

1.Experimental requirements

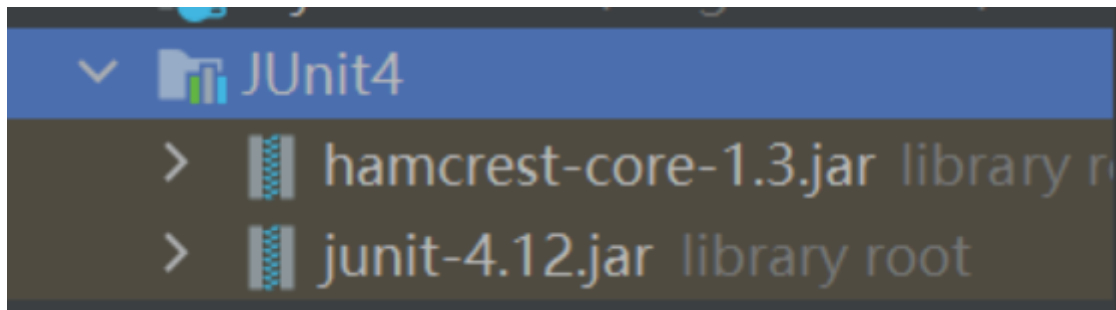
1. Install Junit(4.12), Hamcrest(1.3) with Eclipse/IDEA
2. Install Eclemma with Eclipse
3. Write a java program for the given problem and test the program with Junit.

a) Description of the problem:

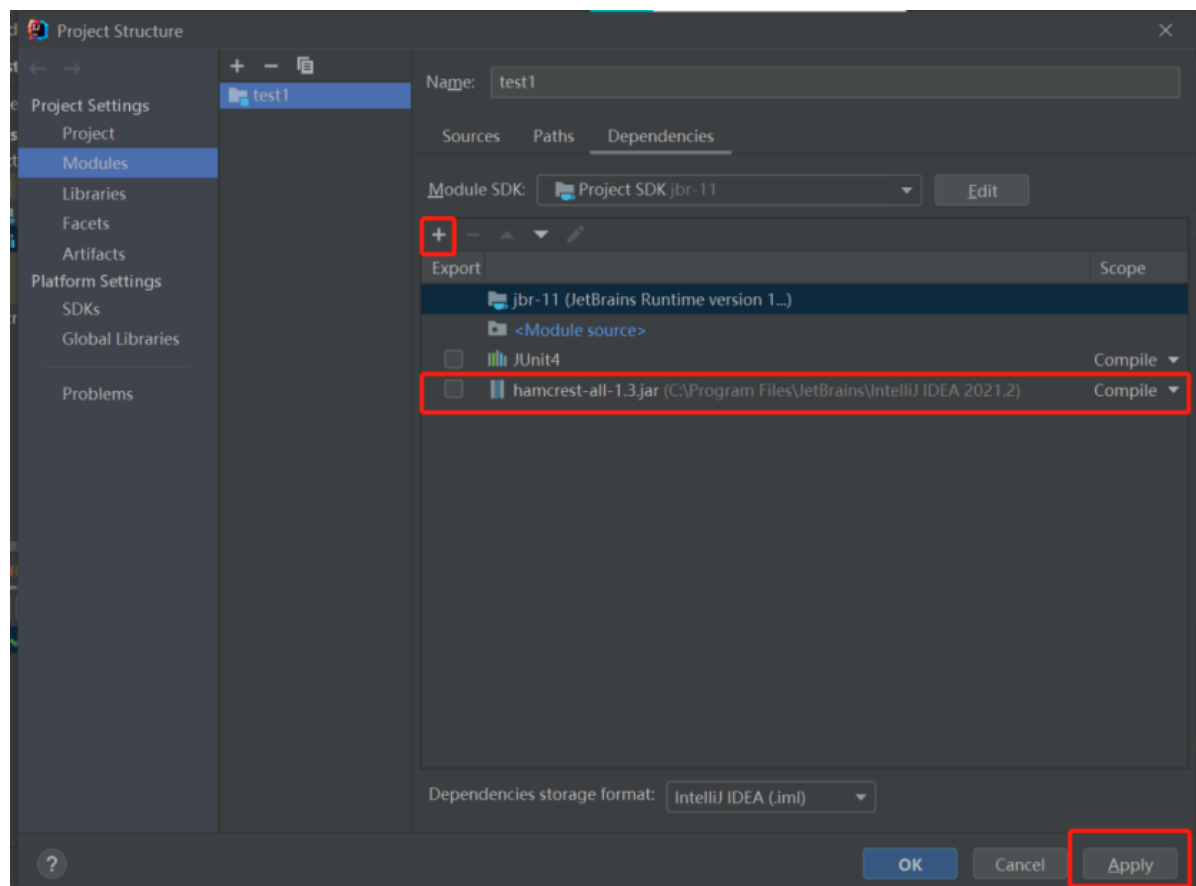
There are one 50 yuan, one 20 yuan, one 10 yuan, two 5 yuan bills and three 1 yuan coins in your pocket. Write a program to find out whether you can take out a given number (x) yuan.

