Faculty of Computing, Online Examinations 2021

STUDENT NAME	**D M N C Dissanayake**		
INDEX NUMBER (NSBM)	**19840**	**YEAR OF STUDY AND SEMESTER**	**Year 1 Semester 3**
MODULE NAME **(As per the paper)**	**Object Oriented Programming with C#**		
MODULE CODE	**CS 107.3**		
MODULE LECTURER	**Mr. Pramudya Thilakarathne**	**DATE SUBMITTED**	**14. 07. 2021**

For office purpose only:

GRADE/MARK	
COMMENTS	

Declaration

PLEASE TICK TO INDICATE THAT YOU HAVE SATISFIED THESE REQUIREMENTS

- ✓ I have carefully read the instructions provided by the Faculty
- ✓ I understand what plagiarism is and I am aware of the University's policy in this regard.
- ✓ I declare that the work hereby submitted is my own original work. Other people's work has been used (either from a printed source, Internet or any other source), has been properly acknowledged and referenced in accordance with the NSBM's requirements.
- ✓ I have not used work previously produced by another student(s) or any other person to hand in as my own.
- ✓ I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.
- ✓ I hereby certify that the individual detail information given (name, index number and module details) in the cover page are thoroughly checked and are true and accurate.

I hereby certify that the statements I have attested to above have been made in good faith and are true and correct. I also certify that this is my own work and I have not plagiarized the work of others and not participated in collusion.

Date: 14. 07. 2021 **E- Signature: [illegible]

**Please attach a photo/image of your signature in the space provided.

Question 01

1. **Importance of Access Specifiers**

The are four main concepts in object-oriented programming as encapsulation, abstraction, inheritance, and polymorphism. In order to execute these OOP concepts, we need components called **access specifiers.** It basically points out the level of visibility, or the **level of accessibility** of a variable or a method within a class. There are five types of access specifiers in C# as public, protected, private, internal and protected internal. But the public, private, and protected access specifiers are the ones that are used mostly. They **define the different levels of accessibility in class members**.

2. **Public vs Protected**

Public access specifier has the highest level of visibility and accessibility, just as its name suggests. But in protected, it is accessible for only a selected few. For example, a lecture within a university is accessible by all the lecturers and students at the university. Just like that, **public variables and methods can be accessed by all the classes in the project**. But for protected access specifiers, consider a situation where a parent owns some property. The parent consumes it and inherits them to his kids too. Likewise, **protected variables and methods are accessible within the class they are declared, and also within the classes derived from that class.** Protected access specifiers are used in inheritance OOP concept.

3. **Encapsulation**

Private access specifiers have the lowest visibility and lowest accessibility. They can be accessed only within the class they are declared. But in order to access them from an outside class, encapsulation can be used. Encapsulation uses **public methods(getters and setters) to access the private variables.** This is a technique for data hiding and giving out access to only required components. It helps to protect the code and reduce risk of breaking.

4.

Student Class

```
class Student
   {
       private int age;

       public void setAge(int UserAge)
       {
           age = UserAge;
       }

       public int getAge()
       {
           return age;
       }
   }
```

Main Method

```
static void Main(string[] args)
       {
           Console.Write("Please enter your age: ");
           int age = int.Parse(Console.ReadLine());

           Student s = new Student();
           s.setAge(age);
           Console.WriteLine("Your age is " + s.getAge());

           Console.ReadLine();
       }
```

Question 02

1. **Exceptions vs Syntax Errors**

Syntax errors **occur in the syntax**, or the words we are using to write the program. In the modern compilers, the syntax errors get underlined in red color and you cannot run the program when it has a syntax error. But in exceptions, **they occur at the runtime of the program**. Exceptions occur when the program is running, and when it detects an exceptional scenario like an attempt to divide by zero. And then, as the response, the exception causes the program to **crash.** We can use **exception handling** to manage the situation in case of exceptions.

2.

Exception is important when dealing with exceptions. Just imagine you write a program to a client who does not have much computer related knowledge. In such instances, that person does not know what happened when an exception occurred, and the program crashed. So, through exception handling **we can figure out possible code pieces that can cause exceptions and use exception handling on them. It prevents the program from crashing** even if an exception occurs. And also, we can give a **human readable message** so as to why the exception occurred and thereby notify the user about the mistake he made.

3. **try, catch, finally**

These are keywords used in **exception handling**. **Try** is used to **hold the code that probably cause an exception**. Try block should have at least one catch block. There can be many catch block for one try statement. **Catch** is used to handle the exception, if any occurred within the try block. It figures out what kind of exception occurred and also can give out a message as to what caused the exception. **Finally** is not an essential component. It can either be used or not. But it is necessary for the **code cleanup** part. The **code within the finally block executes, regardless an exception occurred or not**. This can be used to close database connections, or even end the program if necessary.

4.

