操作题

以下读程序题给出必要的说明。

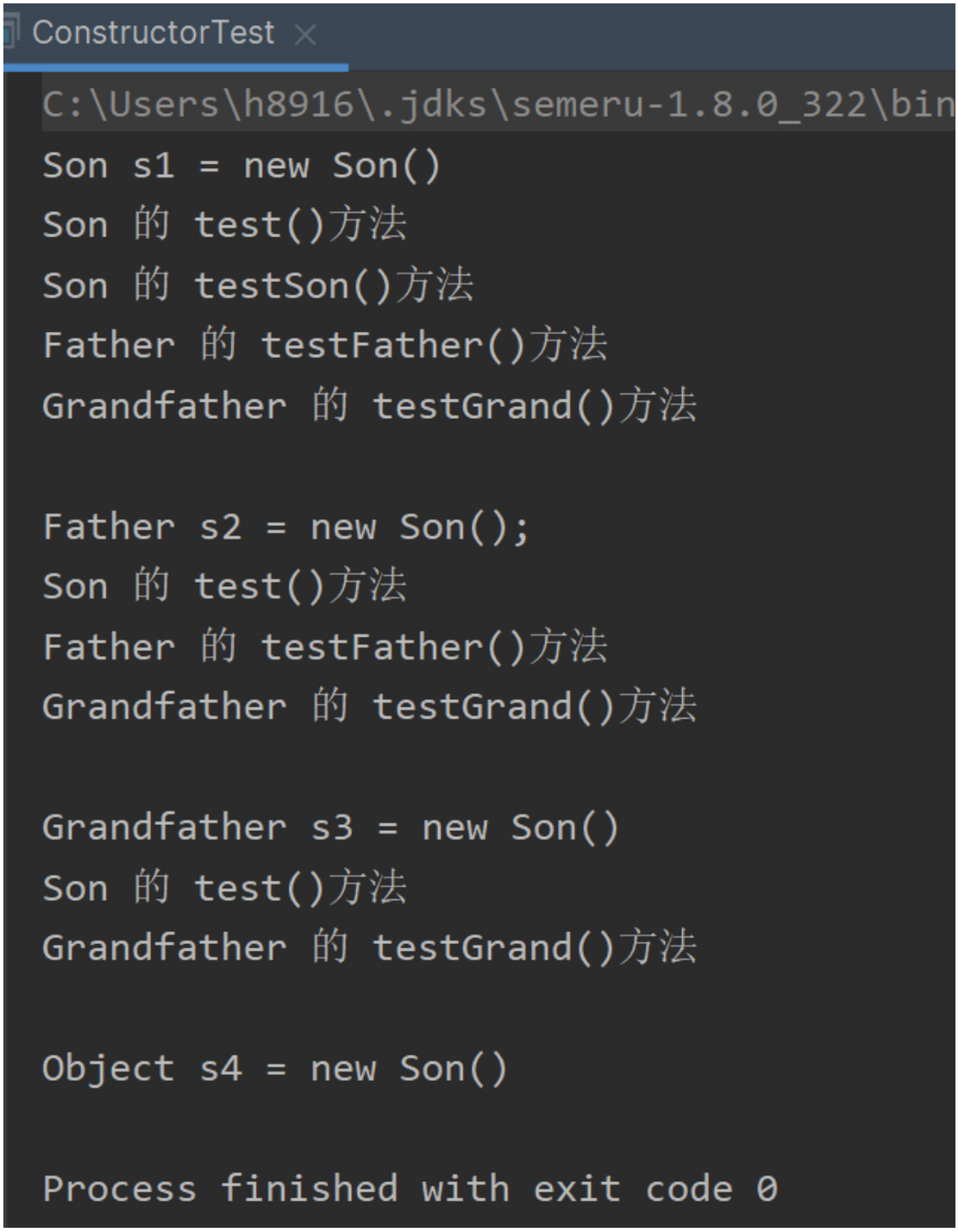
1. 将以下代码复制到IDEA中，思考代码为什么会编译报错。然后尝试修改代码，让代码不要报错。

```
class  Student{
    int age;
    String name;
    int var;
    public Student(){
    }
    public Student(int age){
        this.age = age;
    }
    public Student(String name){
        this(var);
        this.name = name;
    }
}
```

```
package com.cskaoyan.homework.day09;

public class ConstructorTest {
    public static void main(String[] args) {
        Son s1 = new Son();
        Son s12 = new Son(1,1);
        Father s2 = new Son();
        Grandfather s3 = new Son();
        Object s4 = new Son();
        System.out.println("Son s1 = new Son()");
        System.out.println(s12.a);
        s1.test();
        s1.testSon();
        s1.testFather();
        s1.testGrand();
        System.out.println("\nFather s2 = new Son();");
        s2.test();
        s2.testFather();
        s2.testGrand();
        System.out.println("\nGrandfather s3 = new Son()");
```

```
        s3.test();
        s3.testGrand();
        System.out.println("\nObject s4 = new Son()");
    }
}

class Grandfather {
    int gA;
    int a;

    public Grandfather(){

    }

    public Grandfather(int a) {
        this.a = a;
    }

    public void testGrand() {
        System.out.println("Grandfather 的 testGrand()方法");
    }

    public void test() {
        System.out.println("Grandfather 的 test()方法");
    }
}

class Father extends Grandfather {
    int fA;

    public Father() {
        super();
    }

    public Father(int a) {
        super(a);
    }

    public void testFather() {
        System.out.println("Father 的 testFather()方法");

    }

    @Override
    public void test() {
        System.out.println("Father 的 test()方法");
    }
}
```

```
class Son extends Father {
    int sA;

    public Son() {
        super();
    }

    public Son(int sA,int a) {
        super(a);
        this.sA = sA;
    }

    public void testSon() {
        System.out.println("Son 的 testSon()方法");
    }

    @Override
    public void test() {
        System.out.println("Son 的 test()方法");
    }
}
```



