClassE();
      //调用 ma 方法
      ________________
      //调用 mb 方法
      ________________
      //调用 mc 方法
      ________________
      //调用 md 方法
      ________________
   }
}
```

3) 写出下面代码的输出结果

```
   public class TestClassE{
      public static void main(String args[]){
         IC ic = new ClassE();
         System.out.println(ic instanceof IA);
         System.out.println(ic instanceof IB);
         System.out.println(ic instanceof IC);
         System.out.println(ic instanceof ID);
         System.out.println(ic instanceof ClassE);
      }
```

```
    }
```

4. 把上一章的 Shape 类由抽象类改为接口。

5. *有如下代码：

```
interface IA{
    void ma();
}

interface IB{
    void mb();
}

class MySuper implements IA{
    public void ma(){}
}

class MySub extends MySuper implements IB{
    public void mb(){}
}

public class TestMain{
    public static void main(String args[]){
        MySuper ms = new MySub();
        System.out.println(ms instanceof IA);
        System.out.println(ms instanceof IB);
        System.out.println(ms instanceof MySuper);
        System.out.println(ms instanceof MySub);
    }
}
```

问：该程序输出结果是什么？

6. *关于接口和抽象类，下列说法正确的是：

   A. 抽象类可以有构造方法，接口没有构造方法

   B. 抽象类可以有属性，接口没有属性

   C. 抽象类可以有非抽象方法，接口中都是抽象方法

   D. 抽象类和接口都不能创建对象

   E. 一个类最多可以继承一个抽象类，但是可以实现多个接口

7. *写出下面代码的输出结果：

```
interface IA{
    void m1();
}

class IAImpl1 implements IA{
```

```
    public void m1(){
        System.out.println(“impl1”);
    }
}

class IAImpl2 implements IA{
    public void m1(){
        System.out.println(“impl2”);
    }
}

class MyClass{
    private IA ia;
    public MyClass(IA ia){
        this.ia = ia;
    }
    public void setIa(IA ia){
        this.ia = ia;
    }
    public void myMethod(){
        ia.m1();
    }
}

public class TestMyClass{
    public static void main(String args[]){
        IA ia1 = new IAImpl1();
        MyClass mc = new MyClass(ia1);
        mc.myMethod();
        IA ia2 = new IAImpl2();
        mc.setIa(ia2);
        mc.myMethod();
    }
}
```

8. *写出下面代码的输出结果

```
interface Light{
    void shine();
}

class RedLight implements Light{
    public void shine(){
        System.out.println(“Red Light shine in Red”);
    }
```

```
}

class YellowLight implements Light{
    public void shine(){
        System.out.println(“Yellow Light shine in Yellow”);
    }
}

class GreenLight implements Light{
    public void shine(){
        System.out.println(“Green Light shine in Green”);
    }
}

class Lamp{
    private Light light;
    public void setLight(Light light){
        this.light = light;
    }
    public void on(){
        light.shine();
    }
}

public class TestLamp{
    public static void main(String args[]){
        Light[] ls = new Light[3];
        ls[0] = new RedLight();
        ls[1] = new YellowLight();
        ls[2] = new GreenLight();
        Lamp lamp = new Lamp();
        for (int i = 0; i<ls.length; i++){
            lamp.setLight(ls[i]);
            lamp.on();
        }
    }
}
```

9. *写出下面代码执行的结果

```
interface JavaTeacher{
    void teach();
}
```

```
class TeacherA implements JavaTeacher{
    public void teach(){
        System.out.println(“TeacherA teach Java”);
    }
}

class TeacherB implements JavaTeacher{
    public void teach(){
        System.out.println(“TeacherB teach Java”);
    }
}

class School{
    public static JavaTeacher getTeacher(int i){
        if (i == 0) return new TeacherA();
        else return new TeacherB();
    }
}

public class TestSchool{
    public static void main(String args[]){
        JavaTeacher jt = School.getTeacher(0);
        jt.teach();
        jt = School.getTeacher(10);
        jt.teach();
    }
}
```

10. *代码填空

```
abstract class Animal{
    public abstract void eat();
}
interface Pet{
    void play();
}
class Dog extends Animal implements Pet{
    public void eat(){
        System.out.println(“Dog eat Bones”);
    }
    public void play(){
        System.out.println(“Play with Dog”);
    }
}
```

