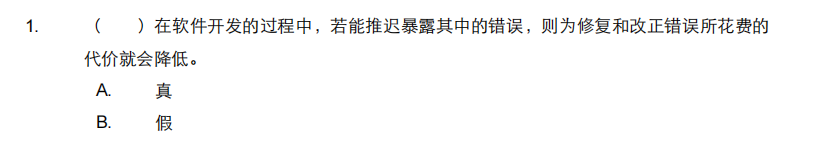
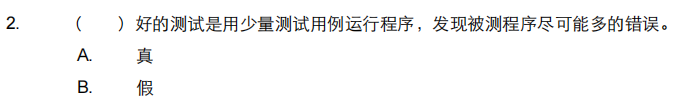
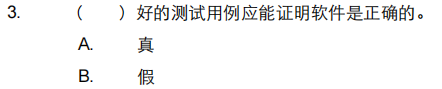
2152118 史君宝 软件工程 软件测试



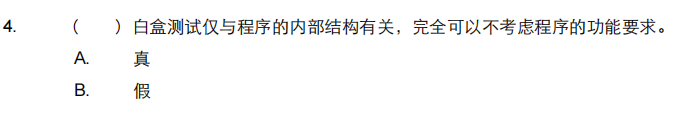
选择：A



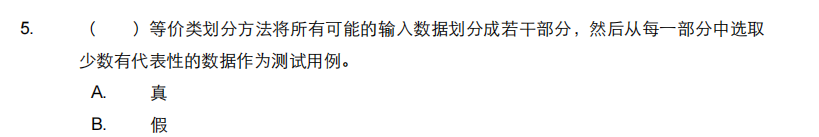
选择：A



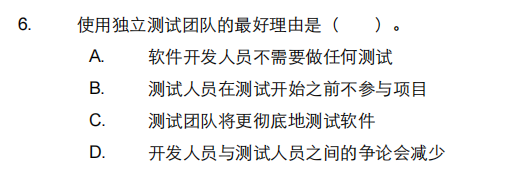
选择：B



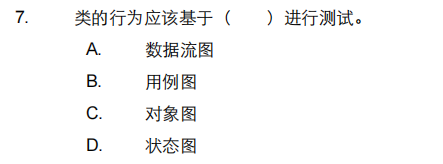
选择：B



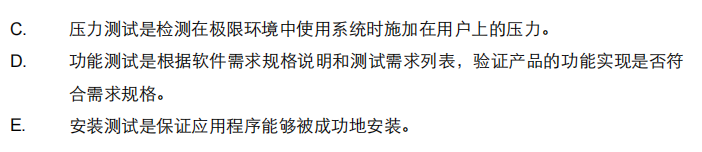
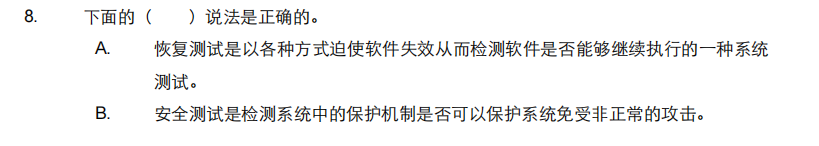
选择：A



