الأسم:- محمد محمود محمود سيد احمد عجوه

القسم:- (SWE)

Sheet 01:-

1- What is software re-engineering?

d) All of the mentioned

2- It is a process of improving the structure of the program to optimize memory use.

d) Program structure improvement

3- Which one of the following approaches replaces the entire system at one time as a total replacement?

a) Big Bang approach

4- In .... approach; there are NOT interfaces between old system and new system.

a) Big Bang

5- System element replaced with newly re-engineered in.......... approach.

b) Incremental

Sheet 02:-

1- When to refactor?

D) all of the mentioned

2- Lack of documentation is not one of the causes of dirty code in software.

B) FALSE

3- Refactoring is a systematic process of improving code with creating new functionality.

B) FALSE

4................................. is one of the causes of dirty code in software.

c) a & b

5-Which of these is true of refactoring?

A) It can be applied to any programming language

Sheet 03:-

1)

```
static void Main()
{
    int number1 = 1;
    int number2 = 2;
    int result = AddNumbers(number1, number2);
    Console.WriteLine(result);
}

static int AddNumbers(int a, int b)
{
    return a + b;
}
```

2)

```
static void Main()
{
    double area = Math.PI * 1.23 * 1.23;
    Console.WriteLine(area);
}
```

3)

```
static void Main()

{

    Addone(5);

}


static void Addone(int number)

{

    int result = number + 1;

    Console.WriteLine(result);

}
```

4)

```
static void Main()

{

    double radius = 1.23;

    double area = Math.PI * radius * radius;

    Console.WriteLine(area);

}
```

5)

```
const double Gravity = 9.81;

double PotentialEnergy(double mass, double height)

{

    return mass * height * Gravity;

}
```

Sheet 04:-

1)

```
public class MyClass

{

    private int m_Number;


    public int Number

    {

        get { return m_Number; }

        set { m_Number = value; }

    }

}
```

2)

```
static void Main()
{
    Console.WriteLine(Increment(6));
}

public static int Increment(int number)
{
    return number + 1;
}
```

Sheet 05:-

➤ What is the Singleton Pattern?

The Singleton is a creational design pattern that ensures a class has only one instance throughout the entire program and provides a global access point to that instance.

➤ Why use Singleton?

It's useful when exactly one object is needed to coordinate actions across the system, such as in:

- **Database connections**
- **Logging systems**
- **Configuration settings**

➤ **Features:**

- **Only one instance is created (ensures uniqueness)**
- **Prevents creating multiple objects that use the same resources**
- **Provides lazy initialization (object is created only when needed)**

➤ **Implementation (Example in C#):**

```
public class Singleton
{
   private static Singleton instance;

   private Singleton() { }

   public static Singleton GetInstance()
   {
       if (instance == null)
           instance = new Singleton();
       return instance;
   }
}
```

➤ **Advantages:**

- **Saves memory by preventing multiple object creation.**
- **Centralized management of shared resources.**
- **Easy access from anywhere in the application.**

➤ **Disadvantages:**

- **Difficult to test (because of the global state).**
- **Hidden dependencies between classes.**
- **Can be misused like a global variable.**

➤ **Conclusion:**

The Singleton Pattern is a powerful and commonly used design pattern that ensures a single point of control in applications. However, it should be used wisely to avoid over-dependency and testing issues.