```
class Cat extends Animal implements Pet{
    public void eat(){
        System.out.println(“Cat eat fish”);
    }
    public void play(){
        System.out.println(“Play with Cat”);
    }
}

class Wolf extends Animal{
    public void eat(){
        System.out.println(“Wolf eat meat”);
    }
}

public class TestMain{
    public static void main(String args[]){
        Animal as[] = new Animal[3];
        as[0] = new Dog();
        as[1] = new Cat();
        as[2] = new Wolf();
        //调用 as 数组中所有动物的 eat 方法
        //1
        //调用 as 数组中所有宠物的 play 方法
        //2
    }
}
```

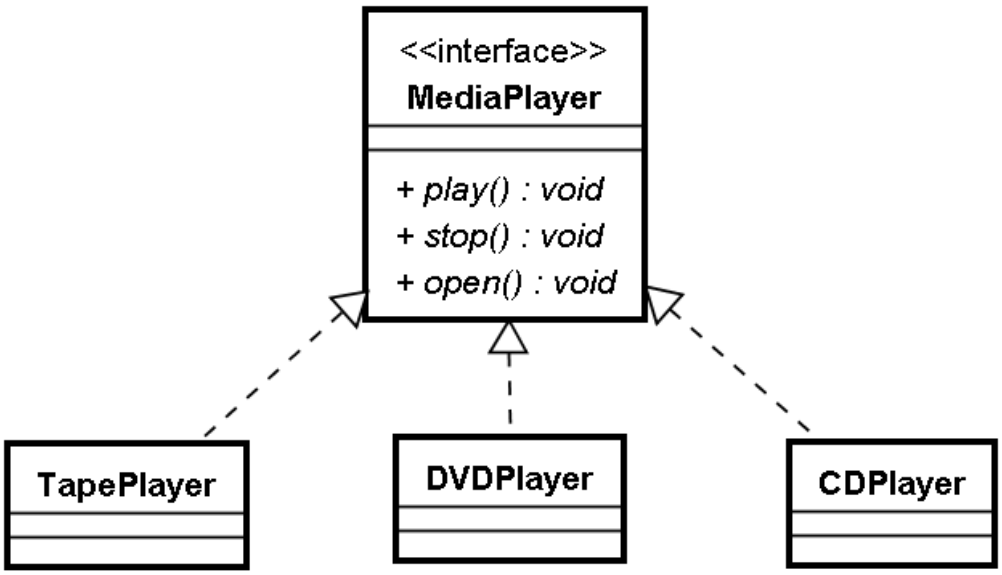
//1 处应该填入的代码为：

//2 处应该填入的代码为：

11. *定义一个接口 MediaPlayer，表示家庭影院的一个设备。MediaPlayer 中包含 play()，stop()，open()三个方法，分别表示播放、停止和开仓功能。

    MediaPlayer 有三个实现类，分别为：DVDPlayer，表示 DVD 播放器；CDPlayer，表示 CD 播放器；TapePlayer，表示录音机（播放磁带）。

    类图如下：




创建一个遥控器 Controller 类。该遥控器有三个控制通道，可以分别控制三个设备。部分代码如下：

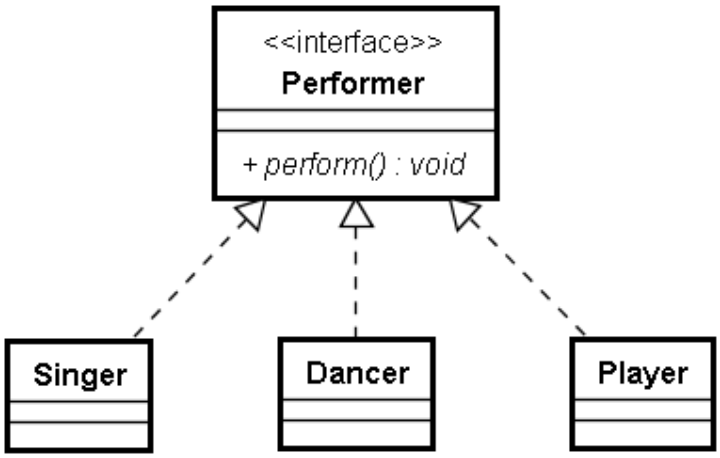
```
class Controller{
    private MediaPlayer[] players;
    public Controller(){
        //构造函数中初始化 players 数组
    }