2. 将以下代码复制到IDEA中，然后分析过程和结果。思考结果为什么会如此，提供必要的文字说明。

```
public class Test{
    public static void main(String[] args){
        Father f1 = new Son(1000);
                Father f2 = new Father();
                Son s = new Son(1000);
    }
}

class Father {
```

```
    int i = 10;
    public Father() {
        System.out.println(getI());
    }
    public int getI() {
        return i;
    }
}

class Son extends Father {
    int i = 100;
    public Son(int i) {
        this.i = i;
    }
    public int getI() {
        return i;
    }
}
```

构造器Student（String name）的形参列表没有var；改成下列这样，因为var和age都是int型变量，如果this（var）其实调用的也还是age的单参构造器。

```
    public Student(String name,int age){
        this(age);
        this.name = name;
    }
```

简答题

1. 描述一下**方法覆盖/重写**的语法，从以下角度：
    - 访问权限修饰符
    - **返回值类型（重点测试一下）**
    - 方法名
    - 形参列表

    最后一定要使用注解 `@Override` 来检查方法重写的正确性。

语法：

```
@Override
[访问权限修饰符] 返回值类型 方法名(形参列表){
 // 方法体
}
```

1.子类重写父类方法,访问权限修饰符不需要保持一致,允许更宽松,但是不允许更严格
2.子类重写父类方法,返回值类型并不要求完全一致,需要保持兼容.
3.子类重写父类方法,方法名必须强制保持一致,不能有任何不同.
4.子类重写父类方法,形参列表必须保持一致,不能有任何不同.

2. final修饰类时表示什么？final修饰方法时表示什么？

`final`修饰类时表示不能被继承，`final`修饰方法时表示方法不能被重写

3. final修饰变量表示什么？分别从两个角度回答：
   - 修饰基本数据类型变量和引用数据类型
   - 修饰局部变量，成员变量，静态成员变量

`final`修饰基本数据类型变量表示：对于基本数据类型全局常量和`String`的字面值常量赋值的全局常量，访问它们不会触发这个类的类加载!
`final`修饰引用数据类型:会触发这个类的类加载!
`final`修饰局部变量,表示一个局部常量
`final`修饰成员变量,表示成员常量,它仍然属于对象
`final`修饰静态成员变量,表示静态成员常量,表示"全局常量",在整个类的全局唯一且不可变,是一个真正意义上的常量!!

