

Chp6 面向对象三大特性

Key Point

- 封装/数据隐藏
- 继承的基本语法
- 访问修饰符
- 对象创建过程
- super 关键字
- 方法覆盖
- 多态的基本语法和使用
- instanceof
- 多态用在参数和返回值上

练习

1. （继承、this 和super 关键字）有以下代码

```
class Super{
    public Super() {
        System.out.println("Super()");
    }
    public Super(String str) {
        System.out.println("Super(String)");
    }
}
class Sub extends Super{
    public Sub() {
        System.out.println("Sub()");
    }
    public Sub(int i) {
        this();
        System.out.println("Sub(int)");
    }
    public Sub(String str) {
        super(str);
        System.out.println("Sub(String)");
    }
}
public class TestSuperSub{
    public static void main(String args[]) {
        Sub s1 = new Sub();
        Sub s2 = new Sub(10);
        Sub s3 = new Sub("hello");
    }
}
```

写出该程序运行的结果。

Super()
sub()
Super()
Sub()
Sub(int)
Super(String)
Sub(String)

2. (super) 看下面代码, 写出程序运行的结果

```
class Super{
    public void m1() {
        System.out.println("m1() in Super" );
    }
}

    public void m2() {
        System.out.println("m2() in Super" );
    }
}

class Sub extends Super{
    public void m1() {
        System.out.println("m1() in Sub");
        super.m1();
    }
}

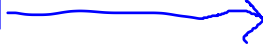
public class TestSuperSub{
    public static void main(String args[]) {
        Sub s = new Sub();
        s.m1();
        s.m2();
    }
}
```

3. (多态) 有如下代码

```
class Super{
    public void method() {
        System.out.println("method() in Super");
    }
    public void method(int i) {
        System.out.println("method(int) in Super");
    }
}

class Sub extends Super{
    public void method() {
        System.out.println("method() in Sub");
    }
    public void method(String str) {
        System.out.println("method(String) in Sub");
    }
}

public class TestSuperSub{
    public static void main(String args[]) {
        Super s = new Sub();
        s.method(10);
    }
}
```



```

        s.method();
        s.method("hello");
    }
}

```

问：该程序是否能编译通过？如果可以，输出结果是什么？如果不可以，应该如何修改？

4. （多态）有如下代码：

```

class Super{
    public void m() {
        System.out.println("m() in Super");
    }
}
class Sub extends Super{
    public void m() {
        System.out.println("m() in Sub");
    }
}

```

```

public class TestSuperSub{
    public static void foo(Super s) {
        s.m();
    }
    public static void main(String args[]) {
        Sub sub = new Sub();
        Super sup = new Super();
        foo(sup);
        foo(sub);
    }
}

```

m() in Super
m() in Sub

写出该程序编译运行后输出的结果。

5. （多态）有如下代码

```

class Animal {}
class Dog extends Animal {}
class Cat extends Animal {}
public class TestAnimal {
    public static void main(String args[]) {
        //主方法代码省略
    }
    public static Animal getAnimal() {
        //1
    }
}

```

问：下列几个选项中，有哪几个放在//1 位置能够编译通过？

A. return null;

B. return new Animal();

ALL

C. return new Dog();
D. return new Cat();

6. *(访问修饰符)有如下代码

```
//MyClass.java
package corejava.chp6;
public class MyClass{
    private int value;
    public MyClass() {}
    MyClass(int value){
        this.value = value;
    }
    public int getValue(){
        return value;
    }
    public void setValue(int value){
        this.value = value;
    }
}

//TestMyClass1.java
package corejava.chp6;
public class TestMyClass1{
    public static void main(String args[]){
        MyClass mc1 = new MyClass();
        MyClass mc2 = new MyClass(10);
        System.out.println(mc1.value);
        System.out.println(mc2.value);
    }
}

//TestMyClass2.java
package corejava.temp;
import corejava.chp6.*;
public class TestMyClass2{
    public static void main(String args[]){
        MyClass mc1 = new MyClass();
        MyClass mc2 = new MyClass(10);
        System.out.println(mc1.value);
        System.out.println(mc2.value);
    }
}
```

Handwritten annotations in blue:

- An arrow points from the `MyClass(int value)` constructor in `MyClass.java` to a handwritten version: `public MyClass(int value){ this.value=value; }`.
- A box around `System.out.println(mc1.value);` in `TestMyClass1.java` has an arrow pointing to `System.out.println(mc1.getValue());`.
- A box around `MyClass mc2 = new MyClass(10);` in `TestMyClass2.java` has an arrow pointing to `System.out.println(mc1.getValue());`.
- A box around `System.out.println(mc2.value);` in `TestMyClass2.java` has an arrow pointing to `System.out.println(mc2.getValue());`.

