**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **Java语言程序设计** |
| **学生姓名：** |  |
| **学生学号：** |  |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2024-2025学年第一学期** |

**软件学院**

**2023年11月**

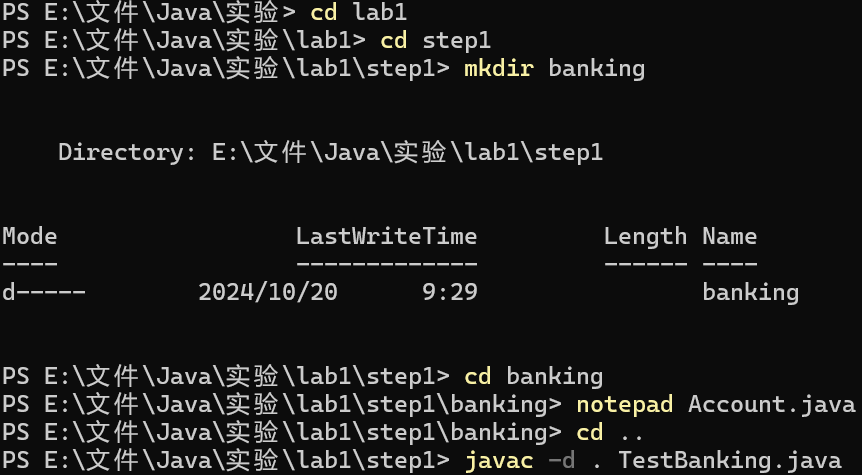
**实验一 熟悉Java 运行环境**

**一、实验目的**

1. 安装与配置JDK环境变量；
2. 掌握Java程序的编辑、编译和运行过程，熟悉Javac和Java命令
3. 掌握Java的基本语法；
4. 使用类来封装对象的属性和方法；熟悉使用package和import语句。

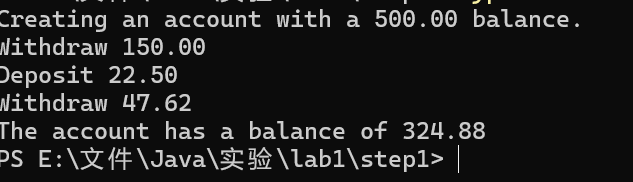
**二、实验内容及结果**

**1. step1**

**（1）. 使用命令行创建文件夹以及.java文件**

**（2）在banking中写入代码**：

1. **package** banking;
3. **public** **class** Account {
4. // 声明余额属性
5. **private** **double** balance;
6. // 构造函数初始化余额
7. **public** Account(**double** init\_balance) {
8. **this**.balance = init\_balance;
9. }
10. // 获取当前余额
11. **public** **double** getBalance() {
12. **return** balance;
13. }
14. // 存款方法
15. **public** **void** deposit(**double** amount) {
16. **if** (amount > 0) {
17. balance += amount;
18. }
19. }
20. // 取款方法
21. **public** **void** withdraw(**double** amount) {
22. **if** (amount > 0 && amount <= balance) {
23. balance -= amount;
24. }
25. }
26. }

（**3）得到实验结果**

**2. step2**

2.1进入工作目录： 在命令行中，切换到 /step2 目录：

2.2创建 banking 目录： 在命令行中输入命令创建 banking 目录：

2.3复制文件： 从上一个实验 step1 中将所有 Java 文件复制到当前的 banking 目录中：

2.4创建 Customer 类：

2.5进入 banking 目录：

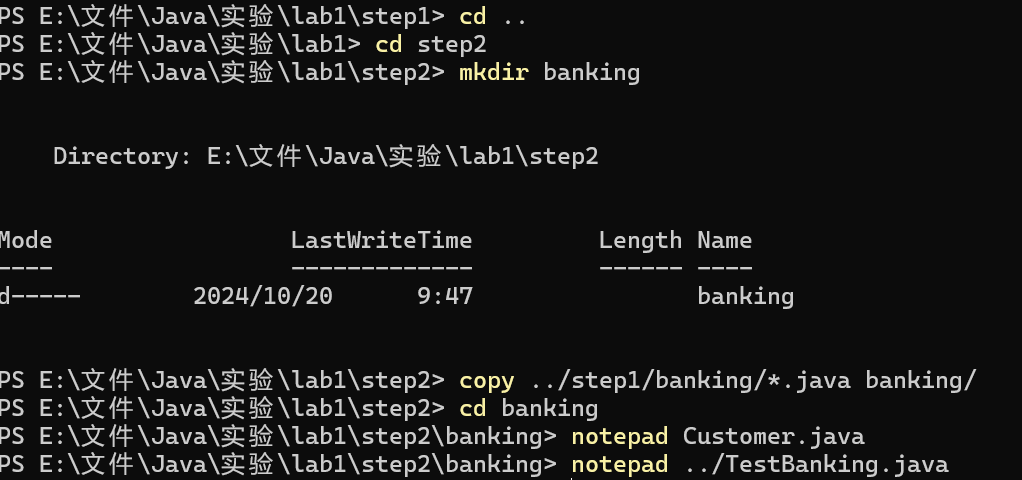
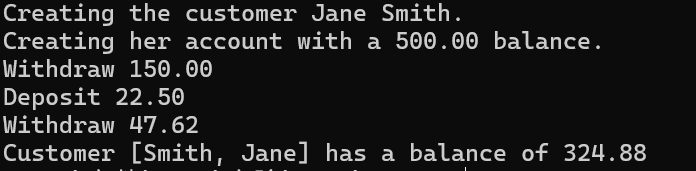
2.6使用记事本创建 Customer.java 文件：

