day10_polymorphic_王世杰

第一题 简答题

1. 指出多态的发生条件，并说明不能发生多态的场景。
2. 当存在父类引用指向多个子类对象时：
    1. 分别访问父子类的同名成员变量，存在多态现象吗?
    2. 分别调用父子类的同名成员方法，存在多态现象吗?
3. 引用数据类型的强制类型转换（向下转型）语句，能够通过编译的条件是什么?
4. 引用数据类型的强制类型转换（向下转型）语句，能够运行成功的条件是什么?
5. 向下转型如此危险，需要instanceof关键字判断后再进行，请描述instanceof的使用语法。

1.
    多态的发生条件：(1)继承(2)重写(3)父类引用指向子类对象
    不能发生多态的场景：
2.
    访问变量不会发生多态
    访问成员方法要看是否符合多态发生条件才能判断是否存在多态
3.
    需要转型的类名与对象的类名一致or是子类
4.
  当引用类型的真实身份是父类本身的类型时，强制类型转换就会产生错误。
                    例如：`Person person = new Person();`
  这个系统会抛出`ClassCastException`异常信息。
  所以编译器在编译时只会检查类型之间是否存在继承关系，有则通过；而在运行时就会检查它的真实类型，是则通过，否则抛出`ClassCastException`异常。
5.
```
    if(引用名 instanceof 强转的类名){

    }
```

第二题 简答题，

1. 请描述抽象类的成员特点（变量、方法、构造器、代码块等）

   注：实际上你完全可以用一句话来回答这道题。

2. 请描述接口的成员特点（变量、方法、构造器、代码块等）

3. 总结接口、抽象类、普通类的继承和实现（谁可以继承谁，谁能够实现谁，具体有什么特点等）

1.
   与普通类一样
2.
   接口的成员变量默认用 `public static final`修饰，但是不需要写明
   接口的方法默认用`public abstract`修饰，不能用`final`和`private`修饰
   接口没有构造器和构造代码块
   接口没有静态代码块
3.
   首先是继承：继承不能跨越种族
      类和类继承
         是单继承
      接口是可以继承接口的，并且是多继承
   接着是实现：
      只能是类实现接口

第三题 多态语法基础练习

请根据题目，作出合理设计，定义如下类：

父类Person
属性：String name，int age
行为：eat();

子类SouthPerson
属性：String name，int age，double salary
行为：eat()，swim()

子类NorthPerson
属性：String name，int age，double height
行为：eat()，drink()

写代码实现，eat()方法的多态效果
1，人都要吃饭
2，南方人喜欢吃米饭
3，北方人喜欢吃面食

最后，在测试类中，编写测试代码，要求进行如下测试：
1，编写测试方法，要求该方法允许传入SouthPerson对象和NorthPerson对象，并在方法体中调用它们的eat()方法。
方法调用的结果一致吗？

2，用父类引用指向子类对象的方式创建SouthPerson对象，能否直接访问salary属性和swim()方法？
如果不能，应该怎么写代码让它能够正常调用？

> 3，用父类引用指向子类对象的方式创建NorthPerson对象，能否（直接或写代码）访问salary属性和swim()方法?
> 如果不能，将该对象引用强转为SouthPerson引用，能否成功？为什么?

1.不一致，父类引用指向子类对象，编译看引用，运行结果看对象。
2.不能，直接调用，父类引用限制访问范围。需要强转
3.都不能，将该对象引用强转为SouthPerson引用，还是不能成功，因为：SouthPerson类里面没有salary成员变量和swim()方法。

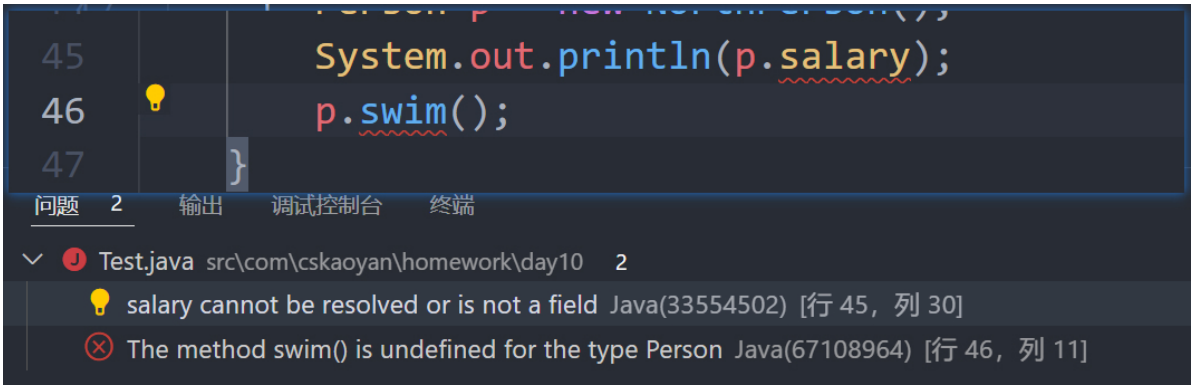
```
package com.cskaoyan.homework.day10;

// 最后，在测试类中，编写测试代码，要求进行如下测试：
public class Test {
    public static void main(String[] args) {

        Person p;
        p = new NorthPerson();
        p.eat();
        p = new SouthPerson();
        p.eat();

        System.out.println("\n测试一：");
        test1(new NorthPerson());
        test1(new SouthPerson());

        System.out.println("\n测试二：");
        test2();

        System.out.println("\n测试三：");
        test3();
    }

    // 1，编写测试方法，要求该方法允许传入SouthPerson对象和NorthPerson对象，
    // 并在方法体中调用它们的eat()方法。
    // 方法调用的结果一致吗？
    public static void test1(Person p) {
        p.eat();
    }

    // 2，用父类引用指向子类对象的方式创建SouthPerson对象，
    // 能否直接访问salary属性和swim()方法？
    // 如果不能，应该怎么写代码让它能够正常调用？
    public static void test2() {
        Person p = new SouthPerson();
        // System.out.println(p.salary);
        // p.swim();
        SouthPerson sp = (SouthPerson) p;
```

