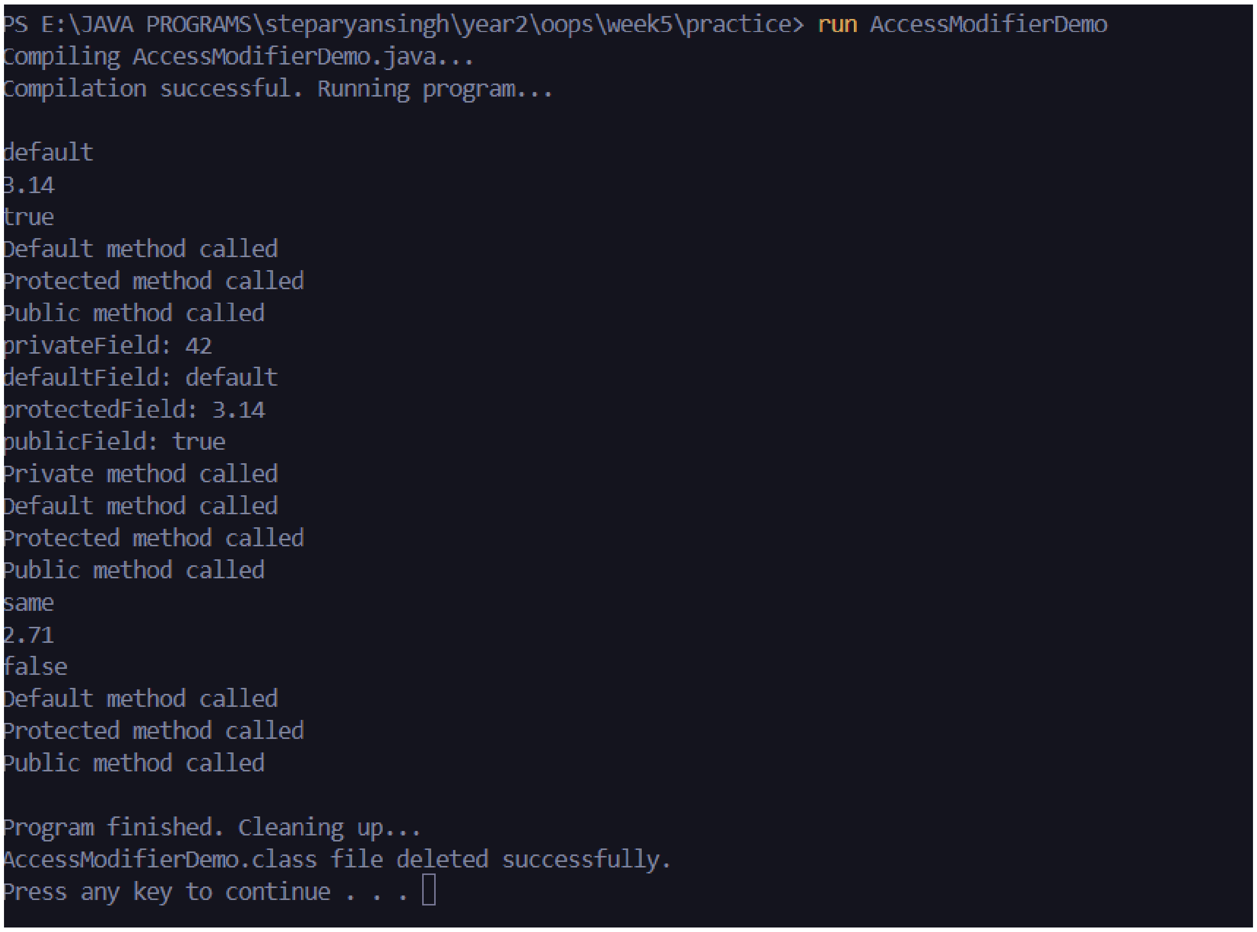
```java
// package com.company.security;

public class AccessModifierDemo {
    // Fields with different access modifiers
    private int privateField;
    String defaultField;
    protected double protectedField;
    public boolean publicField;

    // Constructor to initialize all fields
    public AccessModifierDemo(int privateField, String defaultField, double
protectedField, boolean publicField) {
        this.privateField = privateField;
        this.defaultField = defaultField;
        this.protectedField = protectedField;
        this.publicField = publicField;
    }

    // Methods with matching access levels
    private void privateMethod() {
        System.out.println("Private method called");
    }

    void defaultMethod() {
        System.out.println("Default method called");
    }

    protected void protectedMethod() {
        System.out.println("Protected method called");
    }

    public void publicMethod() {
        System.out.println("Public method called");
    }

    // Public method to test internal access
    public void testInternalAccess() {
        System.out.println("privateField: " + privateField);
        System.out.println("defaultField: " + defaultField);
        System.out.println("protectedField: " + protectedField);
        System.out.println("publicField: " + publicField);

        privateMethod();
        defaultMethod();
        protectedMethod();
        publicMethod();
    }

    public static void main(String[] args) {
        AccessModifierDemo demo = new AccessModifierDemo(42, "default", 3.14,
true);

        // Accessing fields
        // System.out.println(demo.privateField); //  Error: private
        System.out.println(demo.defaultField);    //  Accessible within same
package
        System.out.println(demo.protectedField);  //  Accessible within same
package
        System.out.println(demo.publicField);     //  Accessible everywhere

        // Accessing methods
        // demo.privateMethod(); //  Error: private
        demo.defaultMethod();   //  Accessible within same package
        demo.protectedMethod(); //  Accessible within same package
        demo.publicMethod();    //  Accessible everywhere

        // Internal access test
        demo.testInternalAccess(); //  All members accessible within same class

        // Test same package access
        SamePackageTest.testAccess();
    }
}

// SamePackageTest class - can be in the same file as a non-public class
class SamePackageTest {
    public static void testAccess() {
        AccessModifierDemo demo = new AccessModifierDemo(10, "same", 2.71, false);

        // Accessing fields
