National University of Computer and Emerging Sciences

Software Re-engineering (SE4001)

Date: March 1st 2024

Course Instructor(s)

Dr. Syed Muazzam Ali Shah

Sessional-I Exam

Total Time: 1 Hours
Total Marks: 30
Total Questions: 02

Semester: SP-2024
Campus: Karachi
Dept: Software Engineering

_____ _____ _____ _____

Student Name Roll No Section Student Signature

CLO # 1: Define the concepts and techniques of software reengineering.

Q1: Answer the following questions …… [5*3=15 marks]

a. Explain the differences between software reengineering and refactoring.

Answer:

Re-engineering:

- Re-engineering takes place after a system has been maintained for some time and maintenance costs are increasing. You use automated tools to process and re-engineer a legacy system to create a new system that is more maintainable.

Refactoring:

- Refactoring is a continuous process of improvement throughout the development and evolution process. It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.

Re-engineering	Refactoring
• It is a maintenance process, so that the understandability and the structure of the program can be improved	• The original structure of the program is improved. By doing so, the complexity of the program can be reduced
• It can be applied even to legacy software	• It is limited to object oriented development programs
• You can add any new functionality to the already existing system	• Addition of new functionality is not allowed
• Reverse engineering is allowed	• It is based on agile methods, so reverse engineering is not allowed

b. Discuss the two techniques that facilitate us to accomplish reverse engineering.

Answer:

Redocumentation

- Redocumentation is the simplest and oldest form of reverse engineering, and many consider it to be a weak form of restructuring. The "re-" prefix implies that the intent is to recover documentation about the subject system that existed or should have existed.

Design recovery

- Design recovery recreates design abstractions from a combination of code, existing design documentation (if available), personal experience, and general knowledge about problem and application domain.
-

c. Explain the three approaches/techniques that are effective for successfully reengineering a software system.

Restructuring

- Automatic conversion from unstructured to structured code source code translation

Data reengineering

- Integrating and centralizing multiple databases unifying multiple, inconsistent representations upgrading data models

Refactoring

- Renaming/moving methods/classes/making the code/architecture more understandable etc.

d. Briefly describe the three main types of software maintenance. Why is it sometimes difficult to distinguish between them?

Bug Fixing (Corrective maintenance)- this is repairing faults found in the software after it has been launched. The bugs are there possibly because testing was not as thorough as it should have been or clients have exposed bugs by using the software in unexpected ways. Coding errors, design errors, and requirement errors are the least, middle, and most expensive to correct, respectively.

Modifying software to work in a new environment (Adaptive maintenance) - when the hardware or platform that the system was built to run on changes, then the software must change as well in order to be compatible and avoid being obsolete.

Implementing new or changed requirements (Perfective maintenance)- software must be updated or changed so that it conforms with any new requirements.

It is sometimes difficult to distinguish between the different types of maintenance because they are often given different names and also because faults that arise within a system can maybe have overlapping maintenance requirements.

e. Explain how advances in technology can force a software subsystem to undergo change or run the risk of becoming useless.

Answer:

- New widespread technological advancements need not always be backward compatible with all types of older systems. Newly designed usage interfaces data volume exchanges, and formats may all require existing systems to undergo updates, or risk becoming obsolete:

- ***Example 1:*** With the introduction of widespread cloud-based services, software systems need to adapt to be able to use the cloud infrastructure for their operations similar to what competitive products may be doing.
- ***Example 2:*** An application that allowed mobile applications to connect in an ad-hoc manner needs to adapt as newer communication technology is introduced to new releases of mobile devices. If the applications do not adapt, they risk being obsolete. Because new technologies often offer improved capabilities, security, or efficiency. Failure to adapt can lead to obsolescence and a loss of value to users and organizations.

CLO # 2: **Demonstrate and utilize reengineering techniques to maintain and modify software systems.**

Q2: Answer the following questions …… [15 marks]

a. Consider the given source code and attempt to refactor it. [05 marks]

```
void printOwing(){
    printBanner();
    //Print Details
    System.out.println("name:" + name);
    System.out.println("amount:" + outstanding());}
```

Answer:

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}
void printDetails (double outstanding) {
  System.out.println ("name: " + _name);
  System.out.println ("amount      " + outstanding);
}
```

Consider the given source code and attempt to reverse engineer it into a UML class diagram.
[10 marks]

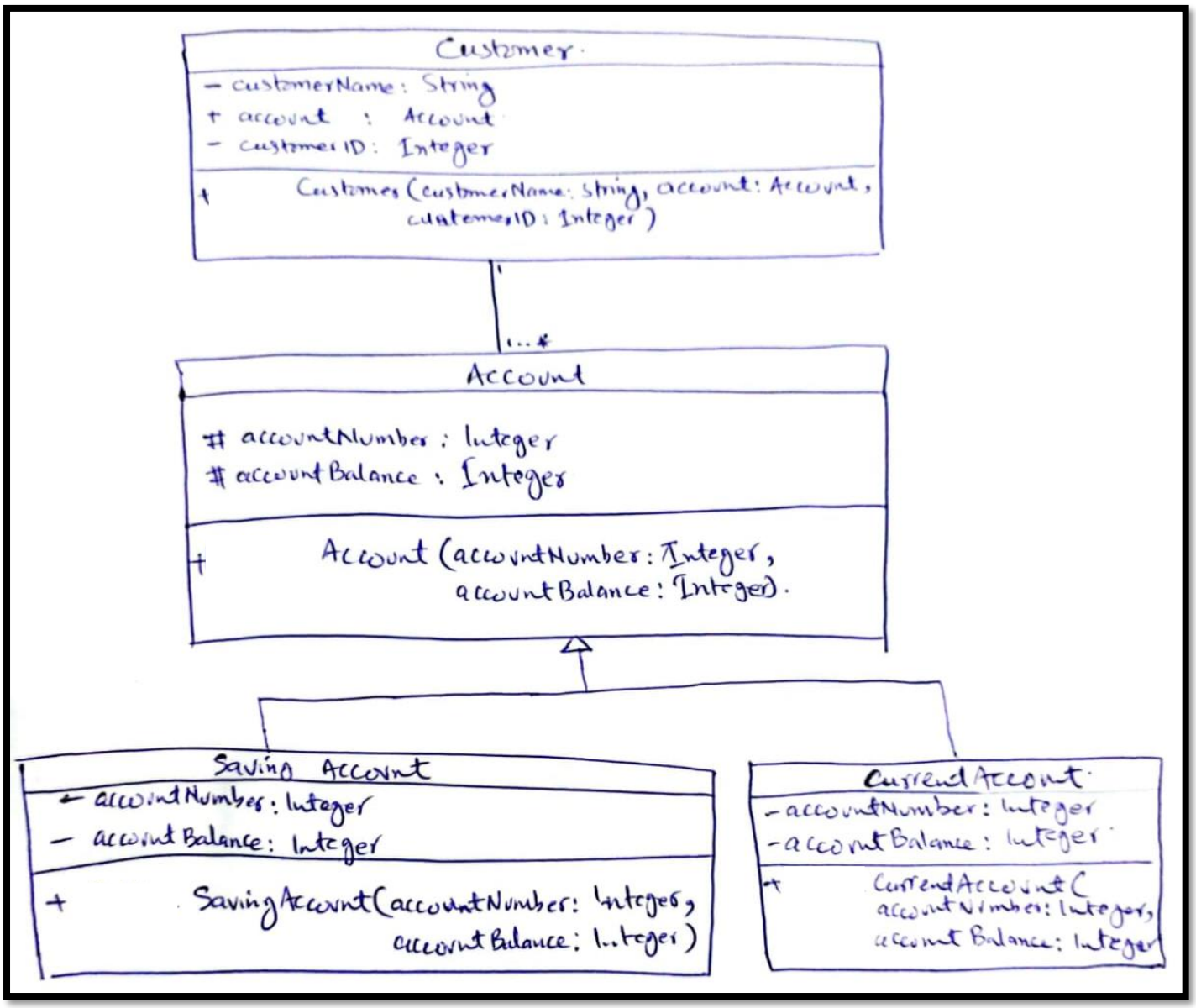
```
public class Customer {
    private String customerName;
    public Account account;
    private Integer customerID;
    //Getter of customerName
    public String getCustomerName() {
    return customerName;
    }
    //Setter of customerName
    public void setCustomerName(String
customerName) {
    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;
         }
}
```

```
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;
         }
}
```

Answer:


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).
SavingAccount
- accountNumber: Integer
- accountBalance: Integer
+ SavingAccount(accountNumber: Integer, accountBalance: Integer)
CurrentAccount
- accountNumber: Integer
- accountBalance: Integer
+ CurrentAccount(accountNumber: Integer, accountBalance: Integer