4. 测试一下访问类的全局常量，哪些情况不会触发该类的静态代码块执行？ **(测试不同种类的全局常量)**

对于基础数据类型全局常量和`String`的字面值常量赋值的全局常量不会触发该类的静态代码块执行

5. 哪些方法不能发生方法的重写？

a.静态方法属于类,如果父子类同名是分别属于两个类的两个方法：继承不能继承,更不能重写
b.构造器：构造器不能被继承,更不能被重写
c.私有(成员)方法：虽然能够被子类继承,但是子类没有访问权限,更不能重写它

编程题

final语法练习

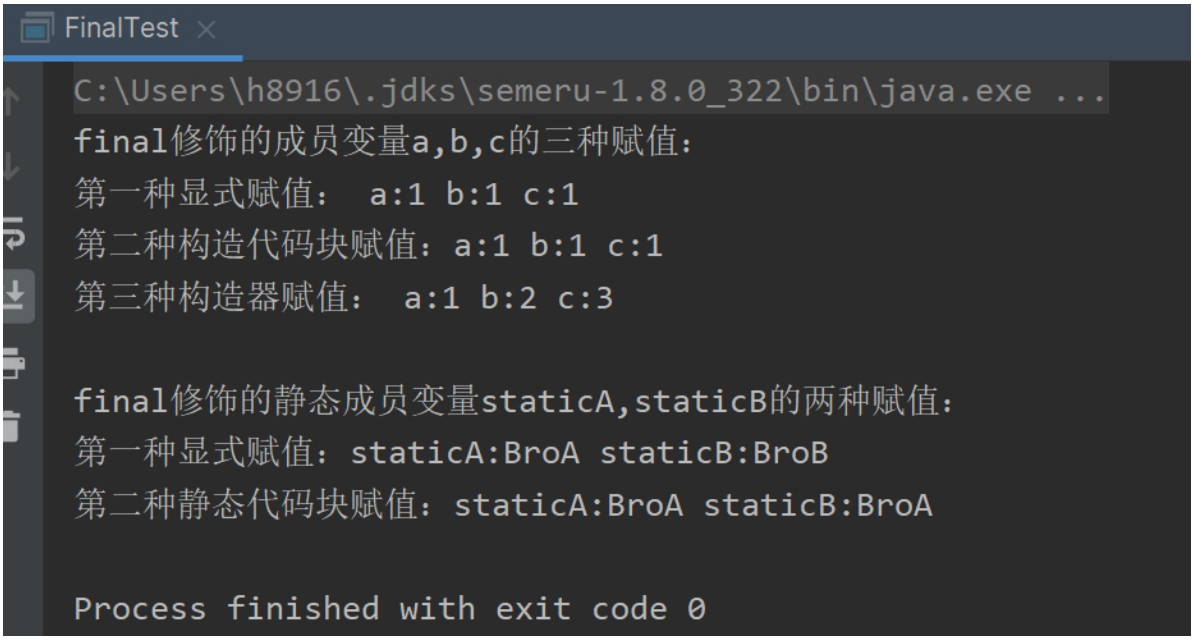
final修饰成员/静态成员变量，表示一个常量，必须明确地赋值。

自定义一个类，类中定义三个成员变量a，b，c，用final修饰这三个成员变量
再定义两个静态成员变量staticA和staticB，也用final修饰这两个静态成员变量
然后：
    1，请用三种不同的方式，分别为成员变量a，b，c赋值
    2，请用两种不同的方式，分别为静态成员变量staticA和staticB赋值

```
package com.cskaoyan.homework.day09;

public class FinalTest {
    public static void main(String[] args) {
        System.out.println("final修饰的成员变量a,b,c的三种赋值：");
        new TestABC1().print();
        new TestABC2().print();
        new TestABC3(1, 2, 3).print();

        System.out.println("\nfinal修饰的静态成员变量staticA,staticB的两
种赋值：");
        new TestStaticAB1().print();
        new TestStaticAB2().print();
    }
}

class TestStaticAB1 {
    static final String staticA = "BroA";
    static final String staticB = "BroB";

    public void print() {
        System.out.println("第一种显式赋值：" + "staticA:" + staticA + "
staticB:" + staticB);
    }
}

class TestStaticAB2 {
    static final String staticA;
    static final String staticB;

    static {
        staticA = "BroA";
```

```
        staticB = "BroA";
    }

    public void print() {
        System.out.println("第二种静态代码块赋值：" + "staticA:" +
staticA + " staticB:" + staticB);
    }
}

class TestABC1 {
    final int a = 1;
    final int b = 1;
    final int c = 1;

    public TestABC1() {
    }

    public void print() {
        System.out.print("第一种显式赋值：");
        System.out.print(" a:" + a);
        System.out.print(" b:" + b);
        System.out.println(" c:" + c);
    }
}

class TestABC2 {
    final int a;
    final int b;
    final int c;

    {
        this.a = 1;
        this.b = 1;
        this.c = 1;
    }

    public TestABC2() {
    }

    public void print() {
        System.out.print("第二种构造代码块赋值：");
        System.out.print("a:" + a);
        System.out.print(" b:" + b);
        System.out.println(" c:" + c);
    }
}

class TestABC3 {
    final int a;
```

```
    final int b;
    final int c;

    public TestABC3(int a, int b, int c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public void print() {
        System.out.print("第三种构造器赋值: ");
        System.out.print(" a:" + a);