        // System.out.println(demo.privateField); //  Error: private
        System.out.println(demo.defaultField);    //  Accessible within same
package
        System.out.println(demo.protectedField);  //  Accessible within same
package
        System.out.println(demo.publicField);     //  Accessible everywhere

        // Accessing methods
        // demo.privateMethod(); //  Error: private
        demo.defaultMethod();   //  Accessible within same package
        demo.protectedMethod(); //  Accessible within same package
        demo.publicMethod();    //  Accessible everywhere
    }
}
```

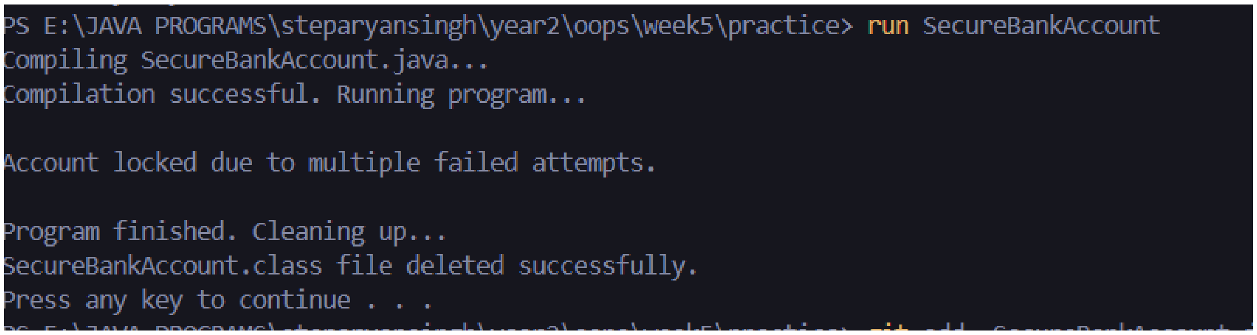


```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week5\practice> run AccessModifierDemo
Compiling AccessModifierDemo.java...
Compilation successful. Running program...

default
3.14
true
Default method called
Protected method called
Public method called
privateField: 42
defaultField: default
protectedField: 3.14
publicField: true
Private method called
Default method called
Protected method called
Public method called
same
2.71
false
Default method called
Protected method called
Public method called

Program finished. Cleaning up...
AccessModifierDemo.class file deleted successfully.
Press any key to continue . . .
```

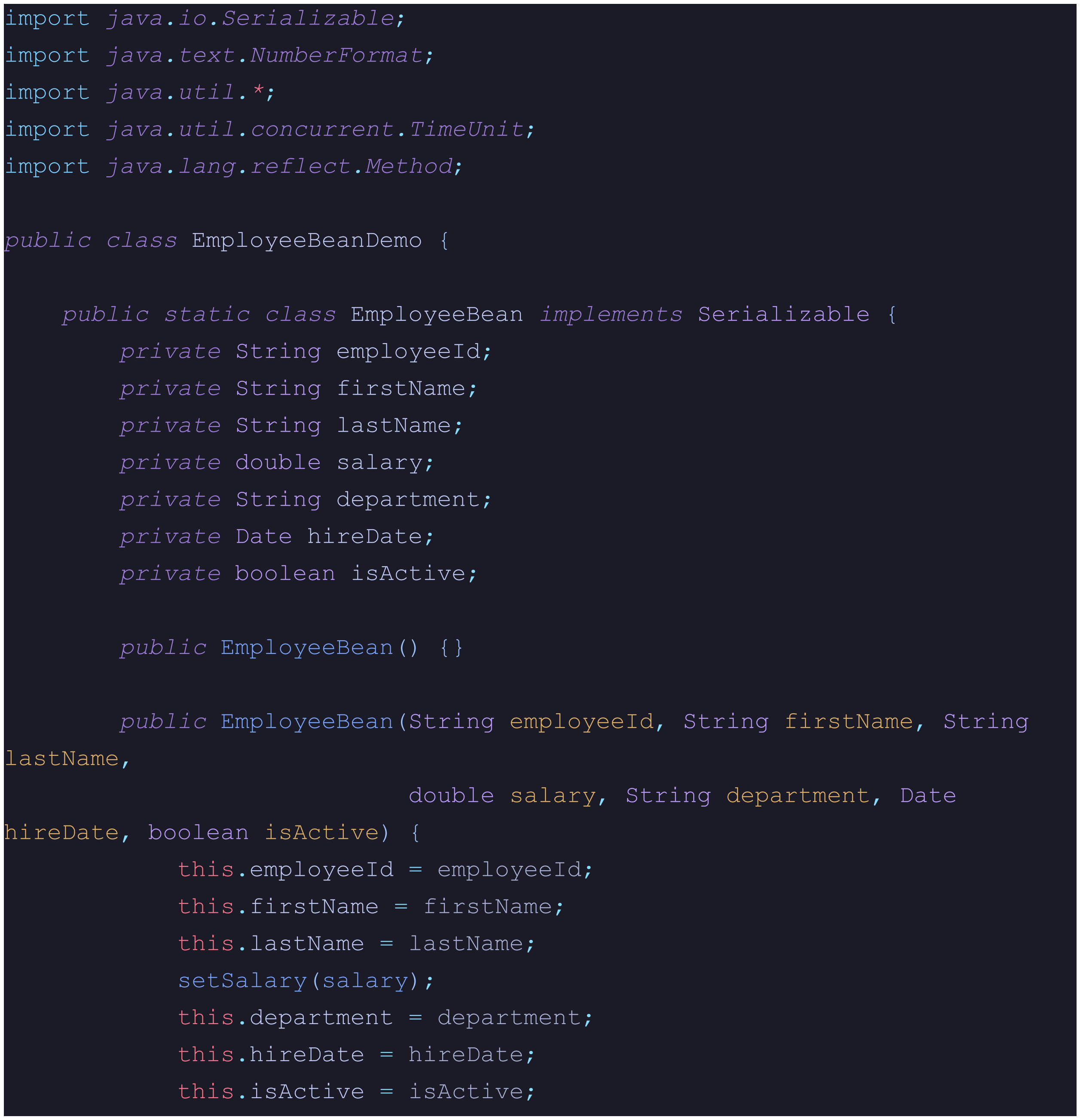
```
// PRACTICE PROBLEM 3: Data Hiding Mastery
// Implementing proper encapsulation with private fields and public
methods

// public class SecureBankAccount {
//     // TODO: Create private fields that should NEVER be accessed
directly:
//     // - accountNumber (String) - read-only after creation
//     // - balance (double) - only modified through controlled methods
//     // - pin (int) - write-only for security
//     // - isLocked (boolean) - internal security state
//     // - failedAttempts (int) - internal security counter

//     // TODO: Create private constants:
//     // - MAX_FAILED_ATTEMPTS (int) = 3
//     // - MIN_BALANCE (double) = 0.0

//     // TODO: Create constructor that takes accountNumber and initial
balance
//     // TODO: Initialize pin to 0 (must be set separately)

//     // TODO: Create PUBLIC methods for controlled access:

//     // Account Info Methods:
//     // - getAccountNumber() - returns account number
//     // - getBalance() - returns current balance (only if not locked)
//     // - isAccountLocked() - returns lock status
// 3



//     // Security Methods:
//     // - setPin(int oldPin, int newPin) - changes PIN if old PIN
correct
//     // - validatePin(int enteredPin) - checks PIN, handles failed
attempts
//     // - unlockAccount(int correctPin) - unlocks if PIN correct

//     // Transaction Methods:
//     // - deposit(double amount, int pin) - adds money if PIN valid