2. Configuration

在IDEA上安装Junit，建立好项目后，输入@Test，会报错，然后根据提示安装JUnit，选择4.12版本，同时hamcrest-core-1.3也被安装，如图所示



File->Project Structure->Modules



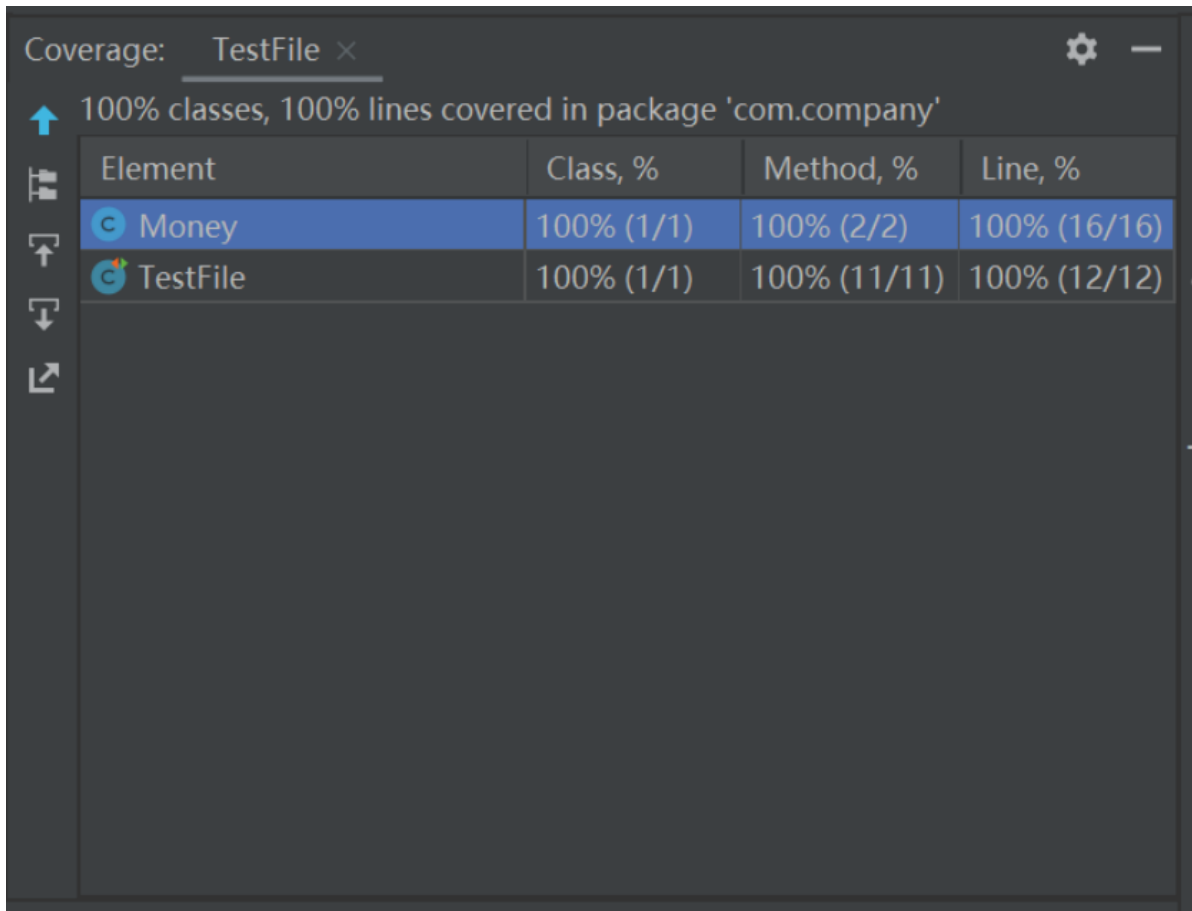
选择hamcrest-all-1.3的位置，添加到环境中。

任务1, Junit(4.12), Hamcrest(1.3)安装完毕

3. Result analysis