2.7在 Customer.java 中输入我们按要求写出的代码：

1. **package** banking;
2. **public** **class** Customer {
3. // 声明私有属性
4. **private** String firstName;
5. **private** String lastName;
6. **private** Account account;
7. // 构造函数初始化 firstName 和 lastName
8. **public** Customer(String f, String l) {
9. **this**.firstName = f;
10. **this**.lastName = l;
11. }
12. // 获取 firstName
13. **public** String getFirstName() {
14. **return** firstName;
15. }
16. // 获取 lastName
17. **public** String getLastName() {
18. **return** lastName;
19. }
20. // 设置账户
21. **public** **void** setAccount(Account account) {
22. **this**.account = account;
23. }
24. // 获取账户
25. **public** Account getAccount() {
26. **return** account;
27. }
28. }

2.8修改 TestBanking 类： 在 TestBanking.java 中，更新代码以测试 Customer 类：

1. /\*
2. \* This class creates the program to test the banking classes.
3. \* It creates a new Bank, sets the Customer (with an initial balance),
4. \* and performs a series of transactions with the Account object.
5. \*/
6. **import** banking.\*;
7. **public** **class** TestBanking {
8. **public** **static** **void** main(String[] args) {
9. Customer customer;
10. Account  account;
11. System.out.println("Creating the customer Jane Smith.");
12. customer = **new** Customer("Jane", "Smith");
14. System.out.println("Creating her account with a 500.00 balance.");
15. customer.setAccount(**new** Account(500.00));
16. account = customer.getAccount();
17. System.out.println("Withdraw 150.00");
18. account.withdraw(150.00);
19. System.out.println("Deposit 22.50");
20. account.deposit(22.50);
21. System.out.println("Withdraw 47.62");
22. account.withdraw(47.62);
23. System.out.println("Customer [" + customer.getLastName() + ", " + customer.getFirstName() + "] has a balance of " + account.getBalance());
24. }
25. }

2.9预期输出：

**3.step3**

3.1进入工作目录： 在命令行中，切换到 /step3 目录：

3.2创建 banking 目录并复制文件： 在命令行中，创建 banking 目录并从上一个实验中3.3修改 Account 类：

1. **package** banking;
2. **public** **class** Account {
3. **private** **double** balance;
4. // 构造函数初始化余额
5. **public** Account(**double** init\_balance) {
6. **this**.balance = init\_balance;
7. }
8. // 获取当前余额
9. **public** **double** getBalance() {
10. **return** balance;
11. }
12. // 存款方法，返回 true
13. **public** **boolean** deposit(**double** amount) {
14. **if** (amount > 0) {
15. balance += amount;
16. **return** **true**;
17. }
18. **return** **false**;
19. }
20. // 取款方法，条件化取款，返回 true 或 false
21. **public** **boolean** withdraw(**double** amount) {
22. **if** (amount > 0 && amount <= balance) {
23. balance -= amount;
24. **return** **true**;
25. }
26. **return** **false**;
27. }
28. }