//     // - withdraw(double amount, int pin) - removes money if PIN valid
and sufficient funds
//     // - transfer(SecureBankAccount target, double amount, int pin) -
transfers between accounts

//     // TODO: Create private helper methods:
//     // - lockAccount() - sets isLocked to true
//     // - resetFailedAttempts() - resets counter to 0
//     // - incrementFailedAttempts() - increases counter, locks if needed

//     public static void main(String[] args) {
//         // TODO: Create two SecureBankAccount objects
//          // TODO: Try to access private fields directly (should fail)
//          // TODO: Demonstrate proper usage through public methods:
//          //   - Set PINs for both accounts
//          //   - Make deposits and withdrawals
//          //   - Show security features (account locking)
//          //   - Transfer money between accounts
//          // TODO: Attempt security breaches:
//          //   - Wrong PIN multiple times
//          //   - Withdrawing more than balance
//          //   - Operating on locked account
//      }
// }
    public class SecureBankAccount {
    // Private fields
    private final String accountNumber;
    private double balance;
    private int pin;
    private boolean isLocked;
    private int failedAttempts;

    // Private constants
    private static final int MAX_FAILED_ATTEMPTS = 3;
    private static final double MIN_BALANCE = 0.0;

    // Constructor
    public SecureBankAccount(String accountNumber, double initialBalance)
{
        this.accountNumber = accountNumber;
        this.balance = Math.max(initialBalance, MIN_BALANCE);
        this.pin = 0;
        this.isLocked = false;
        this.failedAttempts = 0;
    }

    // Account Info Methods
    public String getAccountNumber() {
        return accountNumber;
    }

    public double getBalance() {
```

```java
        if (isLocked) {
            System.out.println("Account is locked. Cannot retrieve 
balance.");
            return -1;
        }
        return balance;
    }

    public boolean isAccountLocked() {
        return isLocked;
    }

    // Security Methods
    public boolean setPin(int oldPin, int newPin) {
        if (this.pin == oldPin) {
            this.pin = newPin;
            resetFailedAttempts();
            return true;
        }
        incrementFailedAttempts();
        return false;
    }

    public boolean validatePin(int enteredPin) {
        if (this.pin == enteredPin) {
            resetFailedAttempts();
            return true;
        } else {
            incrementFailedAttempts();
            return false;
        }
    }

    public boolean unlockAccount(int correctPin) {
