Weak 5 PRACTICE PROBLEM

PRACTICE PROBLEM 1: Access Modifiers - The Four
Levels of Security
Understanding private, default, protected, public modifiers

```
// File: AccessModifierDemo.java
package com.company.security;
public class AccessModifierDemo {
// TODO: Create four different fields with different access modifiers:
// - privateField (int) - only accessible within this class
// - defaultField (String) - accessible within same package
// - protectedField (double) - accessible in package + subclasses
// - publicField (boolean) - accessible everywhere
// TODO: Create four methods with matching access levels:
// - privateMethod() - prints "Private method called"
// - defaultMethod() - prints "Default method called"
// - protectedMethod() - prints "Protected method called"
// - publicMethod() - prints "Public method called"
// TODO: Create a constructor that initializes all fields
// TODO: Create a public method testInternalAccess() that:
// - Accesses and prints all four fields
// - Calls all four methods
// - Demonstrates that private members are accessible within same class
public static void main(String[] args) {
// TODO: Create an AccessModifierDemo object
// TODO: Try to access each field and method
// TODO: Document in comments which ones work and which cause errors
// TODO: Call testInternalAccess() to show internal accessibility
}
}
```

```
// TODO: Create a second class in the SAME package:
class SamePackageTest {
public static void testAccess() {
// TODO: Create AccessModifierDemo object
// TODO: Try accessing each field and method
// TODO: Document which access modifiers work within same package
}
}
```

```
package com.company.security;
```

```java
public class AccessModifierDemo {
    // ===== Fields with different access modifiers =====
    private int privateField;          // Only inside this class
    String defaultField;               // Package-private (default): same
package
    protected double protectedField;  // Same package + subclasses
    public boolean publicField;       // Everywhere

    // ===== 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");
    }

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

    // ===== Public method to show internal access =====
    public void testInternalAccess() {
        System.out.println("---- Inside AccessModifierDemo ----");
        // All fields are accessible inside the same class
        System.out.println("privateField   = " + privateField);
        System.out.println("defaultField   = " + defaultField);
        System.out.println("protectedField = " + protectedField);
        System.out.println("publicField    = " + publicField);
        // All methods are accessible inside the same class
        privateMethod();
        defaultMethod();
        protectedMethod();
        publicMethod();
    }

    public static void main(String[] args) {
        AccessModifierDemo demo =
            new AccessModifierDemo(10, "Hello", 99.9, true);

        // ---- Access from SAME CLASS ----
        // All work because we're inside AccessModifierDemo
        System.out.println("Accessing from main (same class):");
        System.out.println(demo.privateField);
        System.out.println(demo.defaultField);
        System.out.println(demo.protectedField);
        System.out.println(demo.publicField);

        demo.privateMethod();
        demo.defaultMethod();
        demo.protectedMethod();
        demo.publicMethod();


        demo.testInternalAccess();
    }
}

// ===== Second class in the SAME package =====
class SamePackageTest {
    public static void testAccess() {
        AccessModifierDemo demo =
            new AccessModifierDemo(20, "World", 55.5, false);

        System.out.println("---- Inside SamePackageTest ----");

        // Fields:
        // System.out.println(demo.privateField);
        System.out.println(demo.defaultField);      default/package-private
works
        System.out.println(demo.protectedField);
        System.out.println(demo.publicField);
        demo.defaultMethod();
        demo.protectedMethod();
        demo.publicMethod();
    }
}
```

🛠️ PRACTICE PROBLEM 2: Cross-Package Visibility
Rules
Testing access modifiers across different packages
// File: com/company/main/PackageTestMain.java
package com.company.main;
// TODO: Import the AccessModifierDemo class from com.company.security
public class PackageTestMain {
public static void main(String[] args) {
// TODO: Create AccessModifierDemo object
// TODO: Attempt to access each field and method
// TODO: Document which access modifiers work across packages
// TODO: Explain why certain accesses fail
}
}
// TODO: Create a subclass in different package:
// File: com/company/extended/ExtendedDemo.
package com.company.extended;
// TODO: Import AccessModifierDemo
// TODO: Create class ExtendedDemo that extends AccessModifierDemo
public class ExtendedDemo extends AccessModifierDemo {
// TODO: Create constructor that calls super constructor
public void testInheritedAccess() {
// TODO: Try accessing inherited fields with different modifiers