3.4使用记事本打开 Account.java 文件进行修改：

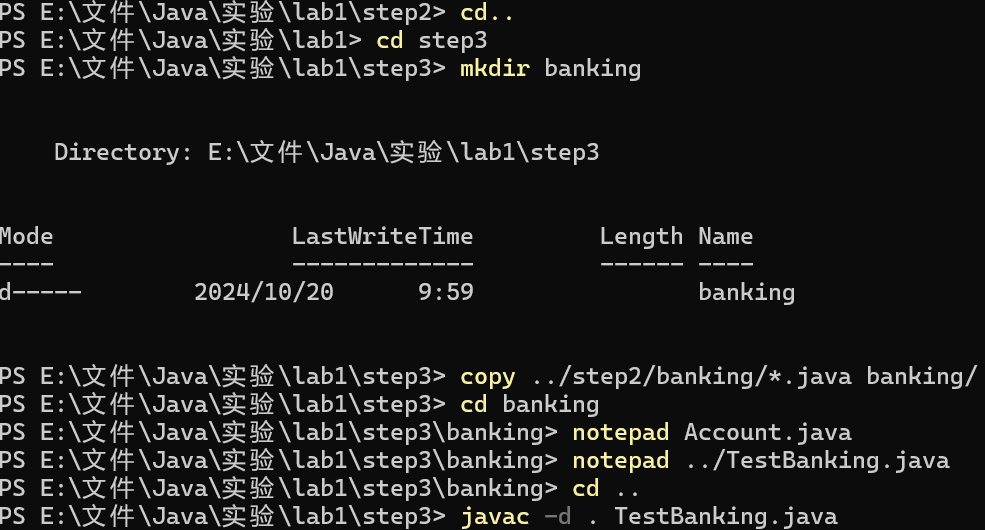
3.5修改 Account.java，将 deposit 和 withdraw 方法更新为返回布尔值：

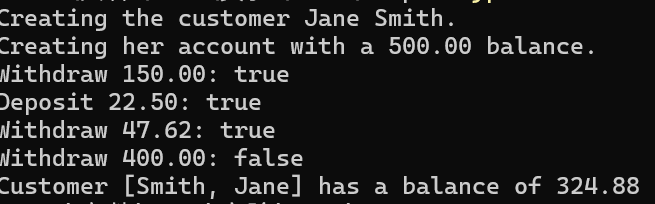
3.6修改 TestBanking 类：

使用记事本打开 TestBanking.java 文件，更新代码以测试 withdraw 方法的布尔返回值

1. **import** banking.\*;
2. **public** **class** TestBanking {
3. **public** **static** **void** main(String[] args) {
4. Customer customer;
5. Account  account;
6. System.out.println("Creating the customer Jane Smith.");
7. customer = **new** Customer("Jane", "Smith");
8. System.out.println("Creating her account with a 500.00 balance.");
9. customer.setAccount(**new** Account(500.00));
10. account = customer.getAccount();
11. System.out.println("Withdraw 150.00: " + account.withdraw(150.00));
12. System.out.println("Deposit 22.50: " + account.deposit(22.50));
13. System.out.println("Withdraw 47.62: " + account.withdraw(47.62));
14. System.out.println("Withdraw 400.00: " + account.withdraw(400.00));
15. System.out.println("Customer [" + customer.getLastName() + ", " + customer.getFirstName() + "] has a balance of " + account.getBalance());
16. }
17. }