        if (this.pin == correctPin) {
            isLocked = false;
            resetFailedAttempts();
            return true;
        }
        return false;
    }

    // Transaction Methods
    public boolean deposit(double amount, int pin) {
        if (isLocked || !validatePin(pin)) return false;
        if (amount > 0) {
            balance += amount;
            return true;
        }
        return false;
    }

    public boolean withdraw(double amount, int pin) {
        if (isLocked || !validatePin(pin)) return false;
        if (amount > 0 && balance >= amount) {
            balance -= amount;
            return true;
        }
        System.out.println("Insufficient funds or invalid amount.");
        return false;
    }

    public boolean transfer(SecureBankAccount target, double amount, int
pin) {
        if (withdraw(amount, pin)) {
            return target.deposit(amount, pin);
        }
        return false;
    }

    // Private helper methods
    private void lockAccount() {
        isLocked = true;
        System.out.println("Account locked due to multiple failed
attempts.");
    }

    private void resetFailedAttempts() {
        failedAttempts = 0;
    }
```

```java
    private void incrementFailedAttempts() {
        failedAttempts++;
        if (failedAttempts >= MAX_FAILED_ATTEMPTS) {
            lockAccount();
        }
    }

    // Main method for testing
    public static void main(String[] args) {
        SecureBankAccount acc1 = new SecureBankAccount("ACC123", 5000);
        SecureBankAccount acc2 = new SecureBankAccount("ACC456", 3000);

        // Direct access (should fail)
        // System.out.println(acc1.balance); // ❌ Error: private field

        // Set PINs
        acc1.setPin(0, 1234);
        acc2.setPin(0, 5678);

        // Valid transactions
        acc1.deposit(1000, 1234);
        acc1.withdraw(2000, 1234);
        acc1.transfer(acc2, 500, 1234);

        // Security breach: wrong PIN multiple times
        acc1.validatePin(1111);
        acc1.validatePin(2222);
        acc1.validatePin(3333); // Should lock account

        // Attempt operation on locked account
        acc1.withdraw(100, 1234); // Should fail

        // Unlock account
        acc1.unlockAccount(1234);
        acc1.withdraw(100, 1234); // Should succeed
    }
}
```



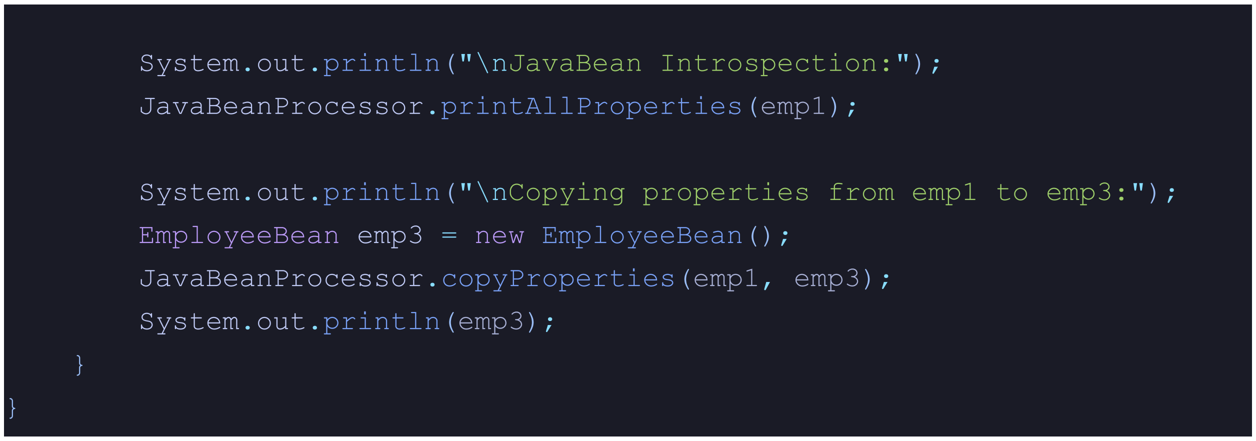
```
PS E:\JAVA PROGRAMS\stepayansingh\year2\oops\week5\practice> run SecureBankAccount
Compiling SecureBankAccount.java...
Compilation successful. Running program...

Account locked due to multiple failed attempts.

Program finished. Cleaning up...
SecureBankAccount.class file deleted successfully.
Press any key to continue . . .