2

// TODO: Try calling inherited methods with different modifiers
// TODO: Document which protected members are accessible
// TODO: Show that private members are NOT inherited
}
// TODO: Override protected method from parent class
public static void main(String[] args) {
// TODO: Test inheritance access rules
// TODO: Create both parent and child objects

// TODO: Compare what each can access

```java
package com.company.extended;

import com.company.security.AccessModifierDemo;

public class ExtendedDemo extends AccessModifierDemo {
    public ExtendedDemo(int p, String d, double pr, boolean pub) {
        super(p, d, pr, pub);
    }
    public void testInheritedAccess() {
        System.out.println(protectedField);
        System.out.println(publicField);
        protectedMethod();
        publicMethod();
    }
    @Override
    protected void protectedMethod() {
        System.out.println("Protected method overridden in ExtendedDemo");
    }
    public static void main(String[] args) {
        ExtendedDemo child = new ExtendedDemo(42, "child", 9.81, false);
        child.testInheritedAccess();
        AccessModifierDemo parent = new AccessModifierDemo(10, "parent",
7.77, true);
        System.out.println(parent.publicField);
        parent.publicMethod();
    }
}
```

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

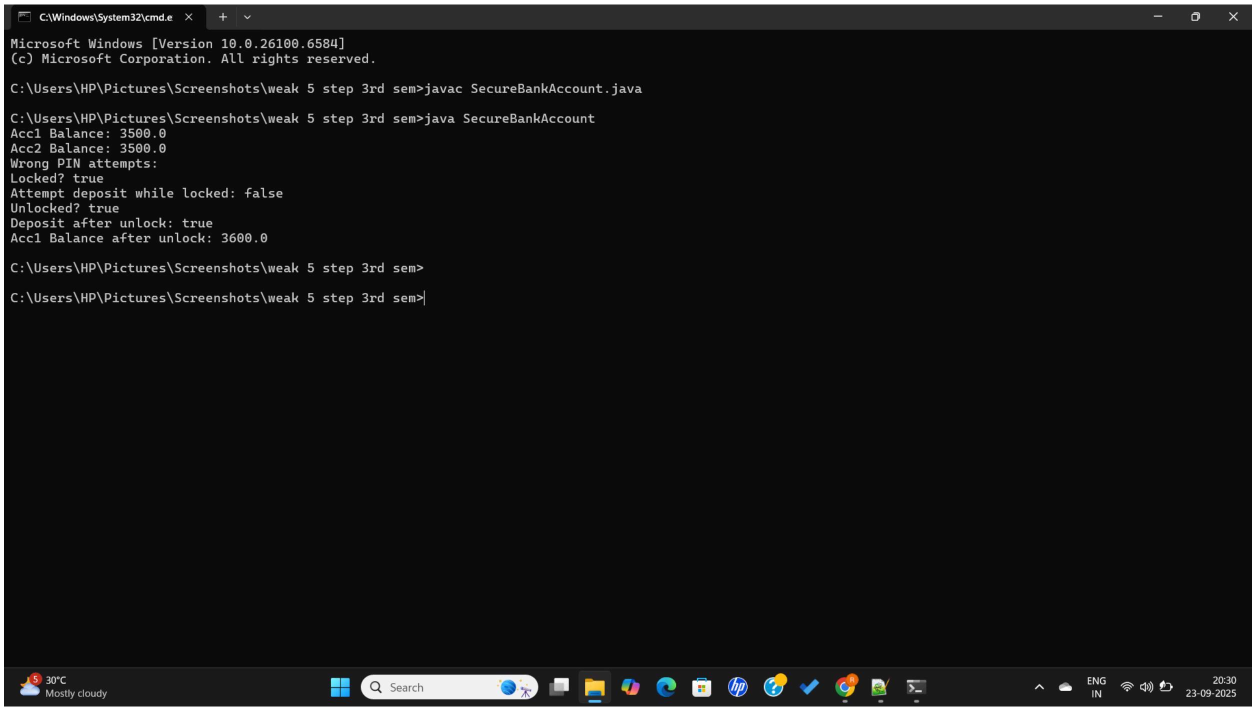
public class SecureBankAccount {
// TODO: Create private fields that should NEVER be accessed directly:
// - accountNumber (String) - read-only after creation
// - balance (double) - only modified through controlled methods
// - pin (int) - write-only for security
// - isLocked (boolean) - internal security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

```java
public class SecureBankAccount {
    private final String accountNumber;
    private double balance;
    private int pin;
    private boolean isLocked;
    private int failedAttempts;
    private static final int MAX_FAILED_ATTEMPTS = 3;
    private static final double MIN_BALANCE = 0.0;

    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;
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public double getBalance() {
        if (!isLocked) {
            return balance;
        }
        throw new IllegalStateException("Account is locked");
    }

    public boolean isAccountLocked() {
        return isLocked;
    }

    public boolean setPin(int oldPin, int newPin) {
        if (pin == oldPin) {
            pin = newPin;
            return true;
        }
        return false;
    }

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

