C# Assignments on Classes & Objects

Assignment 1: Circle Class

Problem Statement: Create a Circle class with a property for Radius. Implement a getter to retrieve the radius and a setter to ensure that the radius cannot be negative.

```
public class Circle
{
    private double radius;
    public double Radius
    {
        get { return radius; }
        set
        {
            if (value < 0)
            {
                throw new ArgumentException("Radius cannot be negative");
            }
            radius = value;
        }
    }
}
```

Assignment 2: Employee Class

Problem Statement: Design an Employee class with properties for Name and Salary. Use getters and setters to manage access to the salary, ensuring it cannot be set to a negative value.

Main program:

```
{
try
{
    Employee employee = new Employee();
    employee.Salary = 12000;
    employee.name = "john";
    Console.WriteLine(employee.name);
    Console.WriteLine(employee.Salary);
    employee.Salary = -50;
}
catch (ArgumentException e)
{
    Console.WriteLine(e.Message);
}
```

```
    Console.ReadLine();
}


public class Employee
{
    private int salary;
    public string name { get; set; }
    public int Salary
    {

        get { return salary; }
        set
        {
            if (value < 0)
            {
                throw new ArgumentException("Salary cannot be negative");
            }
            salary = value;
        }

    }
}
```

Assignment 3: Library Management System

Problem Statement: Design a library management system that manages books. Each book has a title, author, and ISBN number. The system should allow adding a book, removing a book, and displaying all books.

```
 class Book
 {
     public string Title { get; set; }
     public string Author { get; set; }
     public string ISBN { get; set; }
     public Book(string title, string author, string isbn)
     {
         Title = title;
         Author = author;
         ISBN = isbn;
     }
     public override string ToString()
     {
         return $"Title: {Title}, Author: {Author}, ISBN: {ISBN}";
     }
```

```
}
class Library
{
    private List<Book> books = new List<Book>();
    public void AddBook(Book book)
    {
        books.Add(book);
        Console.WriteLine($"Book '{book.Title}' added to the library.");
    }

    public void RemoveBook(string isbn)
    {
        Book bookToRemove = books.Find(b => b.ISBN == isbn);
        if (bookToRemove != null)
        {
            books.Remove(bookToRemove);
            Console.WriteLine($"Book '{bookToRemove.Title}' removed from the library.");
        }
        else
        {
            Console.WriteLine($"Book with ISBN {isbn} not found in the library.");
        }
    }
    public void DisplayBooks()
    {
        if (books.Count == 0)
        {
            Console.WriteLine("No books available in the library.");
        }
        else
        {
            Console.WriteLine("Books in the library:");
            foreach (var book in books)
            {
                Console.WriteLine(book);
            }
        }
    }
}
```

Main program:

```
        {
    Library library = new Library();
    Book book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", "1234567890");
    Book book2 = new Book("1984", "George Orwell", "9876543210");


    library.AddBook(book1);
    library.AddBook(book2);


    library.DisplayBooks();


    library.RemoveBook("1234567890");

    library.DisplayBooks();

}
```

Assignment 4: Banking System

Problem Statement: Create a simple banking system that allows account creation and basic transactions. Each account has an account number, account holder name, and balance. Implement
deposit and withdrawal methods. Use getters and setters to manage access to the balance, ensuring it cannot be set to a negative value.

```
class Account
{
    public string AccountNumber { get; set; }
    public string AccountHolderName { get; set; }
    private decimal balance;
    public decimal Balance
    {
        get { return balance; }
        private set
        {
            if (value < 0)
            {
                Console.WriteLine("Balance cannot be negative. Transaction aborted.");
            }
            else
            {
                balance = value;
            }
        }
    }
    public Account(string accountNumber, string accountHolderName, decimal initialBalance)
    {
        AccountNumber = accountNumber;
        AccountHolderName = accountHolderName;
        Balance = initialBalance;
    }

    public void Deposit(decimal amount)
    {
        if (amount > 0)
        {
            Balance += amount;
            Console.WriteLine($"Deposited: {amount}. New balance: {Balance}");
        }
        else
        {
            Console.WriteLine("Deposit amount must be positive.");
        }
    }
    public void Withdraw(decimal amount)
    {
        if (amount > 0)
        {
            if (amount <= Balance)
            {
                Balance -= amount;
                Console.WriteLine($"Withdrew: {amount}. New balance: {Balance}");
            }
            else
            {
                Console.WriteLine("Insufficient funds for this withdrawal.");
            }
        }
        else
        {
            Console.WriteLine("Withdrawal amount must be positive.");
        }
    }
```