```



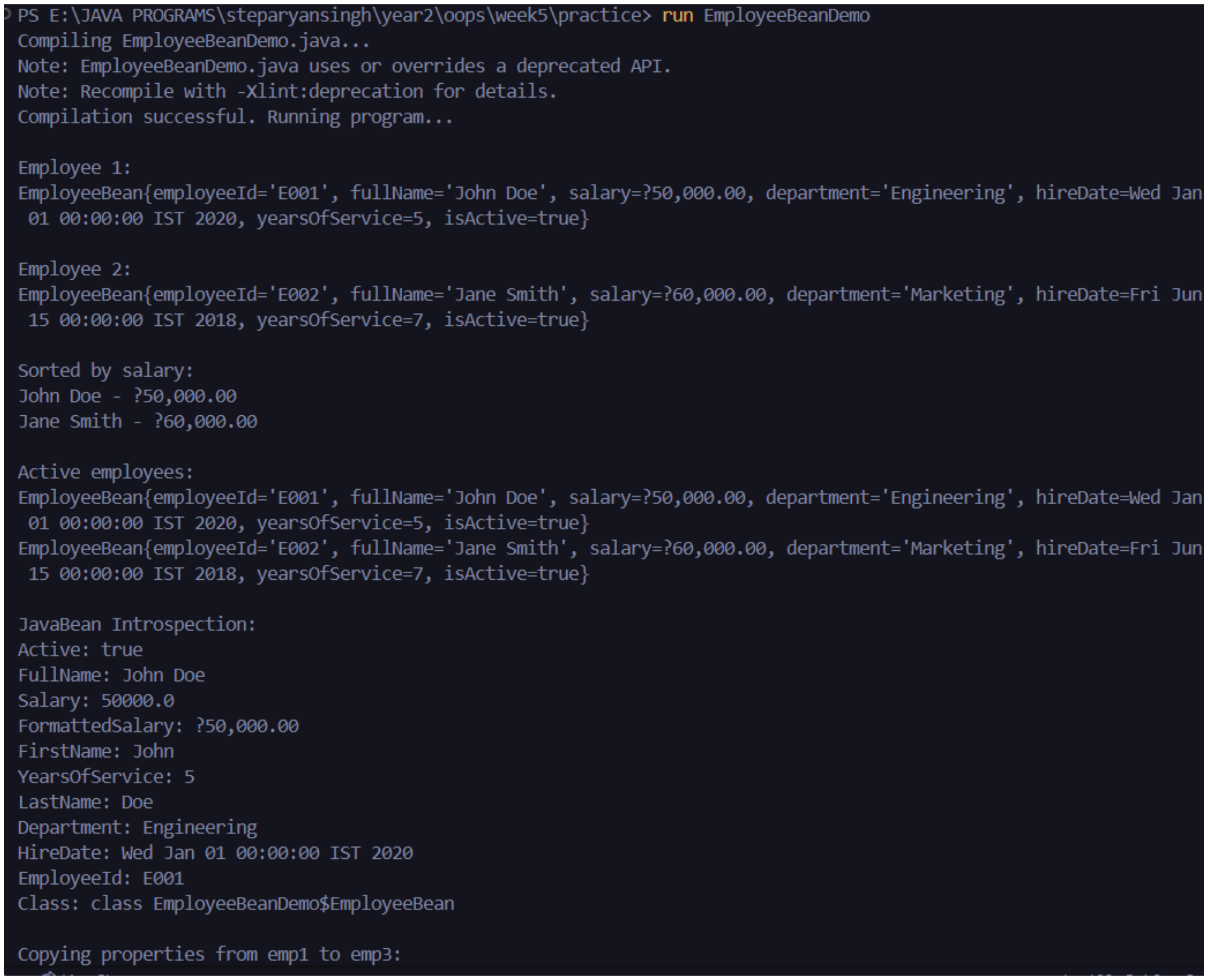
```java
import java.io.Serializable;
import java.text.NumberFormat;
import java.util.*;
import java.util.concurrent.TimeUnit;
import java.lang.reflect.Method;

public class EmployeeBeanDemo {

    public static class EmployeeBean implements Serializable {
        private String employeeId;
        private String firstName;
        private String lastName;
        private double salary;
        private String department;
        private Date hireDate;
        private boolean isActive;

        public EmployeeBean() {}

        public EmployeeBean(String employeeId, String firstName, String
lastName,
                            double salary, String department, Date
hireDate, boolean isActive) {
            this.employeeId = employeeId;
            this.firstName = firstName;
            this.lastName = lastName;
            setSalary(salary);
            this.department = department;
            this.hireDate = hireDate;
            this.isActive = isActive;
        }

        public String getEmployeeId() { return employeeId; }
        public String getFirstName() { return firstName; }
        public String getLastName() { return lastName; }
        public double getSalary() { return salary; }
        public String getDepartment() { return department; }
        public Date getHireDate() { return hireDate; }
        public boolean isActive() { return isActive; }

        public void setEmployeeId(String employeeId) { this.employeeId =
employeeId; }
        public void setFirstName(String firstName) { this.firstName =
firstName; }
        public void setLastName(String lastName) { this.lastName =
lastName; }
        public void setSalary(double salary) {
            if (salary >= 0) {
                this.salary = salary;
            } else {
                throw new IllegalArgumentException("Salary must be
non-negative");
            }
        }
        public void setDepartment(String department) { this.department =
department; }
        public void setHireDate(Date hireDate) { this.hireDate = hireDate;
}
        public void setActive(boolean active) { isActive = active; }

        public String getFullName() {
            return firstName + " " + lastName;
        }

        public int getYearsOfService() {
            if (hireDate == null) return 0;
            long diff = new Date().getTime() - hireDate.getTime();
            return (int) TimeUnit.MILLISECONDS.toDays(diff) / 365;
        }
        public String getFormattedSalary() {
            return NumberFormat.getCurrencyInstance().format(salary);
        }

        public void setFullName(String fullName) {
            String[] parts = fullName.split(" ");
            if (parts.length >= 2) {
                this.firstName = parts[0];
                this.lastName = parts[1];
            } else {
                throw new IllegalArgumentException("Full name must contain
at least first and last name");
            }
        }

        @Override
        public String toString() {
