CLO 3: Implement object-oriented principles for software analysis and design

Q1 ..... [10 x 2 = 20]

a. Consider the following design. Identify which SOLID principle is violated and why. Propose a design to fix the violation.
Note: No credit will be given in case of failure to identify the most appropriate SOLID principle (being violated).

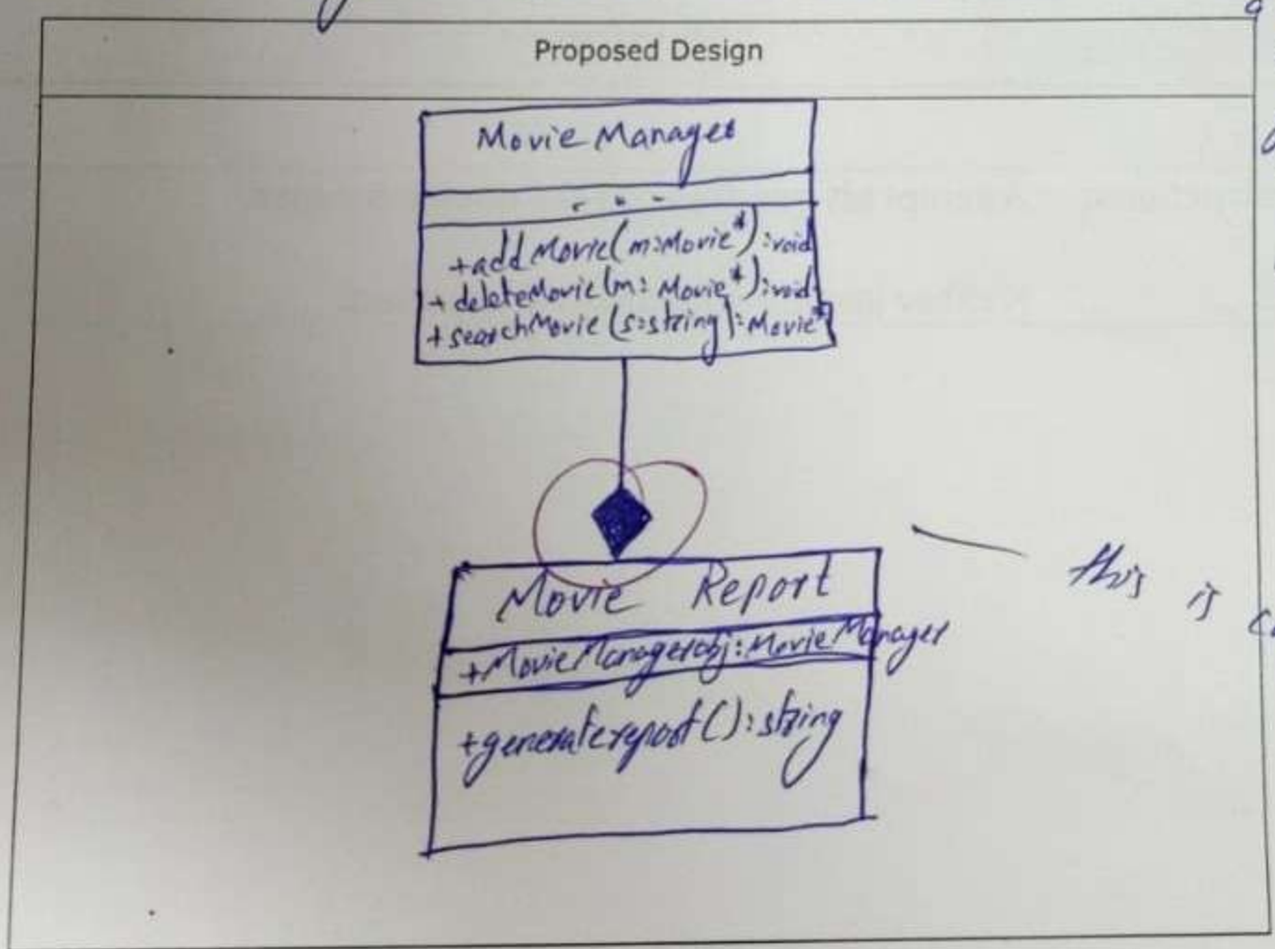
MovieManager
+addMovie(m: Movie*): void +deleteMovie(m: Movie*): void +searchMovie(s: string): Movie* +generateReport(): void

8

SOLID principle being violated: Separation of functions/concerns. (1st poircipl of Solid Principles).

Reason for violation (less than 40 words):
Each class should have a very specific function and not only adds/deletes/ searches the movies, it also generates a report. The report generation should be in a different class.

Proposed Design



b. Consider the following code. Identify which SOLID principle is violated and why. Modify the code to fix the violation.
Note: No credit will be given in case of failure to identify the most appropriate SOLID principle (being violated).

```
public class Account {
  protected double balance;
  public void deposit(double
amount) {
    if (amount <= 0) {
        throw new
        IllegalArgumentException
        ("amount <=0");
    }
    balance += amount;
  }
}
```

```
public class SavingsAccount extends Account {
  @Override
  public void deposit(double amount) {
    super.deposit(amount);  // Call base class deposit
    // Apply minimum balance fee after successful deposit
    if (balance < 100) {
      // Charge a fee if balance falls below minimum
      balance -= 5;
    }
  }
}
```

3

SOLID principle being violated: Liskov Substitution Principle.