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

    public boolean deposit(double amount, int enteredPin) {
        if (amount <= 0) return false;
        if (validatePin(enteredPin)) {
            balance += amount;
            return true;
        }
        return false;
    }

    public boolean withdraw(double amount, int enteredPin) {
        if (amount <= 0) return false;
        if (validatePin(enteredPin) && balance - amount >= MIN_BALANCE) {
            balance -= amount;
            return true;
        }
        return false;
    }

    public boolean transfer(SecureBankAccount target, double amount, int
enteredPin) {
        if (target == null) return false;
        if (withdraw(amount, enteredPin)) {
            target.balance += amount;
            return true;
        }
        return false;
    }

    private void lockAccount() {
        isLocked = true;
    }
    private void resetFailedAttempts() {
        failedAttempts = 0;
    }

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

    public static void main(String[] args) {
        SecureBankAccount acc1 = new SecureBankAccount("AC001", 5000);
        SecureBankAccount acc2 = new SecureBankAccount("AC002", 3000);

        acc1.setPin(0, 1234);
        acc2.setPin(0, 4321);

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

        System.out.println("Acc1 Balance: " + acc1.getBalance());
        System.out.println("Acc2 Balance: " + acc2.getBalance());

        System.out.println("Wrong PIN attempts:");
        acc1.validatePin(1111);
        acc1.validatePin(1111);
        acc1.validatePin(1111);

        System.out.println("Locked? " + acc1.isAccountLocked());
        System.out.println("Attempt deposit while locked: " +
acc1.deposit(100, 1234));
        acc1.unlockAccount(1234);
        System.out.println("Unlocked? " + !acc1.isAccountLocked());
        System.out.println("Deposit after unlock: " + acc1.deposit(100,
1234));
        System.out.println("Acc1 Balance after unlock: " +
acc1.getBalance());
    }
}
```



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP\Pictures\Screenshots\weak 5 step 3rd sem>javac SecureBankAccount.java

C:\Users\HP\Pictures\Screenshots\weak 5 step 3rd sem>java SecureBankAccount
Acc1 Balance: 3500.0
Acc2 Balance: 3500.0
Wrong PIN attempts:
Locked? true
Attempt deposit while locked: false
Unlocked? true
Deposit after unlock: true
Acc1 Balance after unlock: 3600.0

C:\Users\HP\Pictures\Screenshots\weak 5 step 3rd sem>

C:\Users\HP\Pictures\Screenshots\weak 5 step 3rd sem>
```

🛠️ PRACTICE PROBLEM 4: JavaBean Standards
Implementation
Creating professional JavaBean-compliant classes