    //对相应的设备调用 play 方法
    public void play(int i){
        players[i].play();
    }
}
```

要求：

1) 完成 MediaPlayer 接口及其子类的代码。
2) 把 Controller 补充完整，完善其构造函数，并为其增加 stop(int i)和 open(int i)方法

12. *定义一个 Performer 接口，表示一个演员，接口中定义 perform 方法，表示表演。为这个接口提供若干实现类：Singer，表示歌手；Dancer，表示舞蹈演员；Player，表示演奏者。

    类图如下：



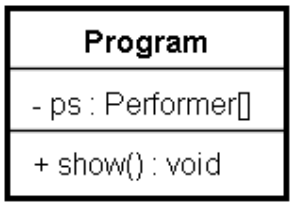

定义一个 Program 类，表示一个节目。每一个节目需要多个演员配合，因此每一个 Program 类中都包含一个属性：Performer 数组，表示表演这个节目的所需要的演员。

给出 Program 的部分代码：

```
class Program {
    private Performer[] ps;
    public Program(){
        ps = new Performer[3];
        ps[0] = new Singer();
        ps[1] = new Dancer();
        ps[2] = new Player();
    }
}
```

在现有代码基础上，为 Program 增加一个 show 方法，在这个方法中，调用所有表演这个节目的所有 Performer 的 perform 方法。

Program 类图如下：




13. *在原有的雇员练习上修改代码

公司会给 SalariedEmployee 每月另外发放 2000 元加班费，给 BasePlusSalesEmployee 发放 1000 元加班费

改写原有代码，加入以上的逻辑

并写一个方法，打印出本月公司总共发放了多少加班费

14. （强制类型转换）**有如下代码

```
interface IA {
    void m1();
    public void m2();
    public abstract void m3();
}

abstract class Super{}

class Sub1 extends Super{}

class Sub2 extends Super{}

public class TestInterface{
    public static void main(String args[]){
        Super sup = new Sub1();
        Sub1 sub1 = (Sub1)sup;
        //1
    }
}
```

在//1 处可以**编译**（不考虑运行时是否会产生异常）通过的代码为：

A． Sub2 sub2 = (Sub2) sup;

B．Sub2 sub2 = (Sub2) sub1;

C．IA ia = (IA) sup;

D． IA ia = (IA) sub1;

15. **有下列代码：

```
interface ServiceInterface{
    void doService1();
    void doService2();
    void doService3();
}

abstract class AbstractService implements ServiceInterface{
    public void doService1(){}
    public void doService2(){}
    public void doService3(){}
}
```

需要一个实现 ServiceInterface 接口的类 MyService，第一种方式可以让 MyService 实现 ServiceInterface 接口，即：

```
    class MyService implements ServiceInterface
```

第二种方式可以让 MyService 继承 AbstractService 类，即

```
class MyService extends AbstractService
```

请问：这两种方式有什么区别？AbstractService 类有什么作用？

16. **写出下面代码的运行结果

```
interface IA{
    void ma(IB ib);
}

interface IB{
    void mb();
}

class IAImpl implements IA{
    public void ma(IB ib){
        System.out.println(“ma in IAImpl”);
        ib.mb();
    }
}

class IBImpl implements IB{
    private IA ia;
    public void setIa(IA ia){
        this.ia = ia;
    }
    public void mb(){
        System.out.println(“mb in IBImpl”);
    }