Main program:

```
        {
```

```
    Account account = new Account("123456789", "John Doe", 1000);


    Console.WriteLine($"Account Created: {account.AccountHolderName}, Account Number:
{account.AccountNumber}, Initial Balance: {account.Balance}");


    account.Deposit(500);
    account.Withdraw(200);
    account.Withdraw(1500);
    account.Deposit(-50);
    account.Withdraw(-100);

    Console.WriteLine($"Final Balance: {account.Balance}");

}
```

Assignment 5: Student Management System

Problem Statement: Develop a student management system that stores student details. Each student has a name, ID, and a list of grades. Implement methods to add a grade and calculate the average grade.

```
public class Student
    {
        public string Name { get; set; }
        public int ID { get; set; }
        public
     List<int> Grades
        { get; set; }

        public Student(string name, int ID)
        {
            Name = name;
            this.ID = ID;
            Grades = new List<int>();
        }
```

Main program:

```
  {
    Student student1 = new Student("Alice", 12345);

}
```

Assignment 6: Inventory System

Problem Statement: Create an inventory management system that manages items in a store. Each item has a name, stock, and price. Implement methods to add, remove, and update items. Use getters and setters to manage access to the stock and price, ensuring it cannot be set to a negative value.

```
 public class Item
    {
        public string Name { get; set; }
        private int stock;
        private decimal price;

        public int Stock
        {
            get { return stock; }
            set
            {
                if (value < 0)
                    throw new ArgumentException("Stock cannot be negative.");
                stock = value;
            }
        }

        public decimal Price
        {
            get { return price; }
            set
            {
                if (value < 0)
                    throw new ArgumentException("Price cannot be negative.");
                price = value;
            }
        }

        public Item(string name, int stock, decimal price)
        {
            Name = name;
            Stock = stock;
            Price = price;
        }
    }

    public class Inventory
    {
        private List<Item> items = new List<Item>();
        public void AddItem(Item item)
        {
            items.Add(item);
            Console.WriteLine($"Added: {item.Name}");
        }

        public void RemoveItem(string itemName)
        {
            Item itemToRemove = items.Find(item => item.Name.Equals(itemName,
StringComparison.OrdinalIgnoreCase));
            if (itemToRemove != null)
            {
                items.Remove(itemToRemove);
                Console.WriteLine($"Removed: {itemToRemove.Name}");
            }
            else
            {
                Console.WriteLine($"Item '{itemName}' not found.");
            }
        }

        public void UpdateItem(string itemName, int stock, decimal price)
        {
            Item itemToUpdate = items.Find(item => item.Name.Equals(itemName,
StringComparison.OrdinalIgnoreCase));
            if (itemToUpdate != null)
            {
                itemToUpdate.Stock = stock;
                itemToUpdate.Price = price;
                Console.WriteLine($"Updated: {itemToUpdate.Name} - Stock: {stock}, Price:
{price}");
            }
            else
            {
                Console.WriteLine($"Item '{itemName}' not found.");
            }
        }

        public void DisplayItems()
        {
            Console.WriteLine("Current Inventory:");
            foreach (var item in items)
            {
                Console.WriteLine($"- {item.Name}: Stock = {item.Stock}, Price = {item.Price}");
            }
        }
    }
```

Main program:

```
   {
   Inventory inventory = new Inventory();


   inventory.AddItem(new Item("Apple", 100, 50));
   inventory.AddItem(new Item("Banana", 150, 30));
   inventory.AddItem(new Item("Orange", 80, 60));


   inventory.DisplayItems();


   inventory.UpdateItem("Apple", 120, 55);


   inventory.RemoveItem("Banana");


   inventory.DisplayItems();

}
```