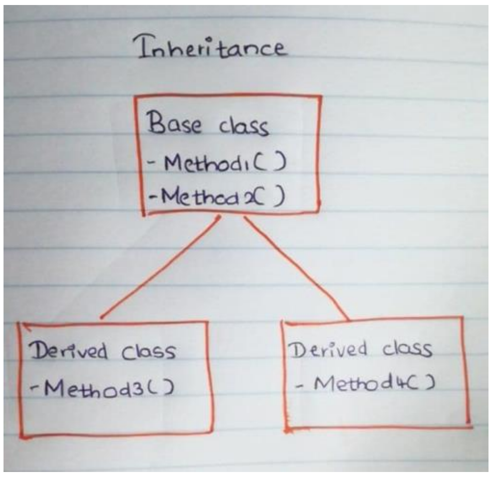
```
class Division
   {
      public void CalcDiv()
      {
         Console.Write("Enter first number: ");
         int num1 = int.Parse(Console.ReadLine());
         Console.Write("Enter second number: ");
         int num2 = int.Parse(Console.ReadLine());

         try
         {
            Console.WriteLine("Your values is: " + (num1 / num2));
         }
         catch(System.DivideByZeroException d)
         {
            Console.WriteLine(d);
         }
         finally
         {
            Console.WriteLine("Thank you!");
         }

      }
}
```

Question 03

1. **Inheritance**

Inheritance is one of the four major OOP concepts and it is used by many instance in C# programming. Inheritance **allows creating one base class, and deriving multiple classes from it**. So, it helps to reduce the code repetition too. In this method, the code can be reused again and again within other classes. **The base class is also known as the parent classes and the derived classes are also known as child classes.** The child class owns all the public and protected methods and variables from the parent class, and the child class can have its own methods and variables too.

2.



As in the above diagram, the derived class inherits its methods to the derived classes and the derived classes have their own methods and variables too.

3.
a.

```
   class Person

      {

          protected int age;
```

```
        protected string name, gender;

    }
```

b.

```
  class Employee : Person

    {

        public int Employee_ID;

    }
```

c.

```
  class Specialist : Employee

    {

        public int Specialist_ID;

    }
```

d.

```
  class Specialist : Employee

    {

        public int Specialist_ID;


        public void PrintValues()

        {

            Console.WriteLine("Name is : " + name);

            Console.WriteLine("Age is : " + age);

            Console.WriteLine("Gender is : " + gender);
            Console.WriteLine("Employee ID is : " + Employee_ID);

            Console.WriteLine("Specialist ID is : " + Specialist_ID);

          }

        }
```

e.

Specialist Class

```
class Specialist : Employee

    {

        public int Specialist_ID = 23421;


        public void PrintValues()

        {

            Console.WriteLine("Name is : " + name);

            Console.WriteLine("Age is : " + age);

            Console.WriteLine("Gender is : " + gender);

            Console.WriteLine("Employee ID is : " + Employee_ID);

            Console.WriteLine("Specialist ID is : " + Specialist_ID);

        }

    }
```

Main method

```
static void Main(string[] args)

        {

            Specialist s = new Specialist();
            s.PrintValues();


            Console.ReadLine();

        }
```

Question 04

```
private void button1_Click(object sender, EventArgs e)

        {

            string ConnectionString = @"DataProvider=.\SQLEXPRESS;Data
Source=E:\NSBM\School.mdf";

            string show = "SELECT * FROM Modules";

            try

            {

                SqlDataAdapter sda = new SqlDataAdapter(show, ConnectionString);

                DataSet d = new DataSet();


                sda.Fill(d, "Modules");

                dataGridView1.DataSource = d.Tables["Modules"];

            }

            catch(SqlException s)

            {

                MessageBox.Show(s.ToString());

            }

        }
```

Question 05

1. **Constructor vs Method**

Constructor	**Method**
Two types as default constructor and parameterized constructors	Four types as, 1. Return type with no parameters 2. Return type with parameters 3. No return type with parameters 4. No return type with no parameters
Has a name, but it should be the name of the class	Has a name, but it can be any name that comply with the naming conventions
Does not have a return type	Has a return type
Has access specifiers, but has to be declared as public. There is no point of declaring as any other access specifier as then it becomes unable to access from other classes	Has access specifiers and they can be used as desired
Can contain parameters and supports overloading. The parameters are entered through object at initiation.	Can contain parameters and supports overloading
Called by default at the moment the object is created.	Called when necessary after creating the object
Avoid code complexity	Avoid code complexity

2.

```
class Student

   {

      public Student(int Stud_ID)
```

```
        {
            Console.WriteLine("Hi " + Stud_ID);
        }
    }
```

3.

The class constructor, whether parameterized or not, executes at the moment the object of its class is created. That is why the constructor have the same name as its class. If it is parameterized, we have to pass the parameters when the object is created as given in the code below.

```
static void Main(string[] args)
    {
        Student s = new Student(19840);


        Console.ReadLine();
    }
```

4. Console Applications vs Windows Forms Applications

The major difference between these two types of programs is that the windows forms are **GUI applications,** and the Console applications are not. Console application programs run on the console. Windows forms applications have its own components like tool box for UI components, server explorer to connect databases and so on. The console application do not produce GUI outputs as it performs the logical and arithmetic operations and just displays on the console. On the other hand, in windows forms applications, the GUI is displayed upon execution and the logical and arithmetic operations are performed by using controllers like buttons.

5.

```
private void signup_Click(object sender, EventArgs e)

        {

                Registration r = new Registration();

                this.Hide();

                r.Show();

        }
```