    public void method(){
        ia.ma(this);
    }
}

public class TestMain{
    public static void main(String args[]){
        IA ia = new IAImpl();
        IBImpl ib = new IBImpl();
        ib.setIa(ia);
        ib.method();
    }
}
```

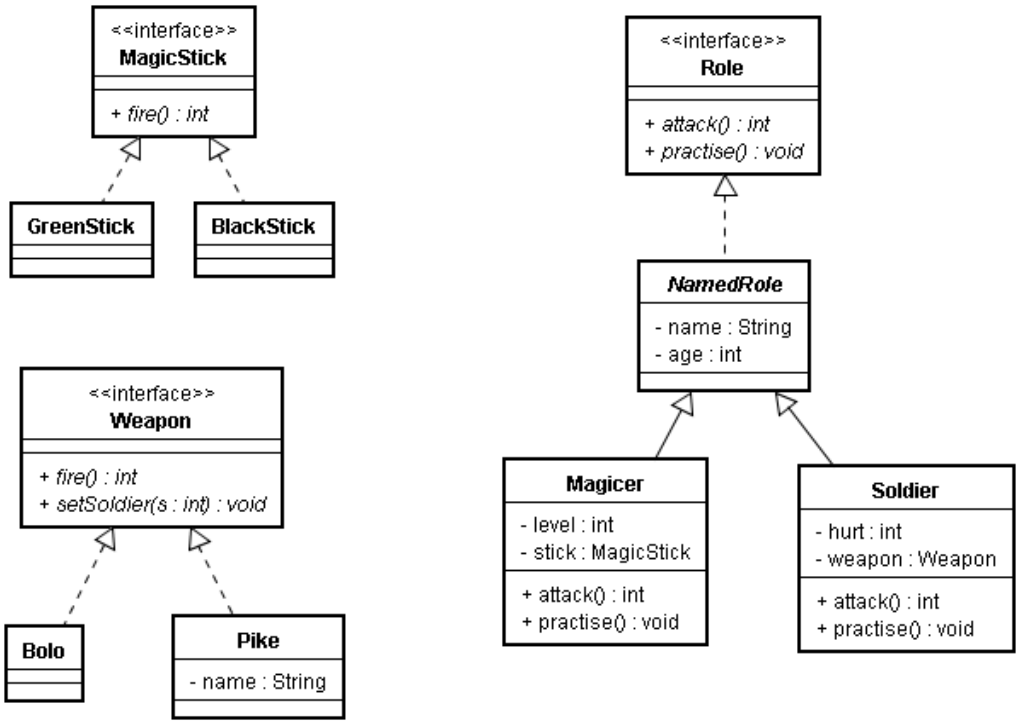
17. **在之前的游戏角色 Role 程序上进行修改。

1） 创建 Role 接口，包含两个方法：
   a) int attack(); 表示攻击，返回值表示对敌人的伤害
   b) void practise(); 表示练习。练习之后对敌人的伤害会增加。
2） 创建 NamedRole 类，该类为一个抽象类，实现了 Role 接口，并有两个属性：name 和 age，表示角色的名字和年龄。
3） 增加 MagicStick 接口。该接口表示法师使用的法杖。接口中包含一个方法，方法为：
   int fire()
4） 为 MagicStick 类增加两个实现类，分别为 GreenStick 和 BlackStick。其中，对于这两个类的 fire 方法：
   a) GreenStick 平时返回 1，夏天（6~8 月）使用时返回 2
   b) BlackStic 奇数月返回 1，偶数月返回 2

5） 修改 Magicer 类
   a) 为法师类增加 MagicStick 类的属性 stick，表示法师使用的法杖。
   b) 让其继承自 NamedRole 类，并实现 attack 和 practise 功能。其中
      i. attack 返回值为法师的魔法等级(level) *每一级的固定伤害(5)
      ii. practise()方法：
         1. 当法师的 stick 属性为 null 时，调用 practise 则 level++
         2. 当法师的 stick 不为 null 时，调用 practise 方法时，法师的等级 level 满足：level = level + 1 + stick.fire()；即：法师的等级增加为 1+stick 属性的 fire 方法的返回值

6） 增加 Weapon 接口，表示战士使用的武器。Weapon 接口中定义了两个方法：
   void setSoldier(Soldier s); 该方法表示设置武器的使用者
   int fire(); 该方法的返回值表示战士使用该武器时，对敌人的伤害值
7） 为 Weapon 增加两个实现了，一个为 Bolo，表示大刀，一个为 Pike，表示长矛。
   对这两个实现类的描述如下：
   Bolo : 当 soldier 的年龄大于等于 18 岁时，fire 方法返回 100
   当 soldier 年龄小于 18 岁时，fire 方法返回 50
   Pike：Pike 类有一个属性：name，表示长矛的名字。
   当长矛的名字和战士的名字一致时，fire 方法返回 1000；
   当长矛的名字和战士的名字不一致时，fire 方法返回 25
8） 修改 Soldier 类
   a) 为 Soldier 类增加一个 Weapon 属性，表示战士的武器
   b) 让其继承自 NamedRole 类，并实现 attack 和 practise 功能。其中
      i. Soldier 的 attack 返回值为战士的 hurt 值与武器的 fire 方法返回值的和，即
      总攻击输出 = 战士的徒手伤害值 + 武器的伤害值
      ii. practise()方法：每调用一次则战士的 hurt 值+10
9） 编写相应的测试代码。

相关类图如下：




18. **验证歌德巴赫猜想,输入一个大于 6 的偶数,请输出这个偶数能被分解为哪两个质数的和

如 10=3+7　　12=5+7

要求：**两个人一组合作完成**。一个人负责把一个整数 n 拆分成两个整数的和，另一个人负责写一个函数，判断某一个整数 a 是否是质数