Assignment 7: E-commerce System

Problem Statement: Design an e-commerce system that manages products and orders. Each product has a name, price, and stock quantity. Implement methods to create an order that reduces
stock quantity. Implement getters and setters to ensure that the price cannot be negative and the
stock cannot be less than zero.

```
  public class ECommerceSystem
  {
      public class Product
      {
          public string Name { get; set; }
          private double price;
          private int stockQuantity;
```

```
    public Product(string name, double price, int stockQuantity)
    {
        Name = name;
        Price = price;
        StockQuantity = stockQuantity;
    }
    public double Price
    {
        get { return price; }
        set
        {
            if (value < 0)
                throw new ArgumentException("Price cannot be negative.");
            price = value;
        }
    }

    public int StockQuantity
    {
        get { return stockQuantity; }
        set
        {
            if (value < 0)
                throw new ArgumentException("Stock quantity cannot be less than zero.");
            stockQuantity = value;
        }
    }

}


public class Order
{
    public Product OrderedProduct { get; set; }
    public int Quantity { get; set; }

    public Order(Product product, int quantity)
    {
        if (quantity <= 0)
            throw new ArgumentException("Quantity must be greater than zero.");

        OrderedProduct = product;
        Quantity = quantity;
```

```
            if (OrderedProduct.StockQuantity < Quantity)
                throw new InvalidOperationException("Not enough stock to fulfill the order.");

            OrderedProduct.StockQuantity -= Quantity;
        }

        public double CalculateTotal()
        {
            return OrderedProduct.Price * Quantity;
        }
    }

    private List<Product> products = new List<Product>();

    public void AddProduct(string name, double price, int stockQuantity)
    {
        products.Add(new Product(name, price, stockQuantity));
    }

    public void DisplayProducts()
    {
        foreach (var product in products)
        {
            Console.WriteLine($"Product: {product.Name}, Price: {product.Price}, Stock:
{product.StockQuantity}");
        }
    }
}
```

Main program:

```
{
    ECommerceSystem eCommerce = new ECommerceSystem();

    // Adding products
    eCommerce.AddProduct("Laptop", 999.99, 10);
    eCommerce.AddProduct("Smartphone", 499.99, 5);

    // Displaying products
    eCommerce.DisplayProducts();

    // Creating an order
    try
    {
        ECommerceSystem.Order order1 = new
ECommerceSystem.Order(eCommerce.products[0], 2); // Ordering 2 Laptops
        Console.WriteLine($"Order Total: {order1.CalculateTotal()}");
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    // Displaying products after the order
    eCommerce.DisplayProducts();
}
```

Assignment 8: Print Class

Problem Statement: Design a Print class that contains overloaded methods to print different types of data: a string, an integer, and an array of integers.

```
  class Print
  {
      public void Emp(string message)
      {
          Console.WriteLine(message);
      }

      public void Emp(int number)
      {
          Console.WriteLine(number);
      }

      public void Emp(int[] array)
      {
          Console.Write("[");
          for (int i = 0; i < array.Length; i++)
          {
              Console.Write(array[i]);
              if (i < array.Length - 1)
              {
                  Console.Write(",");
              }
          }
          Console.WriteLine("]");

      }
  }
```

```
Main program:
{
   Print printer = new Print();

   printer.Emp("Hello, World!");
   printer.Emp(42);
   int[] numbers = { 1, 2, 3, 4, 5 };
   printer.Emp(numbers);
}
```

Assignment 9: Rectangle Class

Problem Statement: Create a Rectangle class with overloaded methods to calculate the area. Implement methods that take either width and height or just one side length (for a square).

```
 class Rectangle
 {


     public double CalculateArea(double width, double height)
     {
        return width * height;
     }

     public double CalculateArea(double side)
     {
        return side * side;
     }
 }

 Main program:

 {
    // Create a rectangle
    Rectangle rect = new Rectangle();

    // Calculate area using width and height
    double area1 = rect.CalculateArea(4, 5);
    Console.WriteLine("Area of rectangle: " + area1);

    // Create a square
    Rectangle square = new Rectangle();

    // Calculate area using side length
    double area2 = square.CalculateArea(3);
    Console.WriteLine("Area of square: " + area2);
 }
```