```
import java.io.Serializable;
public class EmployeeBean implements Serializable {

4

// TODO: Create private fields following JavaBean conventions:
// - employeeId (String)
// - firstName (String)
// - lastName (String)
// - salary (double)
// - department (String)
// - hireDate (java.util.Date)
// - isActive (boolean)
// TODO: Create default no-argument constructor (JavaBean requirement)
// TODO: Create parameterized constructor for convenience
// TODO: Generate standard JavaBean getter methods:
// - getEmployeeId(), getFirstName(), getLastName(), etc.
// - Follow naming convention: get + PropertyName
// - For boolean: isActive() instead of getIsActive()
// TODO: Generate standard JavaBean setter methods:
// - setEmployeeId(String id), setFirstName(String name), etc.
```

// - Follow naming convention: set + PropertyName
// - Include validation where appropriate
// TODO: Create computed properties (getters without corresponding fields):
// - getFullName() - returns firstName + " " + lastName
// - getYearsOfService() - calculates years since hireDate
// - getFormattedSalary() - returns salary with currency formatting
// TODO: Create derived properties with validation:
// - setFullName(String fullName) - splits into firstName/lastName
// - setSalary(double salary) - validates positive amount
// TODO: Override toString() to display all properties
// TODO: Override equals() and hashCode() based on employeeId
public static void main(String[] args) {
// TODO: Create EmployeeBean using default constructor + setters
// TODO: Create EmployeeBean using parameterized constructor
// TODO: Demonstrate all getter methods
// TODO: Test computed properties
// TODO: Test validation in setter methods
// TODO: Show JavaBean in action with collections (sorting, searching)

5

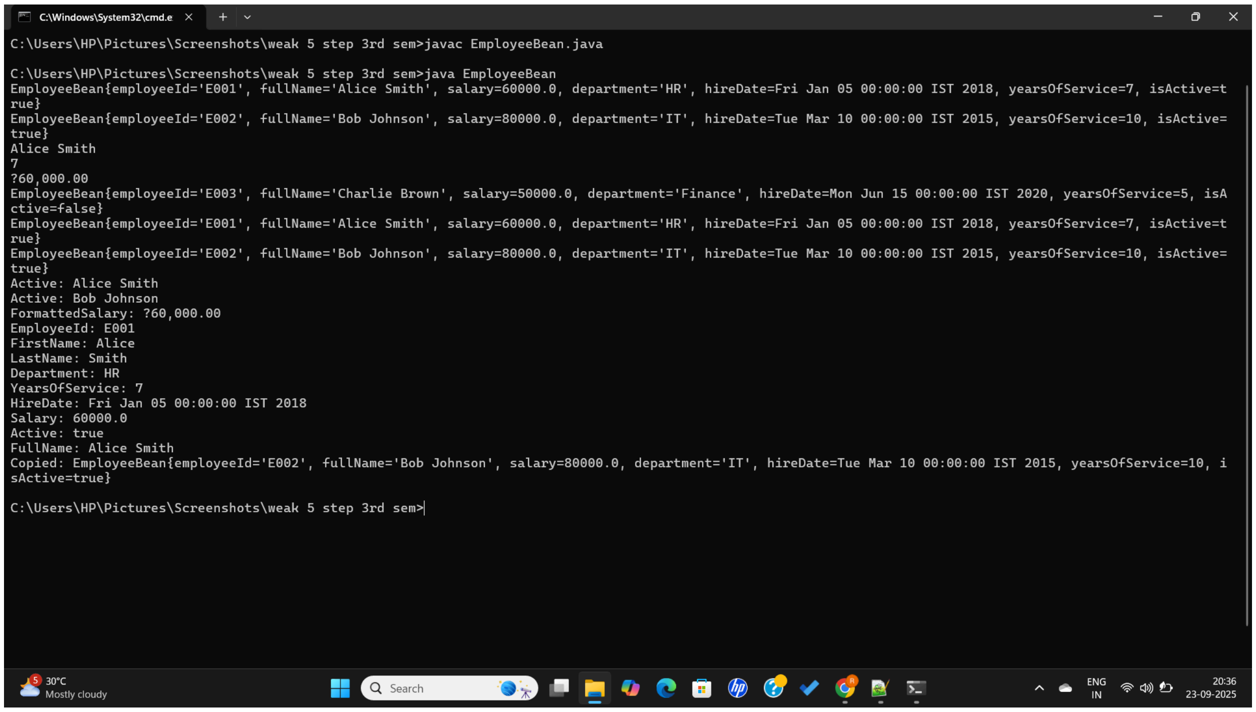
// TODO: Create an array of EmployeeBeans and demonstrate:
// - Sorting by salary using computed properties
// - Filtering active employees
// - Bulk operations using JavaBean conventions
}
}
// TODO: Create a JavaBean utility class:
class JavaBeanProcessor {
// TODO: Create static method printAllProperties(EmployeeBean emp)
// - Uses reflection to find all getter methods
// - Calls each getter and prints property name and value
// - Demonstrates JavaBean introspection capabilities
// TODO: Create static method copyProperties(EmployeeBean source, EmployeeBean target)
// - Uses reflection to copy all properties from source to target
// - Demonstrates JavaBean framework integration potential

