Institute of Information Technology (IIT)

Noakhali Science and Technology University

Final Examination - 2020 (Year-3, Term-1)

Course Title: Design Pattern **Total Time:** 15 Minutes

Course Code: SE 3109 **Full Marks:** 10

Answer all the subsections of Questions No. 1: **1 X 10 = 10**

Question No 1

(a) The OO Principle: "One class, one responsibility" can also mean we want classes to have:

i. High cohesion ii) Low cohesion iii) Tight coupling iv) Loose coupling

(b) "Smaller and well-organized classes are easier to search than monolithic ones" is the benefit of

i.SRP ii. OCP iii. LSP iv. DIP

(c) Which is usually considered the better design in the Observer Pattern?

i). Push data to the Observer ii). Push data to the Observable

iii). Pull data from the Observer iv). Pull data from the Observable

(d) Liskov substitution principle is very closely related to _____

i. SRP ii. OCP iii. ISP iv. DIP

(e) Which pattern prevents one from creating more than one instance of a variable?

i. Factory Method ii. Singleton iii. Observer iv. None

(f) "Design pattern is a solution to a problem that occurs repeatedly in a variety of contexts" is this statement true or false?

(g) Which is often considered a better design choice?

i. Using a null object ii. Using the null reference

(h) Which is not a Bloaters type of code smell?

i. Inappropriate Naming ii. Large Class iii. Primitive Obsession iv. Long Method

(i) "Speculative generality is a piece of code which often exists to support future behavior, which may or may not be necessary in the future" is this statement true or false?

(j) In general, creating a new object passing all the initial values in to a single constructor vs. reusing an existing object and calling multiple mutator (set) methods to reset that object's instance variables is:

i. about the same cost ii. creating a new object is more expensive

iii. calling multiple mutator methods is more expensive

Course Title:	Design Pattern	**Total Time:**	1 Hour
Course Code:	SE 3109	**Full Marks:**	30

Answer any SIX (06) Questions from the following: **6 X 5 = 30**

2. **a.** Consider the following code snippet. If you think you could apply better coding design using design principles then rewrite the code by mentioning which principle you are going to use with the reason. **5**

```
public class Order{
publicGuid ID { get { /*...*/ } }
public decimal Total { get { /*...*/ } }
public List<Item> Items { get { /*...*/ } }
public User User { get { /*...*/ } }
public void Load() { /*...*/ }
public void Save() { /*...*/ }
public void Update() { /*...*/ }
public void Remove() { /*...*/ }
}
```

3. **a.** Explain the benefits of using S.O.L.I.D at the software development phase. **2**

b. Rewrite The code by applying the design principle concept. **3**

```
public class Rectangle{
    public double length, width;
}
public class Circle{
    public double radius;
}
public class AreaCalculator {
   public double calculateRectangleArea(Rectangle rectangle){
   return rectangle.length *rectangle.width;
}
   public double calculateCircleArea(Circle circle){
   return (22/7)*circle.radius*circle.radius;
}}
```

4. **a.** Consider a game in which a player is awarded a certain number of points for collecting a variety of fruit. In order to introduce a new type of fruit, the programmer would have to change the scoreFruit method. **5**
State the violation of design principle(s) associated with this below code snippet. Correct this code snippet so that the violation of design principles is eliminated.

```
public class Player {
    public void scoreFruit(String fruitType) {
        if (fruitType.equals("banana")) {
            this.score += 5;
        } else if (fruitType.equals("apple")) {
            this.score += 20;
        } else if (fruitType.equals("jackfruit")) {
            this.score += 50;
```

```
        }
    }
}
```

5. **a.** The below code snippet calculates the income tax of a person. The method 'calculate' takes income as input. **2.5**

```
public double calculate(int i){
 double t = 0;
 if (i>=500000){ t = (300000-i)*0.20;}
 else if (i>=400000){ t = (300000-i)*0.10; }
 else if (i>=300000){ t = (300000-i)*0.05;}
 return t;
}
```

State at least one 'code smell' in the above code snippet. Refactor that code smell.

b. **2.5**

```
class Coffee {
   void prepareRecipe() {
        boilWater();
        brewCoffeeGrinds();
        pourInCup();
       addSugarAndMilk();
   }
}
```

```
class Tea{
  void prepareRecipe() {
      boilWater();
      steepTeaBag();
      pourInCup();
      addLemon();
  }
}
```

Do you see the similarity? How might you avoid code duplication? Think of more than one way. How should we write the code?

6. **a.** The following code snippet is a method 'sendMoney' of a mobile transaction application. **2.5**

```
public boolean sendMoney(String senderPhone, String senderName, String
receiverPhone, String receiverName, double amount){
    // send money code
}
```

State at least one 'code smell' in the above code snippet. Refactor that code smell.

b. The following code snippet is a method 'getWeight of an object. **2.5**
State 'code smells' in the below code snippet. Refactor those code smells.

```
public double getWeight(double mass, double G, double M, double R){
   double weight = 0;
   double g = G * M;
   g = g / R ;
   g = g / R;
   weight = mass * g;
   return weight;
}
```

7. **a.** Find out the code smell with the category and mention the approaches of the refactoring process to remove the smell. **5**

i.

```
public class CapitalStrategy{
double capital(Loan loan){
  if (loan.getExpiry() == NO_DATE && loan.getMaturity() != NO_DATE)
       return loan.getCommitmentAmount() * loan.duration()
*loan.riskFactor();
  if (loan.getExpiry() != NO_DATE && loan.getMaturity() == NO_DATE) {
       if (loan.getUnusedPercentage() != 1.0)
             return loan.getCommitmentAmount() *
loan.getUnusedPercentage() * loan.duration() * loan.riskFactor();
       else
             return (loan.outstandingRiskAmount() * loan.duration() *
loan.riskFactor()) + (loan.unusedRiskAmount() * loan.duration() *
loan.unusedRiskFactor());
       }
  return 0.0;
}}
```

ii.

```
private void Grow() {
       Object[] newElements = new Object[elements.length + 10];
       for (inti = 0; i< size; i++)
        newElements[i] = elements[i];
       elements = newElements;
}
```

8. **a.** "Comments are often used as deodorant"- justify this statement. What are the remedies of comment? **3**

b. Explain the relation between pattern and refactoring. **2**

9. **a.** Why do we refactor our code? **1**

b. Find out the smells from the following code. Mention the name of the refactoring process to improve this code segment. Write the refactored code. **4**

```
String foundPerson(String[] people){
    for (inti = 0; i<people.length; i++) {
        if (people[i].equals ("Don")){
            return "Don";
}
        if (people[i].equals ("John")){
           return "John";
}
        if (people[i].equals ("Kent")){
            return "Kent";
}
    }
    return "";
}
```