Assignment 10: Time Class

Problem Statement: Create a Time class with overloaded methods to set the time. Implement methods to set the time using hours and minutes, and another method to set the time using seconds.

```
public class Time
{
   private int hours;
   private int minutes;
   private int seconds;

   public Time()
   {
      hours = 0;
      minutes = 0;
      seconds = 0;
   }



   public void SetTime(int hours, int minutes)

   {
      this.hours = hours;
      this.minutes = minutes;
      seconds = 0;
   }

   public void SetTime(int seconds)
   {
      hours = seconds / 3600;
      minutes = (seconds % 3600) / 60;
      seconds = (seconds % 3600) % 60;
   }

   public void DisplayTime()
   {
      Console.WriteLine("{0:00h}:{1:00m}:{2:00s}", hours, minutes, seconds);
   }
}
```

Main program:
```
{
    Time time1 = new Time();
    time1.SetTime(10, 30);
    time1.DisplayTime();



    Time time3 = new Time();
    time3.SetTime(7560);
    time3.DisplayTime();
}
```

Static and Instance Blocks
Assignment 11: Initializing a Static Field

Problem Statement: Create a class Bank that has a static field for the interest rate and a non-static field for the account holder's balance. Write a static constructor to initialize the interest rate to a default value and a regular constructor for setting up the account balance.
```
public class Bank
{
    private static decimal interestRate = 0.05m; // Default interest rate
    private decimal balance;

    static Bank()
    {
        Console.WriteLine("Initializing interest rate to: " + interestRate);
    }

    public Bank(decimal balance)
    {
        this.balance = balance;
    }

    public void CalculateInterest()
    {
        decimal interest = balance * interestRate;
        balance += interest;
        Console.WriteLine("Interest earned: " + interest);
        Console.WriteLine("Updated balance: " + balance);
    }
}
```

```
Main program:
{
   Bank account1 = new Bank(1000);
   account1.CalculateInterest();

   Bank account2 = new Bank(5000);
   account2.CalculateInterest();
}
```

Assignment 12: Counting Objects with Static and Instance Fields

Problem Statement: Create a class Car that counts how many instances of Car have been created using a static counter. Initialize this counter using a static constructor.

```
public class Car
{
   private static int carCount = 0;

   static Car()
   {
      Console.WriteLine("Initializing car count to: " + carCount);
   }

   public Car()
   {
      carCount++;
      Console.WriteLine("Car created. Total car count: " + carCount);
   }

   public static int GetCarCount()
   {
      return carCount;
   }
}
```

```
Main program:
{
   Car car1 = new Car();
   Car car2 = new Car();
   Car car3 = new Car();

   Console.WriteLine("Total number of cars created: " + Car.GetCarCount());
}
```

Assignment 13: Initializing Constants with Static Block

Problem Statement: Create a class MathOperations that initializes a static field representing the value of Pi. Write a static constructor to assign this value.

```
public class MathOperations
{
    private static readonly double pi = 3.14159;

    static MathOperations()
    {
        Console.WriteLine("Initializing pi to: " + pi);
    }

    public static double CalculateAreaOfCircle(double radius)
    {
        return pi * radius * radius;
    }
}
```

Main program:

```
  {
      double radius = 5;
      double area = MathOperations.CalculateAreaOfCircle(radius);
      Console.WriteLine("Area of the circle: " + area);
  }
```

Assignment 14: Initializing Configuration with Static Constructor

Problem Statement: Create a class Configuration to load system-wide settings (e.g., application name) using a static constructor. Allow individual users to set preferences using an instance constructor.

```
 public class Configuration
 {
     private static string applicationName = "My Application";
     private string userPreference;

     static Configuration()
     {
         Console.WriteLine("Loading system-wide settings...");
     }

     public Configuration(string userPreference)
     {
         this.userPreference = userPreference;
```

```
            Console.WriteLine("Setting user preference: " + userPreference);
        }

        public static string GetApplicationName()
        {
            return applicationName;
        }

        public string GetUserPreference()
        {
            return userPreference;
        }
    }
```