以上代码有哪些地方编译出错？假设不允许修改MyClass 类，那应该如何修改？

7. *(继承、访问修饰符)有如下代码

1) //MyClass.java
2) package corejava.chp6;
3) public class MyClass{

```

4)    int value;
5) }
6)
7) //MySubClass.java
8) package corejava.temp;
9) import corejava.chp6.MyClass;
10) public class MySubClass extends MyClass{
11)     public MySubClass(int value){
12)         this.value = value;
13)     }
14) }

```

选择正确答案:

- A. 编译通过
- B. 编译不通过, 应把第12 行改成super.value = value;
- C. 编译不通过, 应把第12 行改成super(value);
- D. 编译不通过, 可以为MySubClass 增加一个value 属性**
- E. 编译不通过, 把第4行改为protected int value; 把第12 行改为super.value = value;**

8. * (继承、对象构造过程) 有以下代码

```

class ClassA{
    public ClassA(){
        ② System.out.println("ClassA()");
    }
}
class ClassB{
    public ClassB(){
        System.out.println("ClassB()"); ①
    }
}
class ClassC extends ClassA{
    public ClassC(){
        System.out.println("ClassC()");
    }
}
class ClassD extends ClassB{
    private ClassA ca = new ClassA();
    private ClassC cc;
    public ClassD(){
        System.out.println("ClassD()");
    }
    ③ public ClassD(int i){
        cc = new ClassC();
        System.out.println("ClassD(int)");
    }
}

```

```

    }
    public class TestConstructors{
        public static void main(String args[]) {
            ClassD cd1 = new ClassD();
            ClassD cd2 = new ClassD(10);
        }
    }

```

编译运行以上代码，请写出运行时输出的结果

9. *（继承、对象构造过程）有以下代码

```

class Meal{
    public Meal() {
        System.out.println("Meal()");
    }
}
class Lunch extends Meal{
    public Lunch() {
        System.out.println("Lunch()");
    }
}
class Vegetable {
    public Vegetable() {
        System.out.println("Vegetable()");
    }
}
class Potato extends Vegetable{
    public Potato() {
        System.out.println("Potato()");
    }
}
class Tomato extends Vegetable{
    public Tomato() {
        System.out.println("Tomato()");
    }
}
class Meat{
    public Meat() {
        System.out.println("Meat()");
    }
}
class Sandwich extends Lunch{
    Potato p = new Potato();
    Meat m = new Meat();
    Tomato t = new Tomato();
    public Sandwich() {

```

ClassB()
ClassA()
ClassD()

ClassB()
ClassA()
ClassA()
ClassC()
ClassD(int)

顺序：

进入构造方法 ->

跳到构造方法的父类 ->

静态方法 ->

普通方法

```

        System.out.println("Sandwich()");
    }
}
public class TestSandwich{
    public static void main(String args[]){
        Sandwich s = new Sandwich();
    }
}

```

写出这段代码的输出结果。

10. *（默认构造函数）有以下代码

```

class Super{
}
class Sub extends Super{
    public Sub() {}
    public Sub(String str){
        super(str);
    }
}

```

问：该程序应该如何修改才能编译通过？

11. *（方法覆盖）有如下代码

```

class Super{
    int method(){return 0;}
}
class Sub extends Super{
    // 1
}

```

在//1 处，能编译通过的代码为：

- A. public int method(){return 0;}
- B. void method() {}
- C. void method(int n) {}

12. *（方法覆盖）有如下代码

```

class Super{
    private void method() {}
}
class Sub extends Super{
    //1
}

```

在//1 处，能编译通过的代码为：

- A. public int method(){return 0;}
- B. void method() {}
- C. void method(int n) {}
- D. private void method() {}