            return "EmployeeBean{" +
                    "employeeId='" + employeeId + '\'' +
                    ", fullName='" + getFullName() + '\'' +
                    ", salary=" + getFormattedSalary() +
                    ", department='" + department + '\'' +
                    ", hireDate=" + hireDate +
                    ", yearsOfService=" + getYearsOfService() +
                    ", isActive=" + isActive +
                    '}';
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (!(o instanceof EmployeeBean)) return false;
            EmployeeBean that = (EmployeeBean) o;
            return Objects.equals(employeeId, that.employeeId);
        }

        @Override
        public int hashCode() {
            return Objects.hash(employeeId);
        }
```

```
    }

    public static class JavaBeanProcessor {
        public static void printAllProperties(EmployeeBean emp) {
            Method[] methods = emp.getClass().getMethods();
            for (Method method : methods) {
                if ((method.getName().startsWith("get") ||
method.getName().startsWith("is"))
                        && method.getParameterCount() == 0) {
                    try {
                        Object value = method.invoke(emp);
                        String propName = method.getName()
                                .replaceFirst("get", "")
                                .replaceFirst("is", "");
                        System.out.println(propName + ": " + value);
                    } catch (Exception e) {
                        System.out.println("Error accessing " +
method.getName());
                    }
                }
            }
        }

        public static void copyProperties(EmployeeBean source,
EmployeeBean target) {
            Method[] methods = source.getClass().getMethods();
            for (Method getter : methods) {
                if ((getter.getName().startsWith("get") ||
getter.getName().startsWith("is"))
                        && getter.getParameterCount() == 0) {
                    try {
                        Object value = getter.invoke(source);
                        String propName = getter.getName()
                                .replaceFirst("get", "")
                                .replaceFirst("is", "");
                        String setterName = "set" + propName;
                        for (Method setter :
target.getClass().getMethods()) {
                            if (setter.getName().equals(setterName)
                                    && setter.getParameterCount() == 1) {
```

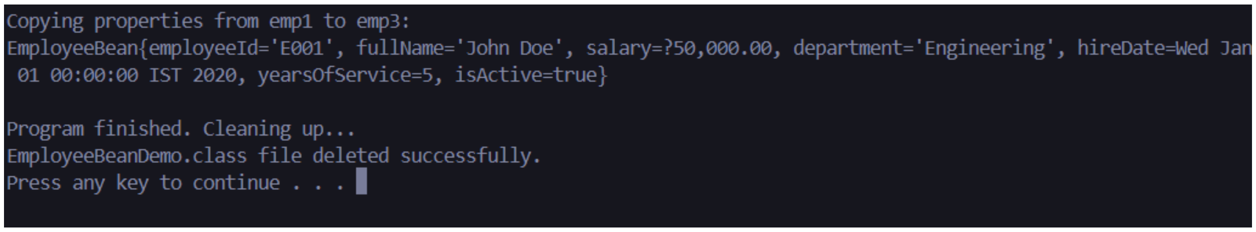
```
                                setter.invoke(target, value);
                            }
                        }
                    } catch (Exception ignored) {}
                }
            }
        }
    }

    public static void main(String[] args) {
        EmployeeBean emp1 = new EmployeeBean();
        emp1.setEmployeeId("E001");
        emp1.setFullName("John Doe");
