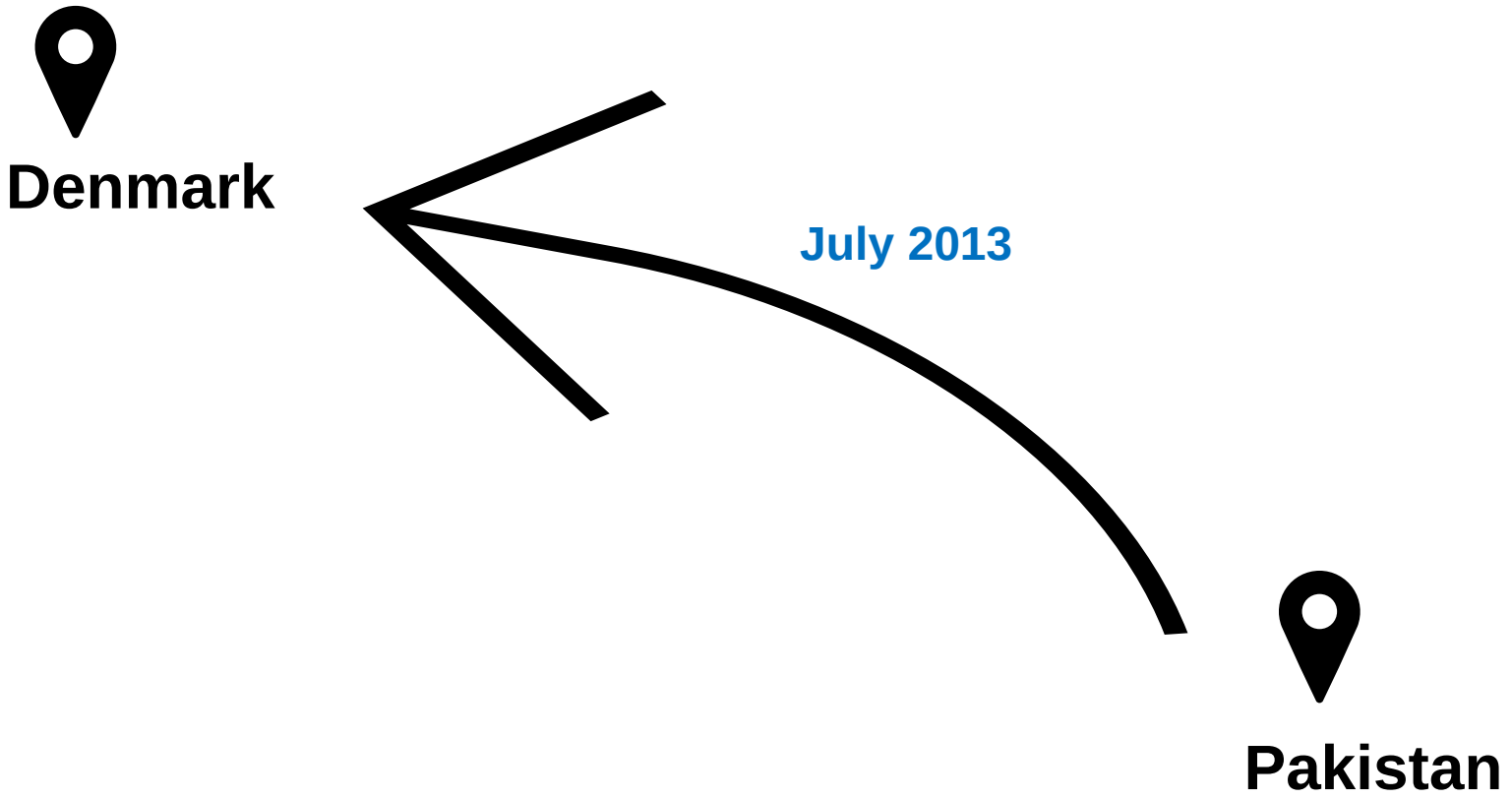
ADVANCED OBJECT-ORIENTED PROGRAMMING

AISHA UMAIR
aiu@mmm.sdu.dk

SDU

ABOUT ME

AISHA UMAIR



SDU

ABOUT ME

AISHA UMAIR

- PhD. in Software Engineering from Syddansk Universitet (SDU)