        System.out.print(" b:" + b);
        System.out.println(" c:" + c);
    }
}
```



练习继承中的构造器使用

使用alt + insert快捷键可以快速生成各种类中结构。子类继承父类后，在子类中按alt + insert快捷键快速创建构造方法时，第一步会让你选择父类构造器（也就是super(参数)）

随后才是选择子类自身的成员变量，生成子类构造器。

当然如果选择使用父类无参构造器，这时`super()`是隐藏的，即子类对象隐式初始化。

当选择使用子类有参构造器时，这时`super(参数)`必须写在子类构造器第一行，这是子类对象的显式初始化。

给出三个层级的继承关系
顶层父类Grandfather，Father继承Grandfather，Son继承Father

成员变量：
Grandfather：gA,a
Father：fA,a
Son：sA,a

成员方法：
Grandfather：testGrand,test
Father：testFather,test
Son：testSon,test

使用快捷键在Son中生成不同的构造方法，调用不同的父类构造器（需要先生成父类相应的构造器）
随后使用多种方式创建Son对象（思考有几种），测试对象名访问成员的机制，理解属性隐藏和方法覆盖

```
package com.cskaoyan.homework.day09;

public class ConstructorTest {
    public static void main(String[] args) {
        Son s1 = new Son();
        Son s12 = new Son(1,1);
        Father s2 = new Son();
        Grandfather s3 = new Son();
        Object s4 = new Son();
        System.out.println("Son s1 = new Son()");
        System.out.println(s12.a);
        s1.test();
        s1.testSon();
        s1.testFather();
        s1.testGrand();
        System.out.println("\nFather s2 = new Son();");
        s2.test();
        s2.testFather();
        s2.testGrand();
        System.out.println("\nGrandfather s3 = new Son()");
        s3.test();
        s3.testGrand();
        System.out.println("\nObject s4 = new Son()");
    }
}

class Grandfather {
    int gA;
    int a;

    public Grandfather(){

    }
```

```
    public Grandfather(int a) {
        this.a = a;
    }

    public void testGrand() {
        System.out.println("Grandfather 的 testGrand()方法");
    }

    public void test() {
        System.out.println("Grandfather 的 test()方法");
    }
}

class Father extends Grandfather {
    int fA;

    public Father() {
        super();
    }

    public Father(int a) {
        super(a);
    }

    public void testFather() {
        System.out.println("Father 的 testFather()方法");

    }

    @Override
    public void test() {
        System.out.println("Father 的 test()方法");
    }
}

class Son extends Father {
    int sA;

    public Son() {
        super();
    }

    public Son(int sA,int a) {
        super(a);
        this.sA = sA;
    }

    public void testSon() {
```

```
        System.out.println("Son 的 testSon()方法");
    }

    @Override
    public void test() {
        System.out.println("Son 的 test()方法");
    }
}
```