Software Re-Engineering

Lecture: 03

Dr. Imran Ali
School of Computing- Software Engineering
FAST-National University of Computer & Emerging Sciences, Karachi

Sequence [Todays Agenda]

Content of Lecture

- Reengineering Code
- Code to Class Diagram

Refactoring Code: Example 1

```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int z = 15;

        if (x > y) {
            System.out.println("x is greater");
        } else {
            System.out.println("x is smaller");
        }

        if (y > z) {
            System.out.println("y is greater");
        } else {
            System.out.println("y is smaller");
        }

        if (x > z) {
            System.out.println("x is greater");
        } else {
            System.out.println("x is smaller");
        }
    }
}
```

Refactoring Code: Solution

```
public class Main {
    public static void main(String[] args) {
        compare(5, 10);
        compare(10, 15);
        compare(5, 15);
    }

    private static void compare(int a, int b) {
        if (a > b) {
            System.out.println(a + " is greater");
        } else {
            System.out.println(a + " is smaller");
        }
    }
}
```

Refactoring Code: Explanation

- Code Duplication Removed:

The logic to compare two values was repeated three times. In the refactored version, helper method compare(int a, int b) was created to handle the comparison.

- Increased Readability:

Now the code is more concise and focused, making it easier to read and maintain.

- Reusability:

The compare method can be reused for any other comparisons without repeating the same block of code.

Refactoring Code: Example 2

```
public class RefactorExample {
    public static void main(String[] args) {
        int number = 5;
        if (number == 1) {
            System.out.println("One");
        } else if (number == 2) {
            System.out.println("Two");
        } else if (number == 3) {
            System.out.println("Three");
        } else if (number == 4) {
            System.out.println("Four");
        } else {
            System.out.println("Unknown");
        }
    }
}
```

Refactoring Code: Solution

Refactored Code (with switch statement):

```
public class RefactorExample {
    public static void main(String[] args) {
        int number = 5;
        switch (number) {
            case 1: System.out.println("One"); break;
            case 2: System.out.println("Two"); break;
            case 3: System.out.println("Three"); break;
            case 4: System.out.println("Four"); break;
            default: System.out.println("Unknown"); break;
        }
    }
}
```

Refactoring Code: Explanation

- The original code uses multiple if-else statements to check different conditions.
- The refactored version replaces the if-else blocks with a switch statement, which is cleaner and more efficient when dealing with multiple conditions based on the same variable.

