2152118 史君宝 软件工程 软件测试

1. （ ）在软件开发的过程中，若能推迟暴露其中的错误，则为修复和改正错误所花费的代价就会降低。

A. 真

B. 假

选择：A

2. （ ）好的测试是用少量测试用例运行程序，发现被测程序尽可能多的错误。

A. 真

B. 假

选择：A

3. （ ）好的测试用例应能证明软件是正确的。

A. 真

B. 假

选择：B

4. （ ）白盒测试仅与程序的内部结构有关，完全可以不考虑程序的功能要求。

A. 真

B. 假

选择：B

5. （ ）等价类划分方法将所有可能的输入数据划分成若干部分，然后从每一部分中选取少数有代表性的数据作为测试用例。

A. 真

B. 假

选择：A

6. 使用独立测试团队的最好理由是（ ）。

A. 软件开发人员不需要做任何测试

B. 测试人员在测试开始之前不参与项目

C. 测试团队将更彻底地测试软件

D. 开发人员与测试人员之间的争论会减少

选择：C

7. 类的行为应该基于（ ）进行测试。

A. 数据流图

B. 用例图

C. 对象图

D. 状态图

选择：B

8. 下面的（ ）说法是正确的。

A. 恢复测试是以各种方式迫使软件失效从而检测软件是否能够继续执行的一种系统测试。

B. 安全测试是检测系统中的保护机制是否可以保护系统免受非正常的攻击。

C. 压力测试是检测在极限环境中使用系统时施加在用户上的压力。

D. 功能测试是根据软件需求规格说明和测试需求列表，验证产品的功能实现是否符合需求规格。

E. 安装测试是保证应用程序能够被成功地安装。

选择：B

1. 理解下面的程序结构，请使用基本路径方法设计该程序测试用例。

```
#include <stdio.h>
int partition(int *data, int low, int high)
{
    int t = 0;

    t = data[low];
    while (low < high) {
      while (low < high && data[high] >= t)
        high--;
      data[low] = data[high];
      while(low < high && data[low] <= t)
        low++;
      data[high] = data[low];
    }
    data[low] = t;

    return low;
}
```

我们给出上述程序的控制流图：

```
1. int partition(int *data, int low, int high)
2. {
3.      int t = 0;
4.      t = data[low];
5.      while (low < high) {
6.          while (low < high && data[high] >= t)
7.              high--;
8.          data[low] = data[high];
9.          while(low < high && data[low] <= t)
10.              low++;
11.          data[high] = data[low];
12.      }
13.      data[low] = t;
14.      return low;
15. }
```

则所有可能的基本路径为：

（1）测试基准路径 1-3-4-5-6-7-8-9-10-11-12-5：

输入：[3, 1, 4, 1, 5, 9, 2, 6, 5, 3] low = 0, high = 9

预期输出：low = 5，数组可能变为 [3, 1, 4, 1, 2, 3, 5, 6, 5, 9]

（2）测试基准路径 1-3-4-5-6-7-8-9-10-11-12-5-6：

输入：[3, 1, 4, 1, 5, 9, 2, 6, 5, 3] low = 0, high = 1

预期输出：low = 1，数组不变

（3）测试基准路径 1-3-4-5-6-7-8-9-10-11-12-5-9：

输入：[3, 1, 4, 1, 5, 9, 2, 6, 5, 3] low = 0, high = 2

预期输出：low = 1，数组不变

（4）测试基准路径 1-3-4-5-6-7-8-9-10-11-12-5-9-10：
输入：[3, 1, 4, 1, 5, 9, 2, 6, 5, 3] low = 0, high = 3
预期输出：low = 1，数组不变

（5）测试基准路径 1-3-4-5-6-7-8-9-10-11-12-5-9-10-11：
输入：[3, 1, 4, 1, 5, 9, 2, 6, 5, 3] low = 0, high = 4
预期输出：low = 1，数组不变

（6）测试基准路径 1-3-4-5-6-7-8-9-10-11-12-5-9-10-11-12-5：

输入：[3, 1, 4, 1, 5, 9, 2, 6, 5, 3] low = 0, high = 5
预期输出：low = 1，数组不变

2. 请使用等价类划分和边界值分析的方法，给出 getNumDaysInMonth(int month, int year)方法的测试用例，其中 getNumDaysInMonth 方法根据给定的月份和年份返回该月份的总天数。

(1) 请给出划分的等价类。

(2) 测试用例设计:

序号	输入参数	期望输出

（1）等价类：
月份：
有效月份：1-12
无效月份：小于 1 或大于 12 的值

年份：
有效年份：大于 0 的整数
无效年份：小于等于 0 的值

（2）测试用例：

输入参数：month = 2, year = 2024 期望输出：29（闰年的二月）

输入参数：month = 0, year = 2024 期望输出：错误或异常处理

输入参数：month = 13, year = 2024 期望输出：错误或异常处理

输入参数：month = 6, year = 0 期望输出：错误或异常处理

输入参数：month = 2, year = 2023　　期望输出：28

输入参数：month = 7, year = 2024　　期望输出：31

输入参数：month = 4, year = 2024　　期望输出：30

3. 现在要对一个咨询公司的薪酬支付软件进行功能测试，该软件的规格说明如下：

对于每周工作超过40小时的咨询人员，前40小时按照他们的小时薪酬计算，多余的小时数按照双倍小时薪酬计算；对于每周工作不足40小时的咨询人员，按照他们的小时薪酬计算，并生成一份缺勤报告；对于每周工作超过40小时的长期工作人员，直接按照他们的固定工资计算。

(1) 使用决策表（格式如下）描述上述的薪酬支付规则；

规则		1	2	……	k
条件桩	条件项1				
	条件项2				
	……				
	条件项m				
动作桩	动作项1				
	动作项2				
	……				
	动作项n				

(2) 根据上述决策表，使用下表列出自己设计的测试用例。

序号	输入参数	期望输出

（1）规则

条件桩：

条件项 1：工作小时数

条件项 2：员工类型（咨询人员或长期工作人员）

动作桩：

动作项 1：薪酬计算方式（按小时薪酬计算/按固定工资计算）

动作项 2：缺勤报告生成

决策表：

规则	条件项 1	条件项 2	动作项 1	动作项 2：
1	大于等于 40	咨询人员	按照规定计算	无
2	小于 40	咨询人员	按照规定计算	生成缺勤报告
3	大于等于 40	长期工作人员	按照固定工资计算	无

（2）测试用例：

输入： 工作小时数：45，员工类型：咨询人员
输出： 薪酬计算方式：前 40 小时按照小时薪酬计算，超过部分按照双倍小时薪酬计算

输入： 工作小时数：35，员工类型：咨询人员
输出： 薪酬计算方式：按照小时薪酬计算，生成缺勤报告

输入： 工作小时数：42，员工类型：长期工作人员
输出： 薪酬计算方式：按照固定工资计算，无缺勤报告生成

输入： 工作小时数：38，员工类型：长期工作人员
输出： 薪酬计算方式：按照固定工资计算，无缺勤报告生成

输入： 工作小时数：40，员工类型：咨询人员
输出： 薪酬计算方式：按照规定计算，无缺勤报告生成

输入： 工作小时数：30，员工类型：长期工作人员
输出： 薪酬计算方式：按照固定工资计算，无缺勤报告生成

4. 下面给出的是购买车票（PurchaseTicket）用例的正常流，除此之外还应包括无零钱找（NoChange）、缺票（OutofOrder）、超时（TimeOut）和取消（Cancel）等四个备选流，请结合场景法和其他方法设计测试用例。

用例名称	购买车票
前置条件	乘客站在售票机前，有足够的钱买车票。
正常流	1. 乘客选择需要达到的地点，如果按下了多个地点按钮，售票机只考虑最后一次按下的地点。 2. 售票机显示出应付款数。 3. 乘客投入钱。 4. 如果乘客在投入足够的钱之前选择了新的地点，售票机应该把所有的钱退还乘客。 5. 如果乘客投入的钱比应付款多，售票机应该退出多余的零钱。 6. 售票机给出车票。 7. 乘客拿走找零和车票。
后置条件	乘客买到了他选择的车票。