3.7编译和运行程序：

* 返回到 /step3 目录：
* 编译 TestBanking.java 文件：

3.8预期输出：

**4. step4**

4.1在命令行中，切换到 /step4 目录：

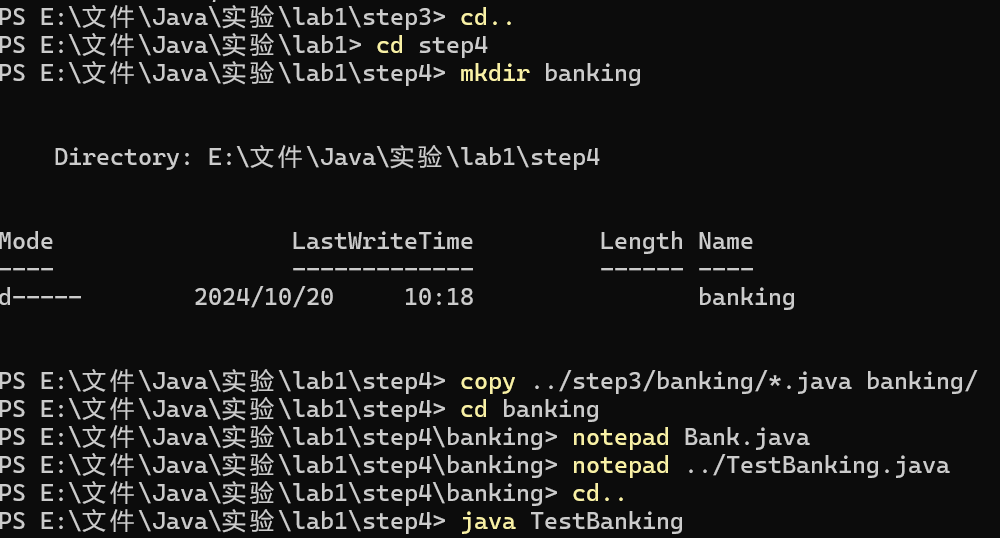
4.2在命令行中，创建 banking 目录并从上一个实验中复制所有文件：

* 1. 进入banking 目录，用记事本创建 Bank.java 文件：

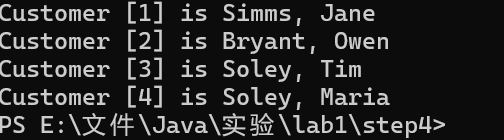
1. **package** banking;
3. **public** **class** Bank {
4. **private** Customer[] customers;
5. **private** **int** numberOfCustomers;
7. // 构造函数初始化客户数组
8. **public** Bank() {
9. customers = **new** Customer[10];  // 数组容量至少为 5
10. numberOfCustomers = 0;
11. }
13. // 添加客户
14. **public** **void** addCustomer(String f, String l) {
15. customers[numberOfCustomers] = **new** Customer(f, l);
16. numberOfCustomers++;
17. }
19. // 获取客户数量
20. **public** **int** getNumOfCustomers() {
21. **return** numberOfCustomers;
22. }
24. // 获取指定索引的客户
25. **public** Customer getCustomer(**int** index) {
26. **if** (index >= 0 && index < numberOfCustomers) {
27. **return** customers[index];
28. }
29. **return** **null**;
30. }
31. }

4.4修改 TestBanking 类：

1. **import** banking.\*;
3. **public** **class** TestBanking {
5. **public** **static** **void** main(String[] args) {
6. Bank bank = **new** Bank();
8. // 添加客户
9. bank.addCustomer("Jane", "Simms");
10. bank.addCustomer("Owen", "Bryant");
11. bank.addCustomer("Tim", "Soley");
12. bank.addCustomer("Maria", "Soley");
14. // 输出客户信息
15. **for** (**int** i = 0; i < bank.getNumOfCustomers(); i++) {
16. Customer customer = bank.getCustomer(i);
17. System.out.println("Customer [" + (i+1) + "] is " + customer.getLastName() + ", " + customer.getFirstName());
18. }
19. }
20. }

4.5编译和运行程序：返回到 /step4 目录，编译 TestBanking.java 文件：

4.6预期输出：

**三、实验问题思考**

1.为什么编译 TestBanking.java 同时也生成了其他类的类文件？

当你仅编译 TestBanking.java 文件时，编译器会自动查找并编译该类中引用的其他类（例如 Customer 和 Account 类）。这是因为 TestBanking 类依赖于这些类来运行其逻辑。Java 编译器会处理文件的依赖关系，即使你只编译一个源文件，编译器也会自动编译该源文件所依赖的其他类文件。

2.在 /banking 目录下直接运行 javac customer.java 出错的原因分析

引用的 Account 类未找到：Customer 类中引用了 Account 类，然而如果你直接在 /banking 目录下编译 Customer.java，编译器可能无法找到 Account 类的定义，因为它可能未被编译或没有被正确引用。

包声明问题：Java 中的类通常是在包内声明的，例如 package banking;。如果你在 /banking 目录下运行 javac Customer.java，而不使用正确的类路径设置（如 -classpath），编译器可能无法识别 banking 包，从而导致编译失败。

3-classpath 类路径的作用与 JDK 编译器和 JVM 定位类文件的方式

-classpath 类路径的作用：

-classpath（或简写为 -cp）选项用于指定 Java 编译器和虚拟机在编译或运行时搜索类文件的路径。它告诉编译器和 JVM 应该从哪些目录或 JAR 文件中查找所需的类。

JDK 编译器和 JVM 定位类文件的方式：

1. 编译器（javac）定位类文件的顺序：

- 首先检查当前目录或指定的 classpath。

- 如果指定了 -classpath，编译器会按照给定的路径查找类。

- 如果找不到 .class 文件，编译器会查找对应的 .java 源文件，并尝试编译。

2. JVM 定位类文件的顺序：

- 首先检查当前目录或指定的 classpath 中是否有对应的 .class 文件。

- 如果指定了 -classpath，JVM 会按照指定路径查找类文件。

- 如果类文件找不到，JVM 会抛出 ClassNotFoundException。