Exercise 1: Protected Keyword and Constructors in Inheritance

Lab Exercise:

1. Create a base class `Person` with protected fields `name` and `age`.
2. Add a constructor to `Person` that initializes these fields.
3. Create a derived class `Student` that inherits from `Person` and has an additional field `studentId`.
4. Add a constructor to `Student` that initializes all fields, including those in the base class.
5. Add a method in `Student` to display all the information.

Solution:

```csharp
public class Person
{
    protected string name;
    protected int age;

    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
}

public class Student : Person
{
    private int studentId;

    public Student(string name, int age, int studentId) : base(name, age)
    {
        this.studentId = studentId;
    }
```

```
    public void DisplayInfo()
    {
        Console.WriteLine($"Name: {name}, Age: {age}, Student ID: {studentId}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Student student = new Student("Alice", 20, 12345);
        student.DisplayInfo();
    }
}
```

Exercise 2: Casting Between Reference Types

Lab Exercise:
1. Create a base class `Animal` and a derived class `Dog` that inherits from `Animal`.
2. Demonstrate upcasting and downcasting between the types.
3. Add a method `MakeSound` in both classes and demonstrate polymorphism.

Solution:
```csharp
public class Animal
{
    public virtual void MakeSound()
    {
        Console.WriteLine("Animal makes a sound");
```

```
    }
}

public class Dog : Animal
{
    public override void MakeSound()
    {
        Console.WriteLine("Dog barks");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Animal animal = new Dog();  // Upcasting
        animal.MakeSound();         // Dog barks

        Dog dog = (Dog)animal;      // Downcasting
        dog.MakeSound();            // Dog barks
    }
}
```

Exercise 3: Static and Dynamic Binding

Lab Exercise:

1. Create a base class `Shape` with a method `Draw()`.
2. Create a derived class `Circle` that overrides `Draw()`.
3. Demonstrate static and dynamic binding using method calls.

Solution:

```csharp
public class Shape
{
    public virtual void Draw()
    {
        Console.WriteLine("Drawing a shape");
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a circle");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Shape shape = new Circle();
        shape.Draw();  // Dynamic binding: Drawing a circle

        Shape shape2 = new Shape();
        shape2.Draw();  // Static binding: Drawing a shape
    }
}
```

Exercise 4: Abstract Class & Methods

Lab Exercise:

1. Create an abstract class `Vehicle` with an abstract method `Drive()`.

2. Create a derived class `Car` that implements the `Drive()` method.

3. Demonstrate creating an instance of `Car` and calling the `Drive()` method.

Solution:

```csharp
public abstract class Vehicle
{
    public abstract void Drive();
}

public class Car : Vehicle
{
    public override void Drive()
    {
        Console.WriteLine("Car is driving");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Vehicle myCar = new Car();
        myCar.Drive();
    }
}
```

Exercise 5: Object Class as Parent

Lab Exercise:

1. Create a class `Person` that overrides the `ToString()` method from the `Object` class.
2. Create an instance of `Person` and display its string representation using `ToString()`.

Solution:

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

    public override string ToString()
    {
        return $"Name: {Name}, Age: {Age}";
    }
}

class Program
{
    static void Main(string[] args)
    {
        Person person = new Person { Name = "John", Age = 30 };
        Console.WriteLine(person.ToString());
    }
}
```

Exercise 6: Single Inheritance

Lab Exercise:

1. Create a base class `Employee` with properties `Name` and `Salary`.

2. Create a derived class `Manager` that adds an additional property `Department`.

3. Demonstrate creating an instance of `Manager` and displaying all its properties.

Solution:

```csharp
public class Employee
{
    public string Name { get; set; }
    public decimal Salary { get; set; }
}

public class Manager : Employee
{
    public string Department { get; set; }

    public void DisplayInfo()
    {
        Console.WriteLine($"Name: {Name}, Salary: {Salary}, Department: {Department}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Manager manager = new Manager { Name = "Alice", Salary = 90000, Department = "HR" };
        manager.DisplayInfo();
    }
```

}
```

### Exercise 7: Multi-level Inheritance

**Lab Exercise:**

1. Create a base class `LivingBeing` with a method `Respire()`.
2. Create a derived class `Animal` that inherits from `LivingBeing`.
3. Create another derived class `Dog` that inherits from `Animal` and adds a method `Bark()`.
4. Demonstrate calling methods from all levels of the inheritance hierarchy.

**Solution:**

```csharp
public class LivingBeing
{
    public void Respire()
    {
        Console.WriteLine("Living being is respiring");
    }
}

public class Animal : LivingBeing
{
    public void Eat()
    {
        Console.WriteLine("Animal is eating");
    }
}

public class Dog : Animal
{

```
    public void Bark()
    {
        Console.WriteLine("Dog is barking");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Dog dog = new Dog();
        dog.Respire();  // Living being is respiring
        dog.Eat();      // Animal is eating
        dog.Bark();     // Dog is barking
    }
}
```

Exercise 8: `var` and `dynamic` Keyword

Lab Exercise:

1. Create a method that returns an integer and assign the result to a `var` variable.

2. Create another method that returns a dynamic type and demonstrate changing its type at runtime.

Solution:

```csharp
class Program
{
    static void Main(string[] args)
    {
```

```
        var number = GetNumber();
        Console.WriteLine($"Number: {number}, Type: {number.GetType()}");

        dynamic dynamicVar = "Hello";
        Console.WriteLine($"DynamicVar: {dynamicVar}, Type: {dynamicVar.GetType()}");

        dynamicVar = 10;
        Console.WriteLine($"DynamicVar: {dynamicVar}, Type: {dynamicVar.GetType()}");
    }

    static int GetNumber()
    {
        return 42;
    }
}
```

Exercise 9: Stopping Inheritance using `sealed` Keyword

Lab Exercise:

1. Create a base class `BaseClass` and a derived class `DerivedClass`.

2. Mark `DerivedClass` as `sealed` to prevent further inheritance.

3. Attempt to create another class that inherits from `DerivedClass` and observe the compile-time error.

Solution:

```csharp
public class BaseClass
{
    public virtual void Display()
    {
```

```
        Console.WriteLine("Base class display");
    }
}

public sealed class DerivedClass : BaseClass
{
    public override void Display()
    {
        Console.WriteLine("Derived class display");
    }
}

// This will cause a compile-time error
// public class AnotherClass : DerivedClass
// {
// }

class Program
{
    static void Main(string[] args)
    {
        DerivedClass obj = new DerivedClass();
        obj.Display();
    }
}
```

Exercise 10: Factory Method Pattern

Lab Exercise:

1. Create an abstract class `Document` with an abstract method `CreatePage()`.

2. Create concrete classes `Resume` and `Report` that inherit from `Document` and implement the `CreatePage()` method.

3. Demonstrate using the factory method to create instances of `Resume` and `Report`.

Solution:

```csharp
public abstract class Document
{
    public abstract void CreatePage();
}

public class Resume : Document
{
    public override void CreatePage()
    {
        Console.WriteLine("Creating a resume page");
    }
}

public class Report : Document
{
    public override void CreatePage()
    {
        Console.WriteLine("Creating a report page");
    }
}

class Program
{
    static void Main(string[] args)
    {
```

```
        Document doc1 = new Resume();
        doc1.CreatePage();

        Document doc2 = new Report();
        doc2.CreatePage();
    }
}
```