Main program:

```
{
    Console.WriteLine("System-wide application name: " + Configuration.GetApplicationName());

    Configuration user1Config = new Configuration("Dark Mode");
    Console.WriteLine("User 1 preference: " + user1Config.GetUserPreference());

    Configuration user2Config = new Configuration("Light Mode");
    Console.WriteLine("User 2 preference: " + user2Config.GetUserPreference());
}
```

Assignment15: Implementing and Understanding Copy Constructor

Problem Statement:
Write a C# program that implements a copy constructor. The program should:
1. Create a class with several fields.
2. Provide a constructor to initialize those fields.
3. Provide a copy constructor that allows the creation of a new object from an existing object.
4. Demonstrate how the copy constructor works by comparing objects created using it with objects created via direct assignment (which just copies references).

```
  public class Person
  {
      public string Name { get; set; }
      public int Age { get; set; }

      public Person(string name, int age)
      {
          Name = name;
          Age = age;
```

```
    }

    // Copy constructor
    public Person(Person other)
    {
        Name = other.Name;
        Age = other.Age;

    }
 }
```

Main program:

```
 {
    Person person1 = new Person("Alice", 30);


    Person person2 = person1;

    //  copy using copy constructor
    Person person3 = new Person(person1);

    person2.Name = "sam";
    person2.Age = 33;
    Console.WriteLine("Person 1: Name = " + person1.Name + ", Age = " + person1.Age);
    Console.WriteLine("Person 2: Name = " + person2.Name + ", Age = " + person2.Age);
    Console.WriteLine("Person 3: Name = " + person3.Name + ", Age = " + person3.Age);

 }
```

Assignment 16: Identifying the Need for Chained Constructors

Tasks:

1. Create a class named Car with the following:

o Fields for make, model, year, and price.

o Multiple constructors:

A constructor that initializes only the make.

A constructor that initializes make and model.

A constructor that initializes make, model, and year.

A constructor that initializes all fields: make, model, year, and price.

o Use constructor chaining to avoid duplicating the logic for initializing fields.

2. In the Main() method:

o Create several Car objects using different constructors.

o Display the details of each car to verify that all fields are initialized correctly.

```
 class Car
```

```
{
    public string Make { get; set; }
    public string Model { get; set; }
    public int Year { get; set; }
    public double Price { get; set; }

    public Car(string make)
    {
        Make = make;
    }

    public Car(string make, string model) : this(make)
    {
        Model = model;
    }

    public Car(string make, string model, int year) : this(make, model)
    {
        Year = year;
    }

    public Car(string make, string model, int year, double price) : this(make, model, year)
    {
        Price = price;
    }
```

Main program:

```
{
    Car car1 = new Car("TOYOTA");
    Car car2 = new Car("HONDA", "CIVIC");
    Car car3 = new Car("FORD", "MUSTANG", 2023);
    Car car4 = new Car("TESLA", "M3", 2014, 50000);

    Console.WriteLine("CAR1:");
    Console.WriteLine($"Make: {car1.Make}");
    Console.WriteLine($"Model:{car1.Model}");
    Console.WriteLine($"Year: {car1.Year}");
    Console.WriteLine($"Price:{car1.Price}");

    Console.WriteLine("CAR2:");
    Console.WriteLine($"Make: {car2.Make}");
    Console.WriteLine($"Model:{car2.Model}");
    Console.WriteLine($"Year: {car2.Year}");
    Console.WriteLine($"Price:{car2.Price}");
    Console.WriteLine("CAR3:");
    Console.WriteLine($"Make: {car3.Make}");
    Console.WriteLine($"Model:{car3.Model}");
    Console.WriteLine($"Year: {car3.Year}");
    Console.WriteLine($"Price:{car3.Price}");

    Console.WriteLine("CAR4:");
    Console.WriteLine($"Make: {car4.Make}");
    Console.WriteLine($"Model:{car4.Model}");
    Console.WriteLine($"Year: {car4.Year}");
    Console.WriteLine($"Price:{car4.Price}");

}
```

Assignment 17: Understanding the Need for Constructor Overloading

Problem Statement:
Write a C# program that models a Product class with overloaded constructors. The class should:
1. Provide flexibility in product initialization based on the availability of price and discount information.
2. Use constructor overloading to handle cases where only basic product information is available, as well as cases where detailed information (price and discount) is provided.
Tasks:
1. Create a class named Product with the following:
o Fields for name, price, and discount.
o Three constructors:
A constructor that initializes only the name.