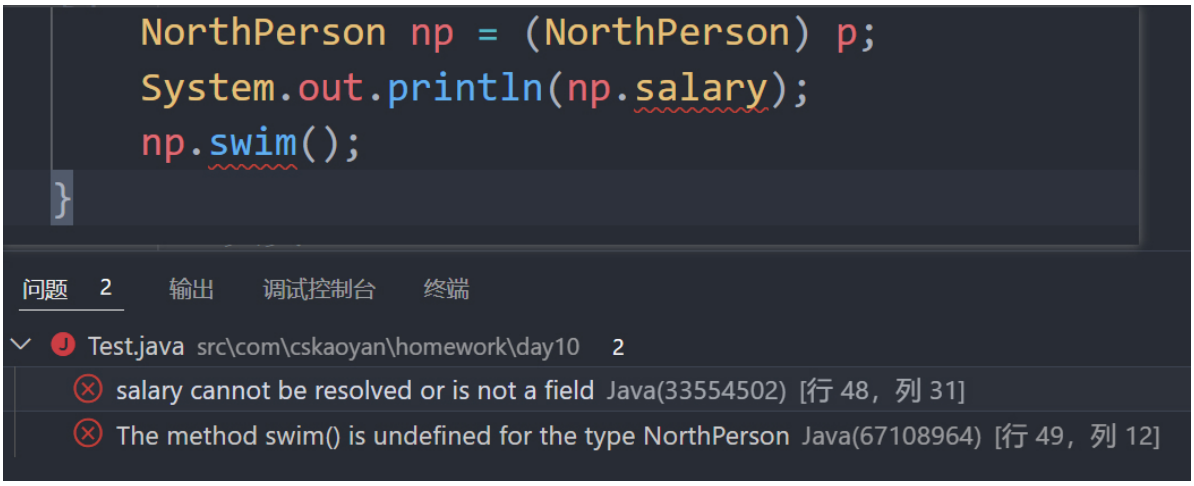
```
        System.out.println(sp.salary);
        sp.swim();
    }

    // 3，用父类引用指向子类对象的方式创建NorthPerson对象，
    // 能否（直接或写代码）访问salary属性和swim()方法？
    // 如果不能，将该对象引用强转为SouthPerson引用，能否成功？为什么？
    public static void test3() {
        Person p = new NorthPerson();
        // System.out.println(p.salary);
        // p.swim();
        NorthPerson np = (NorthPerson) p;
        // System.out.println(np.salary);
        // np.swim();
        System.out.println("np的height为："+np.height);
        np.drink();
    }

}

/*
 * 父类Person
 * 属性：String name，int age
 * 行为：eat();
 */
class Person {
    String name;
    int age;

    public void eat() {
        System.out.println(name + "在吃饭");
    }
}

/*
 * 子类SouthPerson
 * 属性：String name，int age，double salary
 * 行为：eat()，swim()
 */

class SouthPerson extends Person {
    double salary;

    @Override
    public void eat() {
        System.out.println("南方人喜欢吃米饭");
    }

    public void swim() {
```

```
        System.out.println("南方人在游泳！");
    }
}

/*
 * 子类NorthPerson
 * 属性：String name，int age，double height
 * 行为：eat()，drink()
 */
class NorthPerson extends Person {
    double height;

    @Override
    public void eat() {
        System.out.println("北方人喜欢吃面食");
    }

    public void drink() {
        System.out.println("北方人在喝酒！");
    }
}
```

```
问题    输出    调试控制台    终端

PS D:\code\homework>  d:; cd 'd:\code\homework'; & 'C:\eve\.jdks\jdk-8u311\bin\java.exe\User\workspaceStorage\8cd6434e4eeb4685d8d59e9fb1a27eee\redhat.java\jdt_ws\homework_fst'
北方人喜欢吃面食
南方人喜欢吃米饭

测试一：
北方人喜欢吃面食
南方人喜欢吃米饭

测试二：
0.0
南方人在游泳！

测试三：
np的height为：0.0
北方人在喝酒！
PS D:\code\homework>
```

```
        Person p = new SouthPerson();
        System.out.println(p.salary);
        p.swim();

Test.java src\com\cskaoyan\homework\day10  2
    salary cannot be resolved or is not a field Java(33554502) [行 36，列 34]
    The method swim() is undefined for the type Person Java(67108964) [行 37，列 15]
```