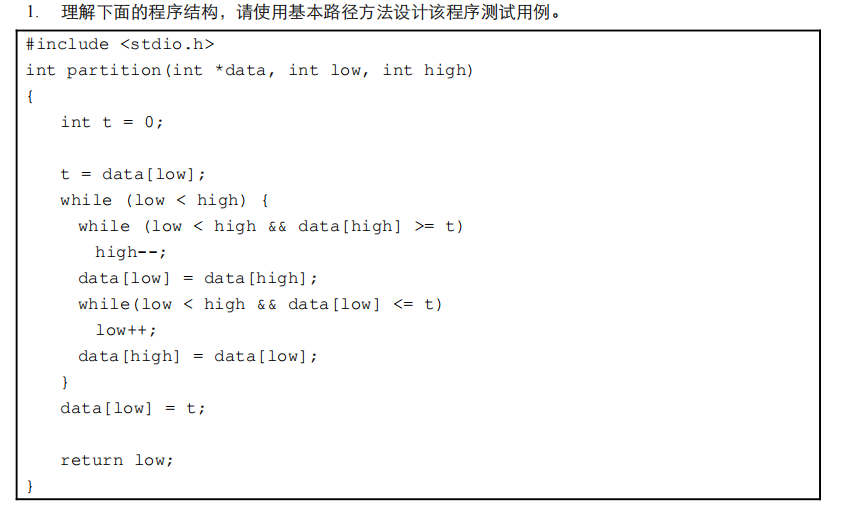
选择：C



选择：B



选择：B



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：

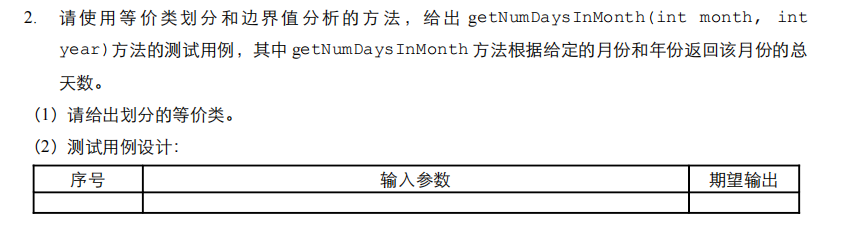
输入：[3, 1, 4, 1, 5, 9, 2, 6, 5, 3] low = 0, high = 4

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



（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

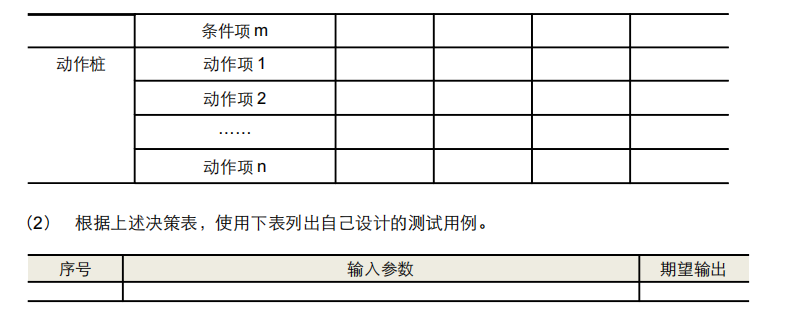
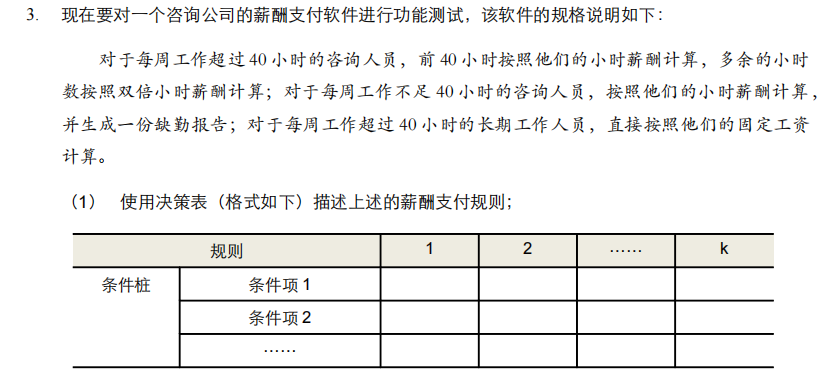
输入参数：month = 13, year = 2024 期望输出：错误或异常处理