A constructor that initializes name and price.
A constructor that initializes name, price, and discount.

2. Implement a method CalculateFinalPrice() that computes the final price after applying the discount (if applicable).
3. In the Main() method:
o Create different Product objects using various constructors.
o Display the details of each product, including the final price after any applicable discount.

```
class Product
{
    // Fields for name, price, and discount
    private string name;
    private decimal price;
    private decimal discount;

    // Constructor that initializes only the name
    public Product(string name)
    {
        this.name = name;
        this.price = 0; // Default price
        this.discount = 0; // Default discount
    }

    // Constructor that initializes name and price
    public Product(string name, decimal price)
    {
        this.name = name;
        this.price = price;
        this.discount = 0; // Default discount
    }

    // Constructor that initializes name, price, and discount
    public Product(string name, decimal price, decimal discount)
    {
        this.name = name;
        this.price = price;
        this.discount = discount;
    }

    // Method to calculate the final price after discount
    public decimal CalculateFinalPrice()
    {
        if (discount > 0)
        {
            return price - (price * discount / 100);
        }
        return price;
    }

    // Method to display product details
    public void DisplayProductDetails()
    {
        Console.WriteLine($"Product Name: {name}");
        Console.WriteLine($"Price: {price:C}");
        Console.WriteLine($"Discount: {discount}%");
        Console.WriteLine($"Final Price: {CalculateFinalPrice():C}");
```

```
        Console.WriteLine();
    }
}
```

Main program:

```
 {
     // Creating different Product objects using various constructors
     Product product1 = new Product("Laptop");
     Product product2 = new Product("Smartphone", 699.99m);
     Product product3 = new Product("Tablet", 299.99m, 10);

     // Display the details of each product
     product1.DisplayProductDetails();
     product2.DisplayProductDetails();
     product3.DisplayProductDetails();
 }
```

Assignment 18: Exploring Different Ways to Initialize Objects

Problem Statement:
Write a C# program that demonstrates different ways to initialize an object of a class. The class should represent a Product with properties such as Name, Price, and Category. Implement the following methods of object initialization:
1. Constructor initialization.
2. Object initializer syntax.
3. Static factory method.
4. Anonymous types.
5. Reflection.
6. Default values in constructors.

Tasks:
1. Create a class named Product with the following:
o Properties for Name, Price, and Category.
o A constructor that initializes all three properties.
o A static method to create a Product object.
o Use reflection to dynamically create a Product object.
o Implement a constructor that provides default values for the properties.
2. In the Main() method:
o Create instances of the Product class using the different initialization techniques mentioned above.
o Display the details of each product.

```
class Product
{
    public string Name { get; set; }
    public double Price { get; set; }
    public string Category { get; set; }

    public Product(string name, double price, string category)
    {
        Name = name;
        Price = price;
        Category = category;

    }

    public static Product CreateProduct(string name, double price, string category)
    {
        return new Product(name, price, category);
    }

    public static Product CreateProductByReflection(string name, double price, string category)
    {
        Type productType = typeof(Product);
        ConstructorInfo constructorInfo = productType.GetConstructor(new[] { typeof(string),
typeof(double), typeof(string) });
        return (Product)constructorInfo.Invoke(new object[] { name, price, category });
    }

    public Product() : this("No Name", 0.0, "No Category") { }
}
```