要求：使用下表列出自己设计的测试用例。

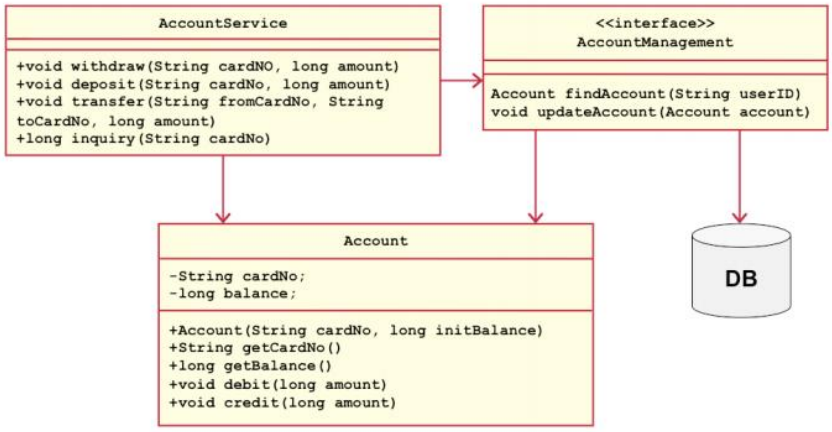
测试用例编号	
测试标题	
预置条件	
操作步骤	
预期输出	

测试样例：

编号	测试标题	预置条件	操作步骤	预期输出
1	正常购买车票	乘客站在售票机前，有足够的钱购买车票。	1. 选择目的地按钮。2. 售票机显示应付款数。 3. 投入足够的钱。 4. 售票机给出车票。 5. 拿走找零和车票。	乘客成功购买到选择的车票。
2	无零钱找雪	乘客站在售票机前，有足够的钱购买车票。	1. 选择目的地按钮。2. 售票机显示应付款数。 3. 投入稍多的钱。 4. 售票机给出车票。 5. 拿走找零和车票。	售票机未找零，乘客购买到车票。
3	缺票情况	乘客站在售票机前，有足够的钱购买车票。	1. 选择目的地按钮。2. 售票机显示应付款数。 3. 选择一个不可用的目的地。4. 投入足够的钱。	售票机显示“缺票”，乘客未购买到车票。

编号	测试标题	预置条件	操作步骤	预期输出
4	超时取消购买	乘客站在售票机前，有足够的钱购买车票。	1. 选择目的地按钮。 2. 售票机显示应付款数。 3. 超过规定时间未投入钱。	售票机显示“超时”，交易取消，乘客未购买到车票。
5	变更目的地取消	乘客站在售票机前，有足够的钱购买车票。	1. 选择目的地按钮。 2. 售票机显示应付款数。 3. 投入部分钱。 4. 变更目的地选择。 5. 售票机退还所有钱。	售票机退还所有钱，交易取消，乘客未购买到车票。

下面的 UML 图显示了银行卡账户的服务功能：



要求：

(1) 请使用 Java 或 C++语言和测试驱动开发方法编写上述程序，并进行单元测试和代码覆盖分析；

(2) 如果银行卡的转账服务增加一个新的需求“一次转账金额限定为 3000 元”，请修改代码并重新进行单元测试和覆盖分析。

（1） java 代码：

```
// Account.java
public class Account {
    private String cardNo;
    private long balance;

    public Account(String cardNo, long initBalance) {
```

```
        this.cardNo = cardNo;
        this.balance = initBalance;
    }

    public String getCardNo() {
        return cardNo;
    }

    public long getBalance() {
        return balance;
    }

    public void debit(long amount) {
        balance -= amount;
    }

    public void credit(long amount) {
        balance += amount;
    }
}

// AccountService.java
public class AccountService implements AccountManagement {
    private Map<String, Account> accounts = new HashMap<>();

    public Account findAccount(String userID) {
        return accounts.get(userID);
    }

    public void updateAccount(Account account) {
        accounts.put(account.getCardNo(), account);
    }

    public void withdraw(String cardNo, long amount) {
        Account account = accounts.get(cardNo);
        if (account != null) {
            account.debit(amount);
            updateAccount(account);
        }
    }

    public void deposit(String cardNo, long amount) {
        Account account = accounts.get(cardNo);
        if (account != null) {
```

```
            account.credit(amount);
            updateAccount(account);
        }
    }

    public void transfer(String fromCardNo, String toCardNo, long amount)
{
        Account fromAccount = accounts.get(fromCardNo);
        Account toAccount = accounts.get(toCardNo);
        if (fromAccount != null && toAccount != null) {
            fromAccount.debit(amount);
            toAccount.credit(amount);
            updateAccount(fromAccount);
            updateAccount(toAccount);
        }
    }

    public long inquiry(String cardNo) {
        Account account = accounts.get(cardNo);
        return account != null ? account.getBalance() : 0;
    }
}
```

单元测试：

```
// AccountServiceTest.java
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class AccountServiceTest {

    @Test
    public void testWithdraw() {
        AccountService accountService = new AccountService();
        Account account = new Account("1234567890", 5000);
        accountService.updateAccount(account);

        accountService.withdraw("1234567890", 2000);

        assertEquals(3000, accountService.inquiry("1234567890"));
    }

    @Test
    public void testTransfer() {
```

```
        AccountService accountService = new AccountService();
        Account fromAccount = new Account("1111111111", 5000);
        Account toAccount = new Account("2222222222", 3000);
        accountService.updateAccount(fromAccount);
        accountService.updateAccount(toAccount);

        accountService.transfer("1111111111", "2222222222", 2000);

        assertEquals(3000, accountService.inquiry("1111111111"));
        assertEquals(5000, accountService.inquiry("2222222222"));
    }
}
```

（2） 新需求：

```
// AccountService.java
public void transfer(String fromCardNo, String toCardNo, long amount) {
    if (amount <= 3000) {
        Account fromAccount = accounts.get(fromCardNo);
        Account toAccount = accounts.get(toCardNo);
        if (fromAccount != null && toAccount != null) {
            fromAccount.debit(amount);
            toAccount.credit(amount);
            updateAccount(fromAccount);
            updateAccount(toAccount);
        }
    } else {
        System.out.println("Transfer amount exceeds limit of 3000");
    }
}
```