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



（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

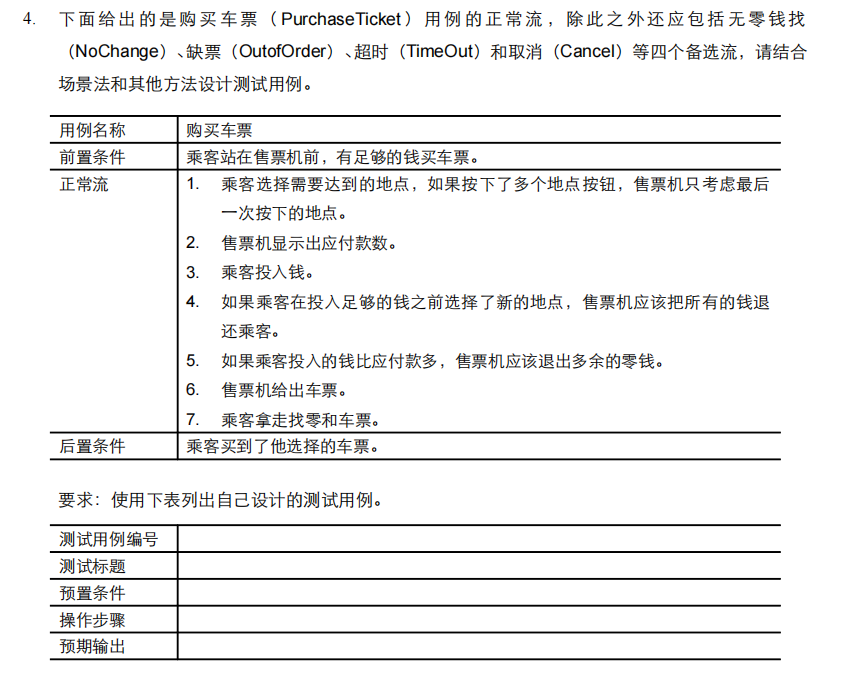
输出： 薪酬计算方式：按照固定工资计算，无缺勤报告生成

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

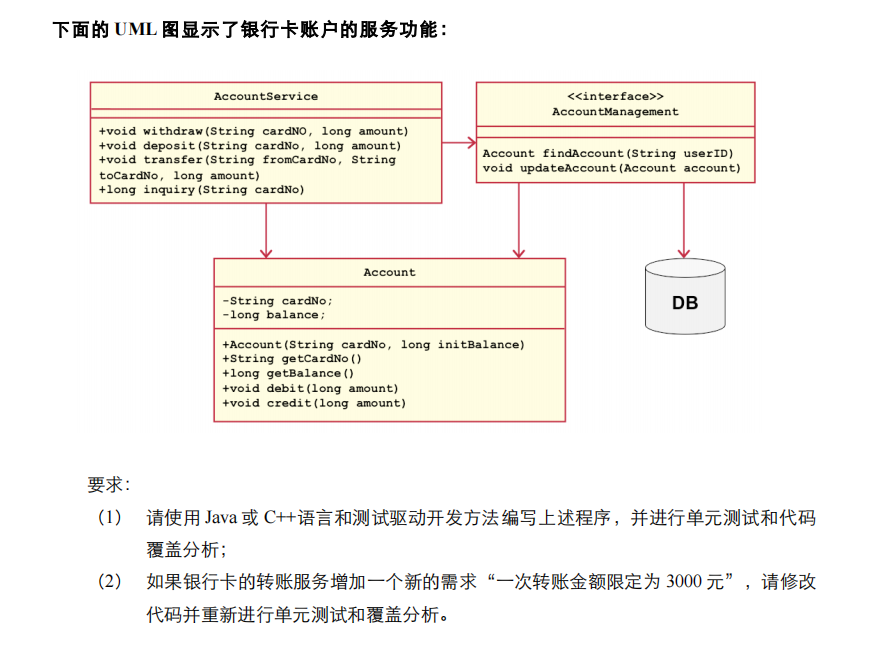
输入： 工作小时数：30，员工类型：长期工作人员

输出： 薪酬计算方式：按照固定工资计算，无缺勤报告生成



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。 |
| 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。 |



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"));

}

}

1. 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");

}

}