Main program:

```
 {
     // Constructor initialization
     Product product1 = new Product("Laptop", 80000, "Electronics");

     // Object initializer syntax
     Product product2 = new Product()
     {
         Name = "Phone",
         Price = 40000,
         Category = "Electronics"
     };

     // Static factory method
     Product product3 = Product.CreateProduct("Book", 500, "Books");
```

```
    // Anonymous type
    var product4 = new { Name = "Pen", Price = 50, Category = "Stationery" };

    // Reflection
    Product product5 = Product.CreateProductByReflection("Watch", 10000, "Accessories");

    // Default constructor
    Product product6 = new Product();

    Console.WriteLine("Product 1: " + product1.Name + ", " + product1.Price + ", " +
product1.Category);
    Console.WriteLine("Product 2: " + product2.Name + ", " + product2.Price + ", " +
product2.Category);
    Console.WriteLine("Product 3: " + product3.Name + ", " + product3.Price + ", " +
product3.Category);
    Console.WriteLine("Product 4: " + product4.Name + ", " + product4.Price + ", " +
product4.Category);
    Console.WriteLine("Product 5: " + product5.Name + ", " + product5.Price + ", " +
product5.Category);
    Console.WriteLine("Product 6: " + product6.Name + ", " + product6.Price + ", " +
product6.Category);
 }
```

Assignment 19: Exploring Initialization Using Tuples and Records

Problem Statement:
Write a C# program that demonstrates object initialization using tuples and records. Create a simple model for Student with properties like Name, Age, and Grade. Use tuples and records to initialize and work with this model.

Tasks:
1. Create a Student class using the record keyword with properties Name, Age, and Grade.
2. Use tuples to store and retrieve student details.
3. Create a method that accepts a tuple as a parameter and returns a Student record.
4. Display the details of the students.

```
   class Program
{
   static void Main()
   {
      // Using Tuple
      var tupleStudent = Tuple.Create("Alice", 20, 10);
```

```
        Console.WriteLine($"Tuple Student: Name: {tupleStudent.Item1}, Age:
{tupleStudent.Item2}, Grade: {tupleStudent.Item3}");

        // Using Record
        var recordStudent = new Student("Bob", 22, 12);
        Console.WriteLine($"Record Student: {recordStudent}");

        // Method to convert tuple to record
        Student ConvertTupleToRecord(Tuple<string, int, int> tuple)
        {
            return new Student(tuple.Item1, tuple.Item2, tuple.Item3);
        }

        var tupleToRecord = ConvertTupleToRecord(Tuple.Create("Charlie", 21, 11));
        Console.WriteLine($"Converted Record Student: {tupleToRecord}");
    }
}
```

Assignment 20: Shopping Cart

Problem Statement:
You need to create a Shopping Cart class that holds a list of Product objects. The Product class will be a nested class. The system should allow users to add products to the cart and display the
total price.
Tasks:
1. Create a ShoppingCart class that contains:
o A list of Product objects.
o Methods to add products and calculate the total price.
2. Create a nested Product class with properties for Name, Price, and Quantity.
3. Demonstrate adding products and displaying the total price in the Main() method.

```
class Program
    {
        class ShoppingCart
        {
            private List<Product> products;

            public ShoppingCart()
            {
                products = new List<Product>();
            }

            public void AddProduct(string name, double price, int quantity)
            {
```

```
            products.Add(new Product(name, price, quantity));
        }

        public double GetTotalPrice()

        {
            return products.Sum(p => p.Price * p.Quantity);
        }

        class Product
        {
            public string Name { get; set; }
            public double Price { get; set; }
            public int Quantity { get; set; }

            public Product(string
  name, double price, int quantity)
            {
                Name = name;
                Price = price;
                Quantity = quantity;

            }
        }
    }
  }
```

Main program:

```
 {
    ShoppingCart cart = new ShoppingCart();
    cart.AddProduct("Laptop", 80000, 1);
    cart.AddProduct("Phone", 40000, 2);
    cart.AddProduct("Book", 500, 3);

    double totalPrice = cart.GetTotalPrice();
    Console.WriteLine("Total Price: " + totalPrice);
 }
```

Assignment 21: Banking System

Problem Statement:
You need to create a Bank class that contains a list of Customer objects. Each Customer can have
multiple Account objects (nested class). Implement methods to add customers, add accounts, and
display customer account details.