13. *（多态）有如下代码

```

class Super{

```

```

        public void m() {
            foo();
        }
        public void foo() {
            System.out.println("foo() in Super");
        }
    }
    class Sub extends Super{
        public void foo() {
            System.out.println("foo() in Sub");
        }
    }
    public class TestSuperSub{
        public static void main(String args[]) {
            Super s = new Sub();
            s.m();
        }
    }

```

选择正确答案

- A. 程序编译不通过
- B. 编译通过，输出foo() in Super
- C. 编译通过，输出 foo() in Sub

14. * (多态, instanceof) 有如下代码

```

class Animal{
    private String name;
    // 1
}
class Dog extends Animal{
    //2
}
class Cat extends Animal{
    //3
}
public class TestAnimal{
    public static void main(String args[]) {
        Animal[] as = new Animal[]{
            new Dog("Pluto"),
            new Cat("Tom");
            new Dog("Snoopy");
            new Cat("Garfield");
        };
        Dog[] dogs = getAllDog(as);
        for(int i = 0; i<=dogs.length; i++){
            System.out.println(dogs[i].getName());
        }
    }
}

```



```

    }
}
public static Dog[] getAllDog(Animal[] as) {
    //4
}
}

```

程序填空：

- a) 在 //1, //2, //3 处填上适当的get/set 方法和构造方法
- b) 完成//4 处的填空。getAllDog 方法从一个Animal 数组中挑选出所有的Dog 对象，并把这些对象放在一个Dog 数组中返回。

15. （封装）把Chp5 中的Worker、Address 类中所有的属性都改成私有，并提供相应的get/set方法。

16. *（封装）已知一个类Student 代码如下：

```

class Student{
    String name;
    int age;
    String address;
    String zipCode;
    String mobile;
}

```

- a) 把Student 的属性都作为私有，并提供get/set 方法以及适当的构造方法。
- b) 为Student 类添加一个getPostAddress 方法，要求返回Student 对象的地址和邮编。

17. *（封装、继承、多态）创建三个类，组成一个继承树，表示游戏中的角色。描述如下：

父类：Role。是所有职业的父亲。

属性：name，表示角色的名字。

方法：public int attack()，该方法返回值为角色的攻击对敌人的伤害。

Role 有两个子类：

1) 法师Magicer

属性：魔法等级（范围为1 ~ 10）

方法：

public int attack()，该方法返回法师的攻击对敌人造成的伤害值。

法师攻击伤害值为：魔法等级*魔法基本伤害值（固定为5）

2) 战士Soldier

属性：攻击伤害值

方法：

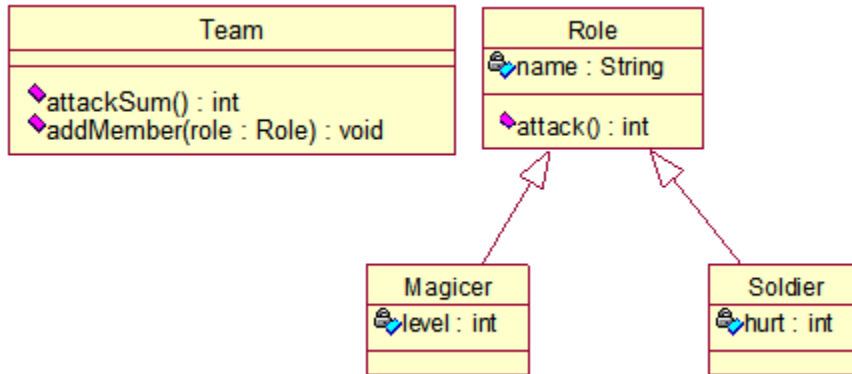
public int attack()，该方法返回战士的攻击对敌人造成的伤害值。