题目分析：对于能拿出的金额，将零钱存在数组pocket中，通过双层for循环，将所有的金额依次相加，再存入另一个数组中，若重复则不存入，从而得到了可以拿出的金额的所有可能情况。然后写一个判断函数，如果测试的金额在结果数组中，则返回true，否则返回false。然后写测试类，进行测试即可。

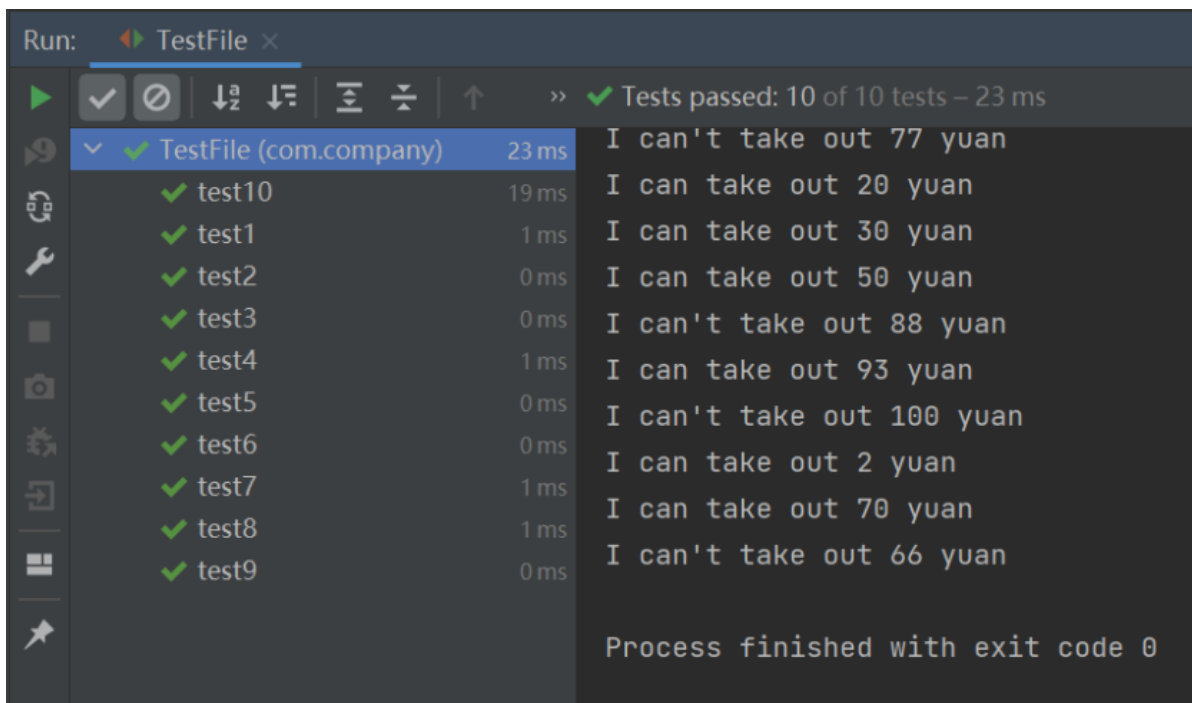
结果截图如下：



Coverage: TestFile ×

100% classes, 100% lines covered in package 'com.company'