  PhD Project: Coordination Protocol for System of Cyber-Physical Systems

- Associate Professor, SDU Software Engineering, MMMI

Research Interests:

- Multi-Objective Optimization
- Cyber-Physical Systems
- Agent Based Systems
- Social Welfare Metrics
- Educational Research: Scalable Teaching

SDU

Responsibilities

- Teaching
  - Scientific Methods (Master: 1st Semester)
  - Advanced Object-Oriented Programming (Bachelor: 2nd Semester)
- Supervision
  - MS Thesis supervision
  - BS SE/ST Final Year Projects' supervision
  - BS SE/ST Internship Projects' supervision
- Administrative Tasks
  - BS Semester-1 SE/ST Coordination

MigratingMonolith to Microservice Architecture

Identifying similar or identical images using reverse image search

Automated Sales Optimisation using Google Analytics Data

Topic Classification using Machine Learning

Estimating Freezers alarm pressure based on historical weather data using Machine Learning

SDU

MAIN CONTACT

- Aisha Umair (aiu@mmmi.sdu.dk)
- If you have any queries outside the lecture, please send an e-mail.

SDU

Lecture 1: Circular Arrays and Design Patterns

Agenda

1. Introduction to the Course
2. Revision of 1st Semester Topics:
   - *Array*
   - *Inheritance and Polymorphism*
   - *Abstract Class and Interface*
3. Topics of the day
   - *Circular Array*
   - *Singleton* and Facade Design Patterns

Course Introduction

Knowledge/Skills acquired after the course:

- Explain and apply **polymorphism** in the implementation of interfaces and **inheritance** from abstract and non-abstract classes
- Implement object-oriented applications based on **design specification**
- Explain and apply **design patterns (Singelton, Facade)** to write object-oriented programs
- Create **desktop GUI application** that responds to user events.
- Describe Character-based and Binary-based File I/O and make informed choices about the appropriate approach depending on type of the data.
- Describe **simple recursive algorithms** and implement methods that use recursion.
- Describe, create and apply **thread synchronization mechanisms** in multi-threaded applications

Knowledge/Skills acquired after the course:

- Describe the C# Collections (**List, Set**, **Stack, Queue** and **Dictionary** and their implementation) and use appropriate data structures in building applications.
- Explain , analyse and implement **handling of exceptions** in own classes.
- Describe **execution time and Big-O notation** and compare **simple search** and **sorting algorithms**
- Describe software testing and create **unit tests** on their own classes

SDU

Point-giving Activities (Graded)

There will be two point-giving activities (graded) during the semester.

1. Activity 1: 5th March [Duration: 14:00 – 16:30] (U301)
2. Activity 2: 9th April [Duration: 10:00 – 12:30] (U1)

- Activities will be conducted by the instructors
- The purpose will be to get the bonus points and also to get an idea about how the final exam will look like.

SDU

Final Exam (100%+ 10% PAs)