Tasks:

1. Create a Bank class with:
o A list of Customer objects.
o Methods to add customers and accounts, and to display customer details.
2. Create a nested Customer class with properties for Name and a list of accounts.
3. Create a nested Account class with properties for AccountNumber and Balance.
4. Demonstrate the functionality in the Main() method.

```
class Bank
{
    private List<Customer> customers;

    public Bank()
    {
        customers = new List<Customer>();
    }

    public void AddCustomer(string name)
    {
        customers.Add(new Customer(name));
    }

    public void AddAccount(string customerName, int accountNumber, double balance)
    {
        Customer customer = customers.Find(c => c.Name == customerName);
        if (customer != null)
        {
            customer.AddAccount(accountNumber, balance);
        }
        else
        {
            Console.WriteLine("Customer not found.");
        }
    }

    public void DisplayCustomerDetails()
    {
        foreach (Customer customer in customers)
        {
            Console.WriteLine("Customer: " + customer.Name);
            foreach (Account account in customer.Accounts)
            {
                Console.WriteLine("  Account Number: " + account.AccountNumber + ", Balance: " +
account.Balance);
            }
        }
    }
    class Customer
    {
        public string Name { get; set; }
        public List<Account> Accounts { get; set; }

        public Customer(string name)
        {
            Name = name;
            Accounts = new List<Account>();
        }

        public void AddAccount(int accountNumber, double balance)
        {
            Accounts.Add(new Account(accountNumber, balance));
        }
    }

    class Account
    {
        public int AccountNumber { get; set; }
        public double Balance { get; set; }

        public Account(int accountNumber, double balance)
        {
            AccountNumber = accountNumber;
            Balance = balance;

        }
    }
}
```

Main program:
```
{
   Bank bank = new Bank();
   bank.AddCustomer("Alice");
   bank.AddCustomer("Bob");

   bank.AddAccount("Alice", 1234, 1000);
   bank.AddAccount("Alice", 5678, 2000);
   bank.AddAccount("Bob", 9012, 3000);

   bank.DisplayCustomerDetails();
}
```

Assignment 22: University System

Problem Statement:
You need to create a University class that holds a list of Department objects. Each Department can have multiple Course objects (nested class). Implement methods to add departments, add courses, and display course information.
Tasks:
1. Create a University class with:
o A list of Department objects.
o Methods to add departments and courses, and to display course details.
2. Create a nested Department class with properties for Name and a list of courses.
3. Create a nested Course class with properties for CourseName, CourseCode, and Credits.
4. Demonstrate the functionality in the Main() method.

```
      class University
   {
      private List<Department> departments;

   public University()
   {
      departments = new List<Department>();
   }

   public void AddDepartment(string name)
   {
      departments.Add(new Department(name));
   }

   public void AddCourse(string departmentName, string courseName, string courseCode, int
credits)
   {
      Department department = departments.Find(d => d.Name == departmentName);
      if (department != null)
      {
          department.AddCourse(courseName, courseCode, credits);
      }
      else
      {
          Console.WriteLine("Department not found.");
      }
   }

   public void DisplayCourseDetails()
   {
      foreach (Department department in departments)
      {
          Console.WriteLine("Department: " + department.Name);
          foreach (Course course in department.Courses)
          {
              Console.WriteLine("  Course Name: " + course.CourseName + ", Course Code: " +
course.CourseCode + ", Credits: " + course.Credits);
          }
      }
   }

   class Department
   {
      public string Name { get; set; }
      public List<Course> Courses { get; set; }

      public Department(string name)
      {
          Name = name;
          Courses = new List<Course>();
      }

      public void AddCourse(string courseName, string courseCode, int credits)
      {
          Courses.Add(new Course(courseName, courseCode, credits));
      }
   }

   class Course
   {
      public string CourseName { get; set; }
      public string CourseCode { get; set; }
      public int Credits { get; set; }

      public Course(string courseName, string courseCode, int credits)
      {
          CourseName = courseName;
          CourseCode = courseCode;
          Credits = credits;
      }
   }
}
```

Main program:

```
{
    University university = new University();
    university.AddDepartment("Computer Science");
    university.AddDepartment("Mathematics");

    university.AddCourse("Computer Science", "CS101", "Introduction to Programming", 3);
    university.AddCourse("Computer Science", "CS201", "Data Structures and Algorithms", 4);
    university.AddCourse("Mathematics", "MA101", "Calculus I", 4);
    university.AddCourse("Mathematics", "MA201", "Linear Algebra", 3);

    university.DisplayCourseDetails();
}
```