Element	Class, %	Method, %	Line, %
Money	100% (1/1)	100% (2/2)	100% (16/16)
TestFile	100% (1/1)	100% (11/11)	100% (12/12)



Run: TestFile ×

Tests passed: 10 of 10 tests – 23 ms

Test	Duration	Output
TestFile (com.company)	23 ms	I can't take out 77 yuan
test10	19 ms	I can take out 20 yuan
test1	1 ms	I can take out 30 yuan
test2	0 ms	I can take out 50 yuan
test3	0 ms	I can't take out 88 yuan
test4	1 ms	I can take out 93 yuan
test5	0 ms	I can't take out 100 yuan
test6	0 ms	I can take out 2 yuan
test7	1 ms	I can take out 70 yuan
test8	1 ms	I can't take out 66 yuan
test9	0 ms	

Process finished with exit code 0

4. Source code

Money.java

```
1 package com.company;
2
3 import java.util.ArrayList;
4
5 public class Money {
6     public int[] pocket = {1,1,1,5,5,10,20,50};
7     public ArrayList<Integer> sum = new ArrayList<>();
8
9     public ArrayList<Integer> getMoney(){
10         for(int i = 0; i < pocket.length; i++){
11             int m = pocket[i];
12             if(!sum.contains(m))
13             {
14                 sum.add(m);
15             }
16             for(int j = i+1; j < pocket.length; j++){
17                 m += pocket[j];
18                 if(!sum.contains(m))
19                 {
20                     sum.add(m);
21                 }
22             }
23         }
24         return sum;
25         // System.out.println(sum);
26     }
27
28     public boolean canTakeMoney(int money){
29         if(getMoney().contains(money)){
30             System.out.println("I can take out "+money+" yuan");
31             return true;
32         }
33         else {
34             System.out.println("I can't take out "+money+" yuan");
35             return false;
36         }
37     }
38 }
39
```

TestFile.java

```
1 package com.company;
2
3 import org.junit.Assert;
4 import org.junit.Before;
5 import org.junit.Test;
6
7
8 public class TestFile {
9
10     public Money money;
11
```

```
12     @Before
13     public void setup() {
14         money = new Money();
15     }
16
17     @Test
18     public void test1(){
19         Assert.assertTrue(money.canTakeMoney(20));
20     }
21
22     @Test
23     public void test2(){
24         Assert.assertTrue(money.canTakeMoney(30));
25     }
26
27
28     @Test
29     public void test3(){
30         Assert.assertTrue(money.canTakeMoney(50));
31     }
32
33     @Test
34     public void test4(){
35         Assert.assertFalse(money.canTakeMoney(88));
36     }
37
38     @Test
39     public void test5(){
40         Assert.assertTrue(money.canTakeMoney(93));
41     }
42
43     @Test
44     public void test6(){
45         Assert.assertFalse(money.canTakeMoney(100));
46     }
47
48     @Test
49     public void test7(){
50         Assert.assertTrue(money.canTakeMoney(2));
51     }
52
53     @Test
54     public void test8(){
55         Assert.assertTrue(money.canTakeMoney(70));
56     }
57
58     @Test
59     public void test9(){
60         Assert.assertFalse(money.canTakeMoney(66));
61     }
62
63     @Test
64     public void test10(){
65         Assert.assertFalse(money.canTakeMoney(77));
66     }
67 }
68
```