第四题 抽象类基础语法练习

> 完成抽象类的基础语法练习，按照说明操作即可。
>
> 定义抽象类A，抽象类B继承A，普通类C继承B。
> A类中，定义成员变量a赋值为10，抽象showA方法
> B类中，定义成员变量b赋值为20，抽象showB方法
> C类中，定义成员变量c赋值为30，重写showA方法打印a，重写showB方法打印b，定义showC方法，打印c
>
> 然后在测试类中，创建C类的对象，调用showA方法，showB方法，showC方法。
> 然后查看方法调用结果，思考为什么会出现这种现象。

因为`classC`对应的`C`对象继承了抽象类`B`同时间接继承了抽象类`A`，而抽象类本质和普通类只是强制要求`C`去重写他们类中的抽象方法。同时`C`会继承`AB`的成员变量。

```
package com.cskaoyan.homework.day10;

public class AbstractBasicTest {
    public static void main(String[] args) {
        C classC = new C();
        classC.showA();
        classC.showB();
        classC.showC();
    }
```

```
}

abstract class A{
    int a = 10;
    abstract void showA();
}

abstract class B extends A{
    int b = 20;
    abstract void showB();
}

class C extends B{
    int c = 30;

    @Override
    void showA() {
        System.out.println(a);
    }

    @Override
    void showB() {
        System.out.println(b);
    }

    void showC() {
        System.out.println(c);
    }
}
```

```
问题    输出    调试控制台    终端

10
20
30
PS D:\code\homework>
```

第五题 接口与抽象类基础语法练习

从实际角度出发，接口和抽象类的差异是十分明显的。可以参考完成以下案例：

学生和老师都有共同的属性：name、gender、age
共同的行为：eat() sleep()
注：虽然行为一致，但实现会不同。
现在为了提升自身素质，大家都需要额外进行技能的学习：学生需要增强实践动手能力，老师需要增强语言能力。
请定义抽象类和接口，描述以上体系。

然后用以下方式进行测试：
1，用不同的父类指向不同的子类对象，理解方法调用时“编译时看左边”
2，用这些引用调用方法，理解方法调用时“运行时看右边”

```
package com.cskaoyan.homework.day10;

public class InterfaceTest {

    public static void main(String[] args) {
        System.out.println("SchoolPerson p = new Student(\"王世杰\",
\"男\", 23);");
        SchoolPerson p = new Student("王世杰", "男", 23);
        p.eat();
        p.sleep();
        //p.practice();
        Student s = (Student) p;
        s.practice();
        System.out.println();
        System.out.println("PracticeSkilll ps = new Student(\"屈正\",
\"男\", 27);");
        PracticeSkilll ps = new Student("屈正", "男", 27);
        ps.practice();
        //ps.eat();
        //ps.sleep();
        Student s2 = (Student) ps;
        s2.eat();
        s2.sleep();
        System.out.println();
        System.out.println("Object o = new Student(\"辛冉冉\",\"男
\",27);");
        Object o = new Student("辛冉冉", "男", 27);
        //o.eat();
        //o.sleep();
        //o.practice();
        ((Student) o).practice();
        ((SchoolPerson) o).eat();
        ((SchoolPerson) o).sleep();

    }
}

abstract class SchoolPerson {
```

```
    String name;
    String gender;
    int age;

    abstract void eat();

    abstract void sleep();

    public SchoolPerson(String name, String gender, int age) {
        this.name = name;
        this.gender = gender;
        this.age = age;
    }

}

interface PracticeSkilll {
    void practice();
}

interface LanguageSkill {
    void languageLearning();
}

class Student extends SchoolPerson implements PracticeSkilll {

    public Student(String name, String gender, int age) {
        super(name, gender, age);
    }

    @Override
    void eat() {
        System.out.println(name + "在吃！");
    }

    @Override
    void sleep() {
        System.out.println(name + "在睡觉！");
    }

    @Override
    public void practice() {
        System.out.println(name + "在进行增强动手实践的技能学习");
    }

}

class Teacher extends SchoolPerson implements LanguageSkill {
```

```
    public Teacher(String name, String gender, int age) {
        super(name, gender, age);
    }

    @Override
    void eat() {
        System.out.println(name + "在吃！");
    }

    @Override
    void sleep() {
        System.out.println(name + "在睡觉！");
    }

    @Override
    public void languageLearning() {
        System.out.println("在进行增强语言能力的技能学习");
    }
}
```