战士的攻击伤害值为：其攻击伤害属性值

注意：上述的三个类所有属性都应当作为私有，并提供相应的get/set 方法。

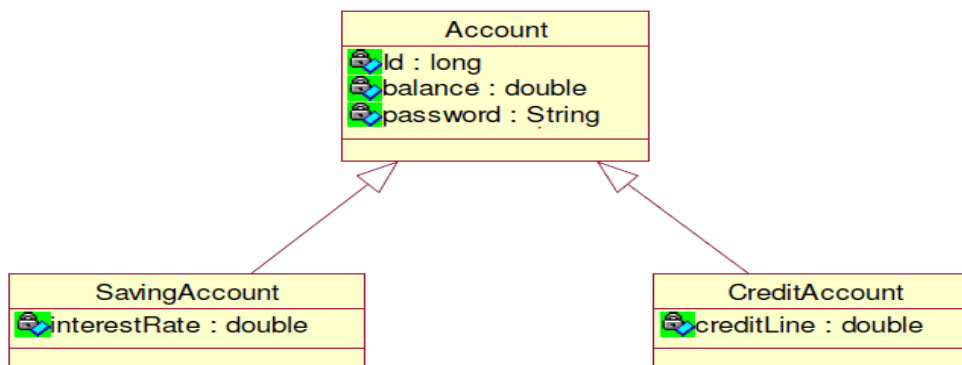
再设计一个Team 类，表示一个组队。具有如下方法

- 1) addMember, 表示组队增加一个成员。注意：组队成员最多为6 人
提示：应当利用一个数组属性，保存所有成员
- 2) attackSum, 表示组队所有成员进行攻击时，对敌人造成的总伤害值
省略 get/set 方法后的类图如下：



根据类图和描述，创建相应的类。并编写相应的测试代码。

18. *（封装、继承）设计如下的继承树：



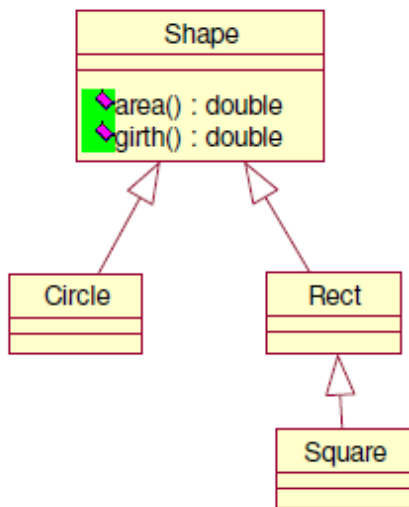
Account 表示银行账户，id 属性表示账户id，balance 表示账户余额，password 表示账户密码；

SavingAccount 表示储蓄账户，interestRate 表示存款利率；

CreditAccount 表示信用账户，creditLine 表示信用额度。

完成下列任务：

- 1) 所有属性都应设为私有，根据需要增加构造方法和get/set 方法。
 - 2) *修改setPassword 方法，要求：
setPassword 判断新密码长度是否是6 位，如果不是则不予修改；
修改getPassword 方法，要求每次都返回null 值。
 - 3) *修改 interestRate 的 set 方法，要求利率大于 0 并小于 10%。
19. *（封装、继承）有以下几个类，根据下面的继承关系，用 Java 代码实现。



- a) Circle 类（圆形），属性：半径；方法：求周长、求面积
 - b) Rect 类（矩形），属性：长、宽；方法：求周长、求面积
 - c) Square 类（正方形），属性：边长；方法：求周长、求面积
- 提示：

- 1) 这三个类均具有求周长和面积的方法；
- 2) 正方形是特殊的矩形

20. *（多态）在上一题的基础上，创建一个长度为3 的数组，里面有三个不同类型的对象，分别打印这三个对象的周长和面积。

21. *（多态）写一个函数getShape(int i)，该函数的参数为一个整数i，返回值由i 决定：

- a) i == 0 时，返回一个半径为1 的圆形
- b) i == 1 时，返回一个长为3 宽为2 的矩形
- c) i == 2 时，返回一个边长为2 的正方形

22. （多态）写一个函数，接受一个图形作为参数，打印出该图形的周长和面积

23. **（继承、访问修饰符） 有如下代码

```

//MySuperClass.java
package corejava.chp6;
class MySuperClass{
    protected int value;
}

//TestMain.java
package corejava.temp;
import corejava.chp6.*;
class MySubClass extends MySuperClass{
    public void print(){
        System.out.println(value);
    }
}
  
```

```

public class TestMain{
    public static void main(String args[]) {
        MySubClass msc = new MySubClass();
        msc.print();
        System.out.println(msc.value);
    }
}

