Task1

1.避免菱形问题:

当一个类继承多个具有相同方法的父类时，会产生调用歧义,
编译器无法确定应该调用哪个父类的方法,
这会导致代码逻辑混乱和不可预测的行为.

Task2

shape类接口:

```
package com.duotai;

public interface Shape {
public double calculateC();

public double calculateS();
}
```

circ类:

```
package com.duotai;

public class circle implements Shape{
private double r;
public circle(double r){
this.r=r;
}
@Override
public double calculateC() {
return 2* Math.PI*r;

    }

    @Override
    public double calculateS() {

        return Math.PI*r*r;
    }
}
```

triangle类:

```java
package com.duotai;

public class triangle implements Shape{
private double a;
private double b;
private double c;
public triangle(double a, double b , double c){
this.a=a;
this.b=b;
this.c=c;
}
@Override
public double calculateC() {
return a+b+c;
}

    @Override
    public double calculateS() {
        double s = (a + b + c) / 2;
        return Math.sqrt(s * (s - a) * (s - b) * (s - c));
    }
}
```

rectangle类:

```java
package com.duotai;

public class rectangle implements Shape{
private double a;
private  double b;
public rectangle(double a,double b){
this.a=a;
this.b=b;
}
@Override
public double calculateC() {
return 2*a+2*b;
}

    @Override
    public double calculateS() {
        return a*b;
    }
```

```
}
```

最后运行：
model类：

```
package com.duotai;

public class model {
public static void main(String[] args) {

        Shape[] shapes = new Shape[]{
                new circle(5),
                new triangle(2, 1, 2),
                new rectangle(3, 9)
        };


        for (Shape shape : shapes) {
            System.out.println("周长: " + shape.calculateC());
            System.out.println("面积: " + shape.calculateS());
            System.out.println("--------------------");
        }
    }
}
```

Task3
public：用于可以任意访问的数据
private：用于只能被该类直接访问的隐私数据
````
public class BankAccount {
// TODO 修改属性的可见性
private String accountNumber ;
public String accountHolder;
private double balance;
private String password; // 敏感信息，需要严格保护
```

```
        BankAccount(String accountNumber, String accountHolder, double
initialBalance, String password) {
            //TODO
            this.accountNumber = accountNumber;
            this.accountHolder = accountHolder;
            this.balance = initialBalance;
            this.password = password;
        }

        void deposit(double amount) {
```

```
            //TODO
            if (validateAmount(amount)) {
                balance += amount;
            }

        }

        boolean withdraw(double amount, String inputPassword) {
            //TODO
            if (!validatePassword(inputPassword) || !validateAmount(amount)) {
                return false;
            }
            if (amount > balance) {
                return false; // 余额不足
            }
            balance -= amount;
            return true;


        }

        boolean transfer(BankAccount recipient, double amount, String inputPassword) {
            //TODO
            if (!validatePassword(inputPassword) || !validateAmount(amount)) {
                return false;
            }
            if (amount > balance) {
                return false; // 余额不足
            }
            if (recipient == null) {
                return false; // 收款账户不存在
            }
            balance -= amount;
            recipient.deposit(amount);
            return true;
        }

        double getBalance() {
            //TODO
           return balance;
        }

        String getAccountInfo() {
            //TODO
```

```
54.             return "Account Number: " + accountNumber + ", Account Holder: " +
    accountHolder + ", Balance: " + balance;
55.
56.         }
57.         // 只需修改可见性
58.         private boolean validatePassword(String inputPassword) {
59.             return true;
60.         }
61.         // 只需修改可见性
62.         private boolean validateAmount(double amount) {
63.             return true;
64.         }
65.     }
```