        emp1.setSalary(50000);
        emp1.setDepartment("Engineering");
        emp1.setHireDate(new Date(120, 0, 1)); // Jan 1, 2020
        emp1.setActive(true);

        EmployeeBean emp2 = new EmployeeBean("E002", "Jane", "Smith",
60000,
                "Marketing", new Date(118, 5, 15), true);

        System.out.println("Employee 1:");
        System.out.println(emp1);
        System.out.println("\nEmployee 2:");
        System.out.println(emp2);

        System.out.println("\nSorted by salary:");
        EmployeeBean[] employees = {emp1, emp2};
        Arrays.sort(employees,
Comparator.comparingDouble(EmployeeBean::getSalary));
        for (EmployeeBean e : employees) {
            System.out.println(e.getFullName() + " - " +
e.getFormattedSalary());
        }

        System.out.println("\nActive employees:");
        Arrays.stream(employees)
                .filter(EmployeeBean::isActive)
                .forEach(System.out::println);
```



```java
        System.out.println("\nJavaBean Introspection:");
        JavaBeanProcessor.printAllProperties(emp1);

        System.out.println("\nCopying properties from emp1 to emp3:");
        EmployeeBean emp3 = new EmployeeBean();
        JavaBeanProcessor.copyProperties(emp1, emp3);
        System.out.println(emp3);
    }
}
```



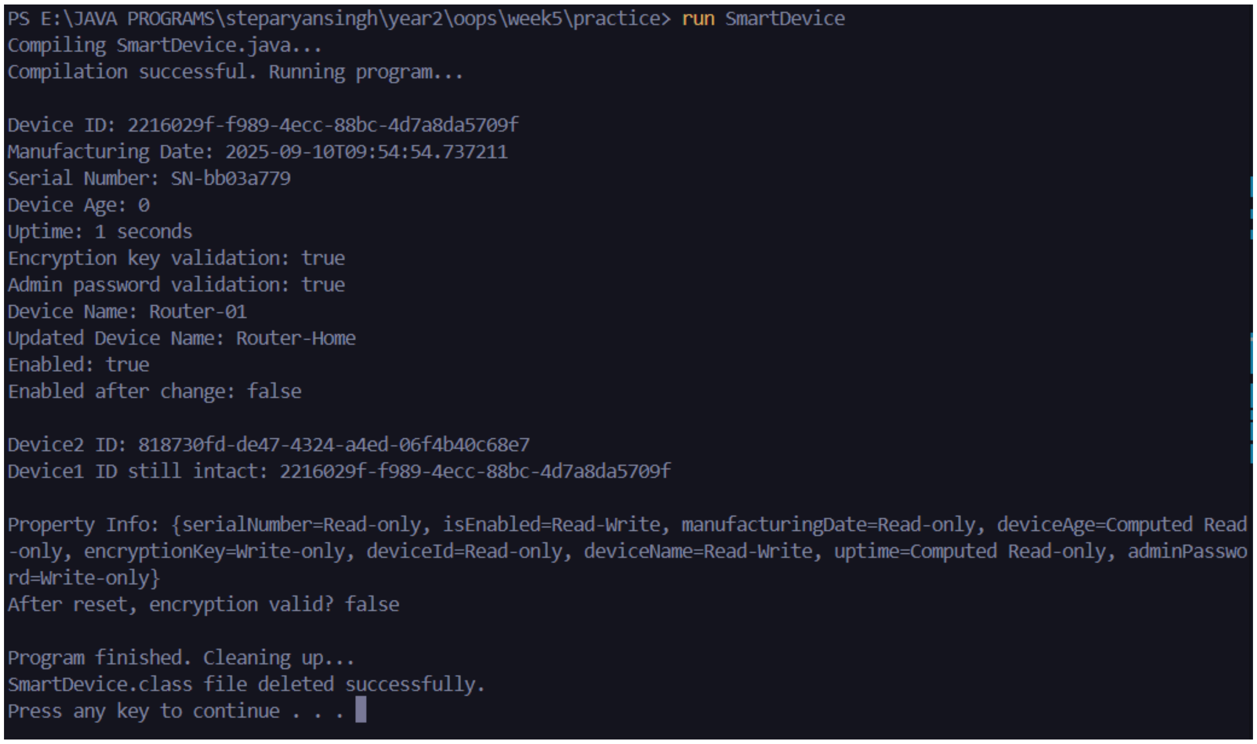
```
PS E:\JAVA PROGRAMS\stepsaryansingh\year2\oops\week5\practice> run EmployeeBeanDemo
Compiling EmployeeBeanDemo.java...
Note: EmployeeBeanDemo.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Compilation successful. Running program...

Employee 1:
EmployeeBean{employeeId='E001', fullName='John Doe', salary=?50,000.00, department='Engineering', hireDate=Wed Jan
 01 00:00:00 IST 2020, yearsOfService=5, isActive=true}

Employee 2:
EmployeeBean{employeeId='E002', fullName='Jane Smith', salary=?60,000.00, department='Marketing', hireDate=Fri Jun
 15 00:00:00 IST 2018, yearsOfService=7, isActive=true}

Sorted by salary:
John Doe - ?50,000.00
Jane Smith - ?60,000.00

Active employees:
EmployeeBean{employeeId='E001', fullName='John Doe', salary=?50,000.00, department='Engineering', hireDate=Wed Jan
 01 00:00:00 IST 2020, yearsOfService=5, isActive=true}
EmployeeBean{employeeId='E002', fullName='Jane Smith', salary=?60,000.00, department='Marketing', hireDate=Fri Jun
 15 00:00:00 IST 2018, yearsOfService=7, isActive=true}

JavaBean Introspection:
Active: true
FullName: John Doe
Salary: 50000.0
FormattedSalary: ?50,000.00
FirstName: John
YearsOfService: 5
LastName: Doe
Department: Engineering
HireDate: Wed Jan 01 00:00:00 IST 2020
EmployeeId: E001
Class: class EmployeeBeanDemo$EmployeeBean

Copying properties from emp1 to emp3:
```



```
Copying properties from emp1 to emp3:
EmployeeBean{employeeId='E001', fullName='John Doe', salary=?50,000.00, department='Engineering', hireDate=Wed Jan
 01 00:00:00 IST 2020, yearsOfService=5, isActive=true}

Program finished. Cleaning up...
EmployeeBeanDemo.class file deleted successfully.
Press any key to continue . . .