```

这段代码能否编译通过？如果可以，输出结果是什么？如果不能，原因是什么？

24. **（封装、继承、super）某公司的雇员分为以下若干类：

Employee：这是所有员工总的父类，属性：员工的姓名, 员工的生日月份。方法：getSalary(intmonth) 根据参数月份来确定工资，如果该月员工过生日，则公司会额外奖励100 元。

SalariedEmployee：Employee 的子类，拿固定工资的员工。属性：月薪

HourlyEmployee：Employee 的子类，按小时拿工资的员工，每月工作超出160 小时的部分按照1.5 倍工资发放。属性：每小时的工资、每月工作的小时数

SalesEmployee：Employee 的子类，销售人员，工资由月销售额和提成率决定。属性：月销售额、提成率

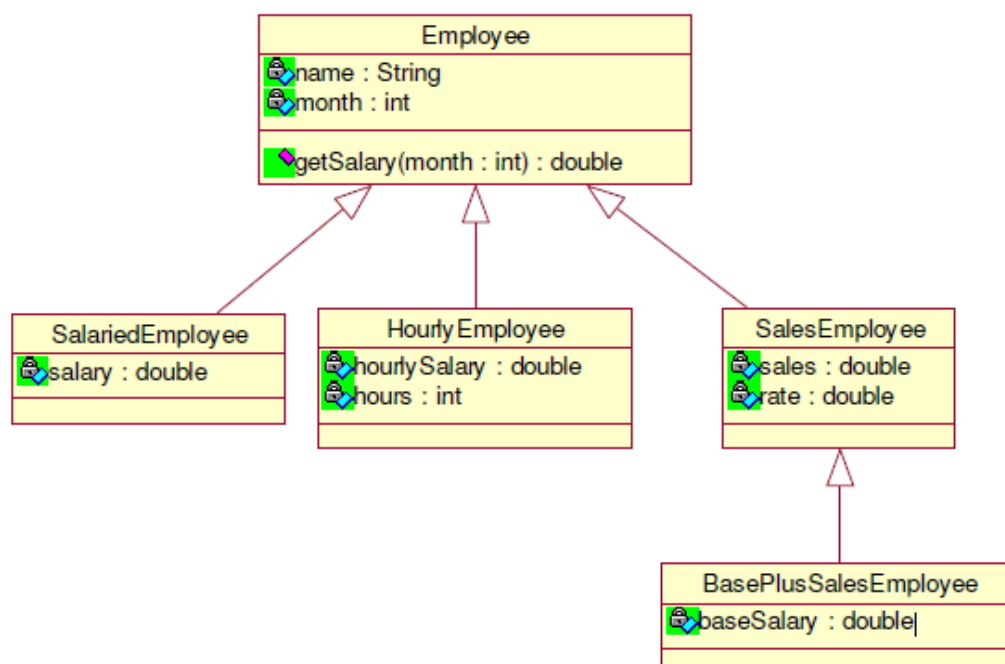
BasePlusSalesEmployee：SalesEmployee 的子类，有固定底薪的销售人员，工资由底薪加上销售提成部分。属性：底薪。

根据要求创建 SalariedEmployee、HourlyEmployees、SaleEmployee 和 BasePlusSalesEmployee

四个类的对象各一个，并计算某个月这四个对象的工资。

注意：要求把每个类都做成完全封装，不允许非私有化属性。

类图如下：



25. *（多态）在上一题的基础上，创建一个Employee 数组，分别创建若干不同的Employee对象，并打印某个月的工资。

26. **（综合）在第21 题的基础上，创建一个Bank 类，其中包括三个方法：开户、存款、取款

a) 开户：

```
Account openAccount(long id, String password, int type)
```

其中，id 表示账户id，password 表示账户密码，type 表示账户类型。如果type 为0则创建一个Account 账户，如果type 为1 则创建一个储蓄账户SavingAccount，如果type为2 则创建一个信用账户CreditAccount。返回值为开户时创建的Account 对象

b) 存款

```
double deposit(Account a, double amount)
```

其中，a 表示存入账号，amount 表示存入的金额。返回值表示存款之后的余额

c) 取款

```
double withdraw(Account a, double amount)
```

其中，a 表示取款账号，amount 表示取出的金额，返回值表示取款之后的余额。特别的，除非Account 类型是CreditAccount，否则不允许透支。