Reason for violation (less than 40 words):
Because the sub or child class needs to be replacable with the function of the parent/super class, but in this case the 2nd deposit function is called from the 1st deposit function

Modified Code

```
public class Account {
  protected double balance;
  public void deposit(
      double
      amount)
  {
    if (amount <=0)
    { throw new
      IllegalArgumentException
      ("amount <=0");
    }
    balance += amount;
  }
}
```

```
public class SavingAccount extend Account
{
  @Override
    public void deposit(double amount)
    {
      if (amount <=0) {
          throw new IllegalArgumentExcep
          ("amount <= 0");
      }
      balance += amount;
      if (balance <100) {
          balance -= 5;
      }
    }
}
```


0155

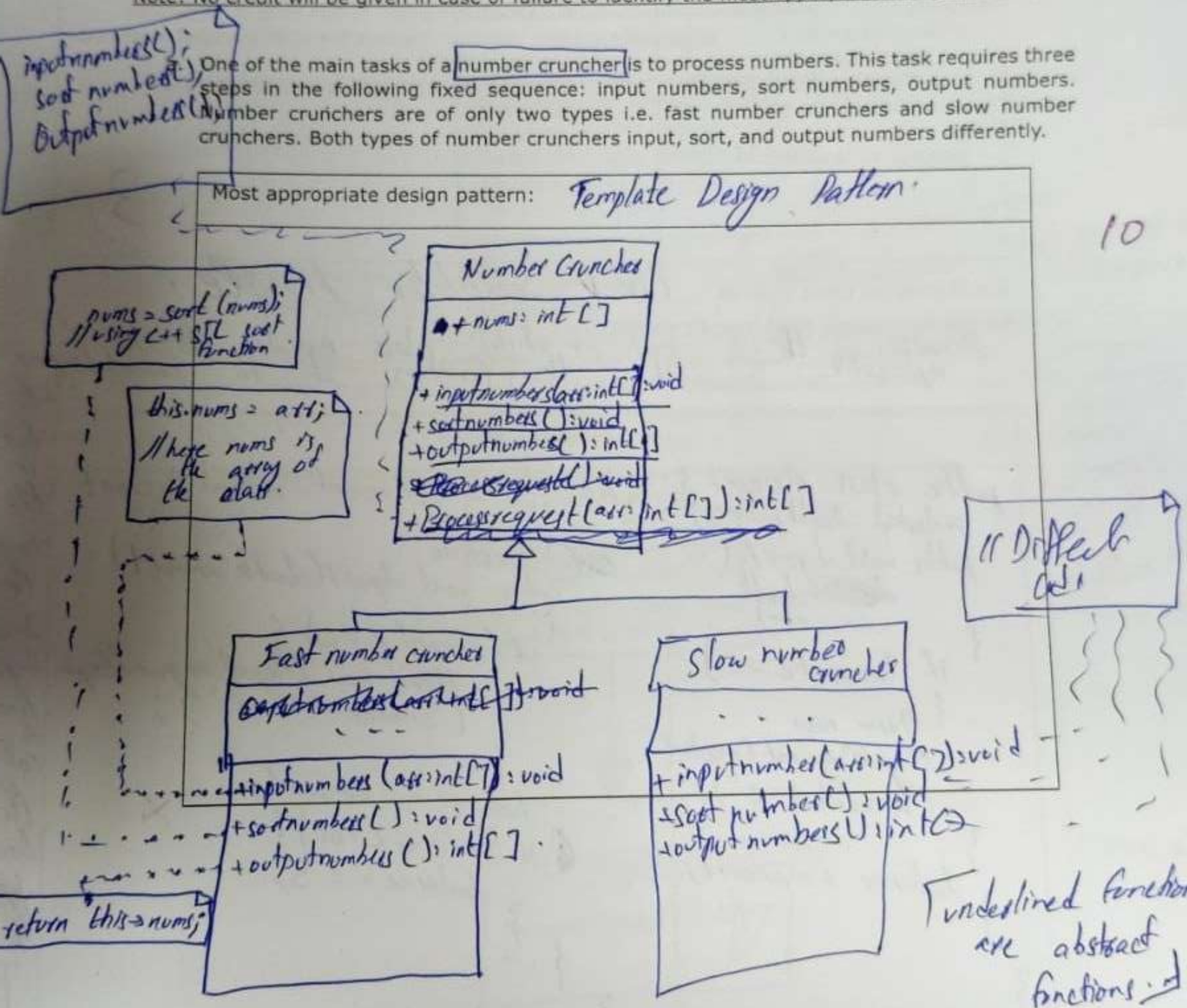
CLO 3: Implement object-oriented principles for software analysis and design

Q2 ..... **[10 x 2 = 20]**

Map the information given in each part below to a **UML 2 design class diagram** that uses the **most appropriate design pattern**. Annotate your diagram (drawn inside the box) with important comments containing error-free C++ code. Realistic and relevant assumptions may be made where necessary.
Note: No credit will be given in case of failure to identify the most appropriate design pattern.

a) One of the main tasks of a number cruncher is to process numbers. This task requires three steps in the following fixed sequence: input numbers, sort numbers, output numbers. Number crunchers are of only two types i.e. fast number crunchers and slow number crunchers. Both types of number crunchers input, sort, and output numbers differently.

Most appropriate design pattern: Template Design Pattern.



10

01

b. Residential plots are of only three types i.e. square plots, rectangular plots, and combo plots. A combo plot contains at least two residential plots of any type (including combo plots). Area can be calculated for all types of residential plots. The area of a square plot is the square of its length. The area of a rectangular plot is the product of its length and width. The area of a combo plot is the sum of the areas of the residential plots it contains.

Most appropriate design pattern: Composite Design Pattern.