Source Code to UML Class Diagram

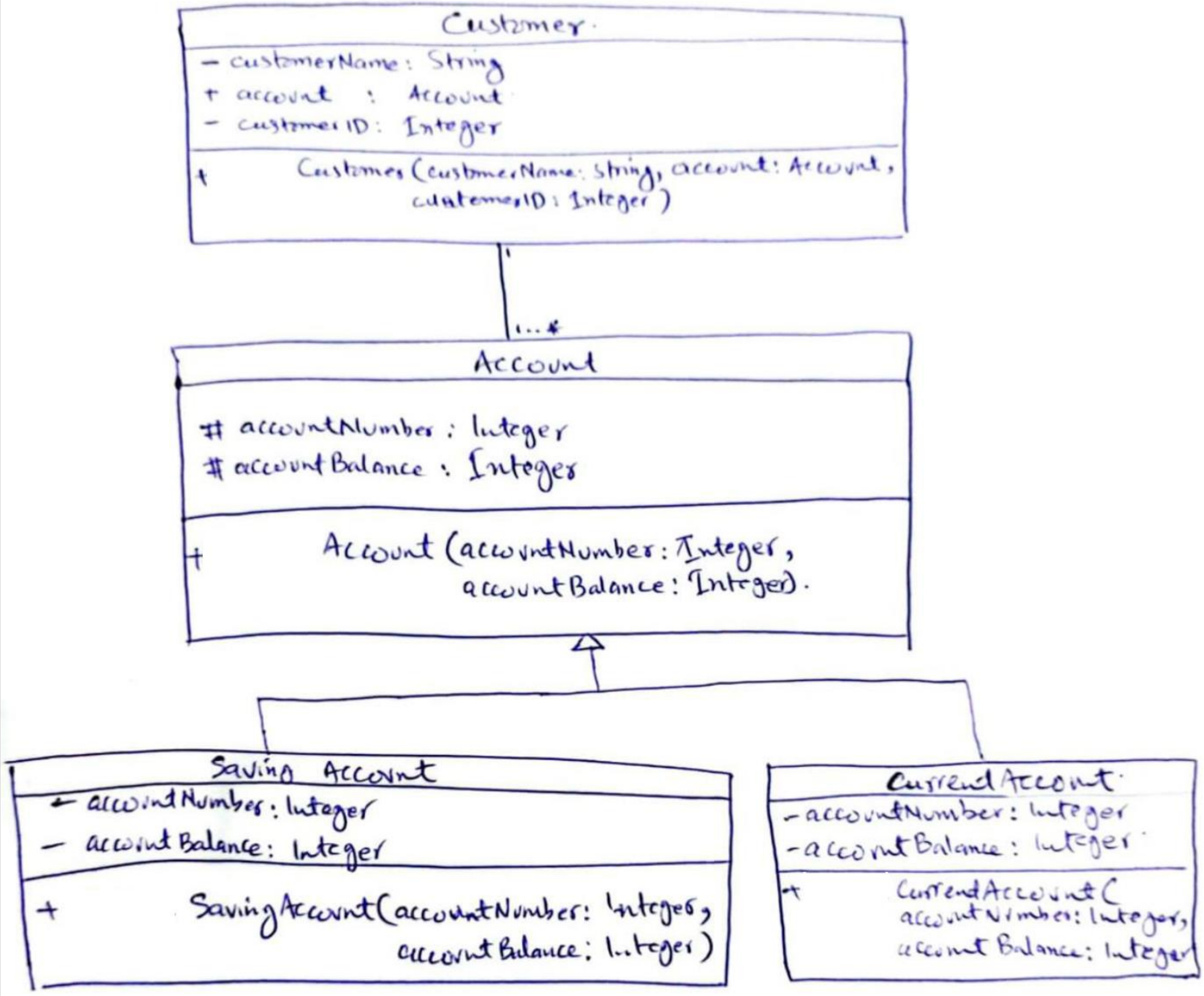
```
public class Customer {
private String customerName;
public Account account;
private Integer customerID;
//Getter of customerName
public String getCustomerName() {
return customerName;
}
//Setter of customerName
public void setCustomerName(String customerName) {
this.customerName = customerName;
}
//Getter of account
public Account getAccount() {
return account;
}
//Setter of account
public void setAccount(Account account) {
this.account = account;
}
//Getter of customerID
public Integer getCustomerID() {
return customerID;
}
//Setter of customerID
public void setCustomerID(Integer customerID) {
this.customerID = customerID;
}
public Customer(String customerName, Account account,
Integer customerID) {
this.customerName = customerName;
this.account = account;
this.customerID = customerID;
} }
```

```
public class Account {
protected Integer accountNumber;
protected Integer accountBalance;
//Getter of accountNumber
public Integer getAccountNumber() {
return accountNumber;
}
//Setter of accountNumber
public void setAccountNumber(Integer accountNumber) {
this.accountNumber = accountNumber;
}
//Getter of accountBalance
public Integer getAccountBalance() {
return accountBalance;
}
//Setter of accountBalance
public void setAccountBalance(Integer accountBalance) {
this.accountBalance = accountBalance;
}
public Account(Integer accountNumber, Integer
accountBalance) {
this.accountNumber = accountNumber;
this.accountBalance = accountBalance;
}
}
```

Source Code to UML Class Diagram

```
public class SavingAccount extends Account {
private Integer accountBalance;
private Integer accountNumber;
//Getter of accountBalance
public Integer getAccountBalance() {
return accountBalance;
}
//Setter of accountBalance
public void setAccountBalance(Integer
accountBalance) {
this.accountBalance =
accountBalance;
}
//Getter of accountNumber
public Integer getAccountNumber() {
return accountNumber;
}
//Setter of accountNumber
public void setAccountNumber(Integer
accountNumber) {
this.accountNumber =
accountNumber;
}
public SavingAccount(Integer
accountBalance, Integer accountNumber;) {
this.accountBalance =
accountBalance;
this.accountNumber =
accountNumber;
} }
```

```
public class CurrentAccount extends Account {
private Integer accountNumber;
private Integer accountBalance;
//Getter of accountNumber
public Real getAccountNumber() {
return accountNumber;
}
//Setter of accountNumber
public void setAccountNumber(Integer
accountNumber) {
this.accountNumber = accountNumber;
}
//Getter of accountBalance
public Integer getAccountBalance() {
return accountBalance;
}
//Setter of accountBalance
public void setAccountBalance(Integer
accountBalance) {
this.accountBalance = accountBalance;
}
public CurrentAccount
(Integer accountBalance, Integer accountNumber;) {
this.accountBalance = accountBalance;
this.accountNumber = accountNumber;
}
}
```


Customer
- customerName : String
+ account : Account
- customerID : Integer
+ Customer (customerName: string, account: Account, customerID : Integer)
1
1...*
Account
# accountNumber : Integer
# accountBalance : Integer
+ Account (accountNumber : Integer, accountBalance : Integer).
Saving Account
- accountNumber : Integer
- accountBalance : Integer
+ SavingAccount(accountNumber: Integer, accountBalance: Integer)
CurrentAccount
- accountNumber : Integer
- accountBalance : Integer
+ CurrentAccount(accountNumber: Integer, accountBalance: Integer

Thank You!