```
import java.io.Serializable;
import java.text.NumberFormat;
import java.time.LocalDate;
import java.time.ZoneId;
import java.time.temporal.ChronoUnit;
import java.util.*;
import java.lang.reflect.*;

public 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 void setEmployeeId(String employeeId) { this.employeeId =
employeeId; }

    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName =
firstName; }

    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }

    public double getSalary() { return salary; }
    public void setSalary(double salary) {
        if (salary < 0) throw new IllegalArgumentException("Salary must be
positive");
```

```java
        this.salary = salary;
    }

    public String getDepartment() { return department; }
    public void setDepartment(String department) { this.department =
department; }

    public Date getHireDate() { return hireDate; }
    public void setHireDate(Date hireDate) { this.hireDate = hireDate; }

    public boolean isActive() { return isActive; }
    public void setActive(boolean active) { this.isActive = active; }

    public String getFullName() { return (firstName != null ? firstName :
"") + " " + (lastName != null ? lastName : ""); }

    public long getYearsOfService() {
        if (hireDate == null) return 0;
        LocalDate hire =
hireDate.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
        return ChronoUnit.YEARS.between(hire, LocalDate.now());
    }

    public String getFormattedSalary() {
        return NumberFormat.getCurrencyInstance().format(salary);
    }

    public void setFullName(String fullName) {
        if (fullName == null || fullName.trim().isEmpty()) return;
        String[] parts = fullName.trim().split("\\s+", 2);
        firstName = parts[0];
        lastName = parts.length > 1 ? parts[1] : "";
    }

    @Override
    public String toString() {
        return "EmployeeBean{" +
                "employeeId='" + employeeId + '\'' +
                ", fullName='" + getFullName() + '\'' +
                ", salary=" + salary +
                ", 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 void main(String[] args) {
        EmployeeBean e1 = new EmployeeBean();
        e1.setEmployeeId("E001");
        e1.setFirstName("Alice");
        e1.setLastName("Smith");
        e1.setSalary(60000);
        e1.setDepartment("HR");
        e1.setHireDate(new GregorianCalendar(2018, Calendar.JANUARY,
5).getTime());
        e1.setActive(true);

        EmployeeBean e2 = new EmployeeBean("E002", "Bob", "Johnson", 80000,
                "IT", new GregorianCalendar(2015, Calendar.MARCH,
10).getTime(), true);

        System.out.println(e1);
        System.out.println(e2);
        System.out.println(e1.getFullName());
        System.out.println(e1.getYearsOfService());
        System.out.println(e1.getFormattedSalary());

        EmployeeBean e3 = new EmployeeBean("E003", "Charlie", "Brown",
50000,
                "Finance", new GregorianCalendar(2020, Calendar.JUNE,
15).getTime(), false);
```

