1：阅读程序并写出输出结果

```
public class ConstructorDemo {
    public static void main(String[] args) {
        new Faculty("hello");
    }
}

class Person {
    public Person() {
        System.out.println("Call Person's constructor without parameter");
    }
}

class Employee extends Person {
    public Employee() {
        this("Call Employee's overloaded constructor");
        System.out.println("Call Employee's constructor without parameter");
    }

    public Employee(String s) {
        System.out.println(s);
      System.out.println("Call Employee's constructor with parameter: " + s);
    }
}

class Faculty extends Employee {
    public Faculty() {
        System.out.println("Call Faculty's constructor without parameter");
    }

    public Faculty(String s) {
     System.out.println("Call Faculty's constructor with parameter: " + s);
    }
}
```

2：阅读程序并写出输出结果 ← 这题不做

```
public class InitializationDemo{
     InitializationDemo(){
          new M();
     }
     public static void main(String[] args){
          new InitializationDemo();
          new InitializationDemo();
```

```
    }
    {
        System.out.println("InitializationDemo's instance " +
                "   initialization block");
    }
    static{
        System.out.println("InitializationDemo's static " +
                "   initialization block");
    }
}

class N{
    N(){    System.out.println("N's constructor body"); }
    {
        System.out.println(" N's instance initialization block");
    }
    static {
        System.out.println(" N's static initialization block");
    }
}
class M extends N{
    M(){    System.out.println("M's constructor body"); }
    {
        System.out.println(" M's instance initialization block");
    }
    static {
        System.out.println(" M's static initialization block");
    }
}
```

3：阅读程序并写出输出结果

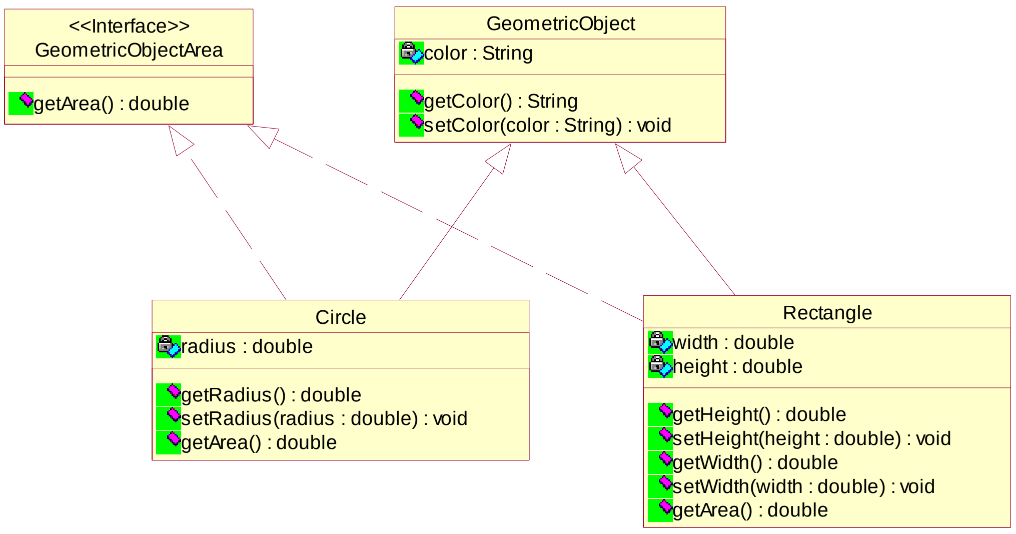
```
public class HideDemo {
   @SuppressWarnings("static-access")
    public static void main(String[] args) {
       A x = new B();
       System.out.println("(1)x.i is " + x.i);
       System.out.println("(2)x.j is " + x.j);
       System.out.println("(3)x.m1() is " + x.m1());
       System.out.println("(4)x.m2() is " + x.m2());
       System.out.println("(5)x.m3() is " + x.m3());
       B y = (B)x;
       System.out.println("(1)y.i is " + y.i);
       System.out.println("(2)y.j is " + y.j);
       System.out.println("(3)y.m1() is " + y.m1());
```

```
        System.out.println("(4)y.m2() is " + y.m2());
        System.out.println("(5)y.m3() is " + y.m3());
    }
}

class A {
    public int i = 1;
    public static int j = 11;
    public static String m1() {
        return "A's static m1";
    }
    public String m2() {
        return "A's instance m2";
    }
    public String m3() {
        return "A's instance m3";
    }
}

class B extends A {
    public int i = 2;
    public static int j = 22;
    public static String m1() {
        return "B's static m1";
    }
    public String m2() {
        return "B's instance m2";
    }
}
```

4. 用 JAVA 实现 C++课本第七章练习题 7.2 所要求的异质链表。

这题不做

5.



上图用 UML 描述了类 GeometricObject、Circle、Rectangle 和接口 GeometricObjectArea 之间的继承关系和实现关系。其中类继承 Circle、Rectangle 了类 GeometricObject；同时 Circle、Rectangle 实现了接口 GeometricObjectArea。三个类的私有属性、公有方法、以及接口声明的方法 getArea 的方法标记如图所示。

（1）　用 JAVA 定义这些类和接口，并给出每个方法的具体实现

（2）　在此基础上，实现类 GeometricObjectsManager， 该类包含一个静态方法 totalAreas。要求 totalAreas 能够计算任意个数的 Circle 对象和 Rectangle 对象的面积之和。

提示：利用接口以及接口类型的引用可以指向实现接口的类的实例，可以把 totalAreas 声明为 double totalArea(GeometricObjectArea[]);

（3）　同时， 编写一个测试类 Test，在 Test 的 main 方法中创建 10 个 Circle 对象和 10 个 Rectangle 对象，调用 GeometricObjectsManager. totalAreas 方法求出这些对象面积之和。

提示：利用数组来管理创建好的对象,将对象数组作为实参调用 GeometricObjectsManager. totalAreas 方法。