```

```java
import java.time.Duration;
import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

public class SmartDevice {
    // === Read-only properties ===
    private final String deviceId;
    private final LocalDateTime manufacturingDate;
    private final String serialNumber;

    // === Write-only properties (stored as hashes) ===
    private int hashedEncryptionKey;
    private int hashedAdminPassword;

    // === Read-write properties ===
    private String deviceName;
    private boolean enabled;

    // === Internal state ===
    private final LocalDateTime startupTime;

    // === Constructor ===
    public SmartDevice(String deviceName) {
        this.deviceId = UUID.randomUUID().toString();
        this.manufacturingDate = LocalDateTime.now();
        this.serialNumber = "SN-" +
UUID.randomUUID().toString().substring(0, 8);
        this.startupTime = LocalDateTime.now();
        this.deviceName = deviceName;
        this.enabled = true; // default enabled
    }

    // === Read-Only Property Methods ===
    public String getDeviceId() {
        return deviceId;
    }

    public LocalDateTime getManufacturingDate() {
        return manufacturingDate;
    }

    public String getSerialNumber() {
        return serialNumber;
    }

    public long getUptime() {
        return Duration.between(startupTime,
LocalDateTime.now()).toSeconds();
    }

    public int getDeviceAge() {
        return LocalDateTime.now().getYear() -
manufacturingDate.getYear();
    }

    // === Write-Only Property Methods ===
    public void setEncryptionKey(String key) {
        if (key == null || key.length() < 8) {
            throw new IllegalArgumentException("Encryption key must be at
least 8 characters long.");
        }
        this.hashedEncryptionKey = key.hashCode();
    }

    public void setAdminPassword(String password) {
        if (password == null || password.length() < 10 ||
!password.matches(".*\\d.*")) {
            throw new IllegalArgumentException("Password must be at least
10 chars and contain a digit.");
        }
        this.hashedAdminPassword = password.hashCode();
    }

    public boolean validateEncryptionKey(String key) {
        return key != null && key.hashCode() == this.hashedEncryptionKey;
    }

    public boolean validateAdminPassword(String password) {
        return password != null && password.hashCode() ==
this.hashedAdminPassword;
    }

    // === Read-Write Property Methods ===
    public String getDeviceName() {
        return deviceName;
    }

    public void setDeviceName(String name) {
        this.deviceName = name;
    }

    public boolean isEnabled() {
        return enabled;
    }

    public void setEnabled(boolean enabled) {
        this.enabled = enabled;
    }

    // === Utility Methods ===
    public Map<String, String> getPropertyInfo() {
        Map<String, String> info = new HashMap<>();
        info.put("deviceId", "Read-only");
        info.put("manufacturingDate", "Read-only");
        info.put("serialNumber", "Read-only");
        info.put("uptime", "Computed Read-only");
        info.put("deviceAge", "Computed Read-only");
        info.put("encryptionKey", "Write-only");
        info.put("adminPassword", "Write-only");
        info.put("deviceName", "Read-Write");
        info.put("isEnabled", "Read-Write");
        return info;
    }

    public void resetDevice() {
        this.hashedEncryptionKey = 0;
        this.hashedAdminPassword = 0;
        this.enabled = false; // simulate device reset
    }

    // === Demo Main Method ===
    public static void main(String[] args) {
        SmartDevice device1 = new SmartDevice("Router-01");

        // Read-only properties
        System.out.println("Device ID: " + device1.getDeviceId());
        System.out.println("Manufacturing Date: " +
device1.getManufacturingDate());
        System.out.println("Serial Number: " + device1.getSerialNumber());
        System.out.println("Device Age: " + device1.getDeviceAge());

        // Wait a bit to show uptime
        try { Thread.sleep(1000); } catch (InterruptedException e) {}
        System.out.println("Uptime: " + device1.getUptime() + " seconds");

        // Write-only properties
        device1.setEncryptionKey("SuperSecureKey");
        device1.setAdminPassword("AdminPass123");

        System.out.println("Encryption key validation: " +
device1.validateEncryptionKey("SuperSecureKey"));
        System.out.println("Admin password validation: " +
device1.validateAdminPassword("AdminPass123"));

        // Read-Write properties
        System.out.println("Device Name: " + device1.getDeviceName());
        device1.setDeviceName("Router-Home");
        System.out.println("Updated Device Name: " +
device1.getDeviceName());

        System.out.println("Enabled: " + device1.isEnabled());
        device1.setEnabled(false);
        System.out.println("Enabled after change: " +
device1.isEnabled());

        // Multiple devices
        SmartDevice device2 = new SmartDevice("Switch-01");
        System.out.println("\nDevice2 ID: " + device2.getDeviceId());
```

```java
        System.out.println("Device1 ID still intact: " +
device1.getDeviceId());

        // Property info
        System.out.println("\nProperty Info: " +
device1.getPropertyInfo());

        // Reset device
        device1.resetDevice();
        System.out.println("After reset, encryption valid? " +
device1.validateEncryptionKey("SuperSecureKey"));
    }
}
```



```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week5\practice> run SmartDevice
Compiling SmartDevice.java...
Compilation successful. Running program...

Device ID: 2216029f-f989-4ecc-88bc-4d7a8da5709f
Manufacturing Date: 2025-09-10T09:54:54.737211
Serial Number: SN-bb03a779
Device Age: 0
Uptime: 1 seconds
Encryption key validation: true
Admin password validation: true
Device Name: Router-01
Updated Device Name: Router-Home
Enabled: true
Enabled after change: false

Device2 ID: 818730fd-de47-4324-a4ed-06f4b40c68e7
Device1 ID still intact: 2216029f-f989-4ecc-88bc-4d7a8da5709f

Property Info: {serialNumber=Read-only, isEnabled=Read-Write, manufacturingDate=Read-only, deviceAge=Computed Read
-only, encryptionKey=Write-only, deviceId=Read-only, deviceName=Read-Write, uptime=Computed Read-only, adminPasswo
rd=Write-only}
After reset, encryption valid? false

Program finished. Cleaning up...
SmartDevice.class file deleted successfully.
Press any key to continue . . .
```