```
        List<EmployeeBean> employees = Arrays.asList(e1, e2, e3);

employees.stream().sorted(Comparator.comparingDouble(EmployeeBean::getSalar
y))
                .forEach(System.out::println);
        employees.stream().filter(EmployeeBean::isActive)
                .forEach(emp -> System.out.println("Active: " +
emp.getFullName()));

        JavaBeanProcessor.printAllProperties(e1);
        EmployeeBean copy = new EmployeeBean();
        JavaBeanProcessor.copyProperties(e2, copy);
        System.out.println("Copied: " + copy);
    }
}

class JavaBeanProcessor {
    public static void printAllProperties(EmployeeBean emp) {
        for (Method m : EmployeeBean.class.getMethods()) {
            if ((m.getName().startsWith("get") ||
m.getName().startsWith("is"))
                    && m.getParameterCount() == 0 &&
!m.getName().equals("getClass")) {
                try {
                    Object val = m.invoke(emp);
                    String name = m.getName().startsWith("get")
                            ? m.getName().substring(3)
                            : m.getName().substring(2);
                    System.out.println(name + ": " + val);
                } catch (Exception ignored) {}
            }
        }
    }

    public static void copyProperties(EmployeeBean source, EmployeeBean
target) {
        for (Method m : EmployeeBean.class.getMethods()) {
            if ((m.getName().startsWith("get") ||
m.getName().startsWith("is"))
                    && m.getParameterCount() == 0 &&
!m.getName().equals("getClass")) {
                try {
                    Object val = m.invoke(source);
```

```java
                    String base = m.getName().startsWith("get")
                        ? m.getName().substring(3)
                        : m.getName().substring(2);
                    try {
                        Method setter = EmployeeBean.class.getMethod("set"
+ base, m.getReturnType());
                        setter.invoke(target, val);
                    } catch (NoSuchMethodException ignored) {}
                } catch (Exception ignored) {}
            }
        }
    }
}
```



```
C:\Users\HP\Pictures\Screenshots\weak 5 step 3rd sem>javac EmployeeBean.java

C:\Users\HP\Pictures\Screenshots\weak 5 step 3rd sem>java EmployeeBean
EmployeeBean{employeeId='E001', fullName='Alice Smith', salary=60000.0, department='HR', hireDate=Fri Jan 05 00:00:00 IST 2018, yearsOfService=7, isActive=t
rue}
EmployeeBean{employeeId='E002', fullName='Bob Johnson', salary=80000.0, department='IT', hireDate=Tue Mar 10 00:00:00 IST 2015, yearsOfService=10, isActive=
true}
Alice Smith
7
?60,000.00
EmployeeBean{employeeId='E003', fullName='Charlie Brown', salary=50000.0, department='Finance', hireDate=Mon Jun 15 00:00:00 IST 2020, yearsOfService=5, isA
ctive=false}
EmployeeBean{employeeId='E001', fullName='Alice Smith', salary=60000.0, department='HR', hireDate=Fri Jan 05 00:00:00 IST 2018, yearsOfService=7, isActive=t
rue}
EmployeeBean{employeeId='E002', fullName='Bob Johnson', salary=80000.0, department='IT', hireDate=Tue Mar 10 00:00:00 IST 2015, yearsOfService=10, isActive=
true}
Active: Alice Smith
Active: Bob Johnson
FormattedSalary: ?60,000.00
EmployeeId: E001
FirstName: Alice
LastName: Smith
Department: HR
YearsOfService: 7
HireDate: Fri Jan 05 00:00:00 IST 2018
Salary: 60000.0
Active: true
FullName: Alice Smith
Copied: EmployeeBean{employeeId='E002', fullName='Bob Johnson', salary=80000.0, department='IT', hireDate=Tue Mar 10 00:00:00 IST 2015, yearsOfService=10, i
sActive=true}

C:\Users\HP\Pictures\Screenshots\weak 5 step 3rd sem>
```