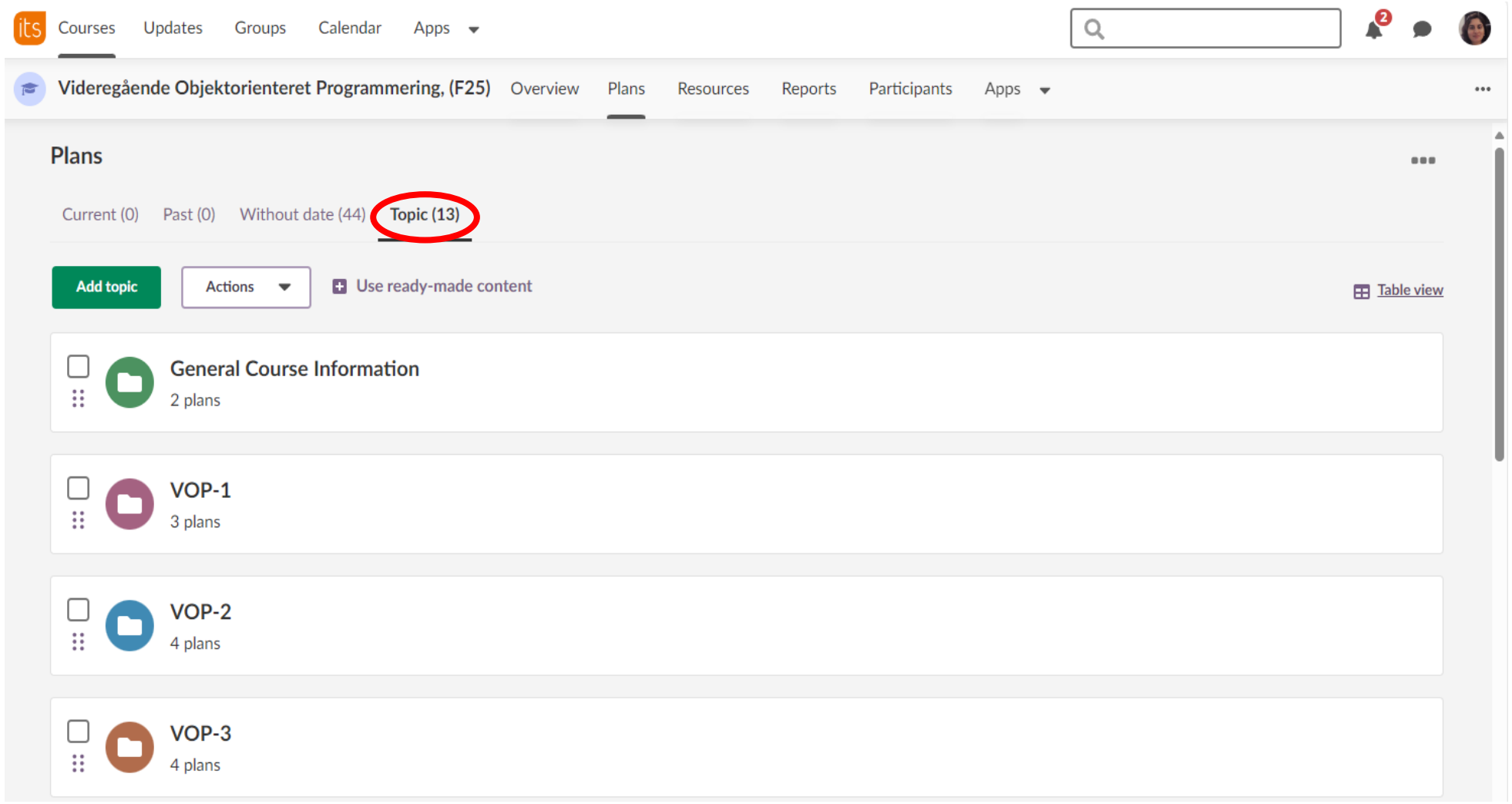
- A zipped C# project with a pre-existing codebase and a ReadMe file comprising a list of programming tasks and the set of instructions to complete those tasks.
- You will be allowed to use the C# books which I uploaded on itslearning + cheat sheet for Avalonia (provided by us)

SDU

- Post announcements (Overview Tab)

itslearning

- Preparation Material, Lecture Slides, Exercises (Plans Tab)



SDU

VOP-1

DESCRIPTION

Introduction to the course + Repetition

Create plan | Actions | Sort by: Your own sorting | Use ready-made content | Table view

VOP-1 (Preparation) VOP-1

Revision of first semester topics

1 resource

VOP-1 (Lecture) VOP-1

1. Introduction to the course 2. Repetition from 1st Semester 3. Circular Array 4. Singleton and Facade Pattern

2 resources

VOP-1 (Instructor Hours) VOP-1

Practice Exercises (Polymorphism, Singelton and Facade pattern)

1 resource

itslearning

SDU

Revision of 1st Semester Topics

SDU

MCQs Quiz

Go to Plans -> VOP-1 -> VOP-1 (Lecture) -> C# Revision

You have 15 min to complete the quiz

You can do it individually or in group ☺

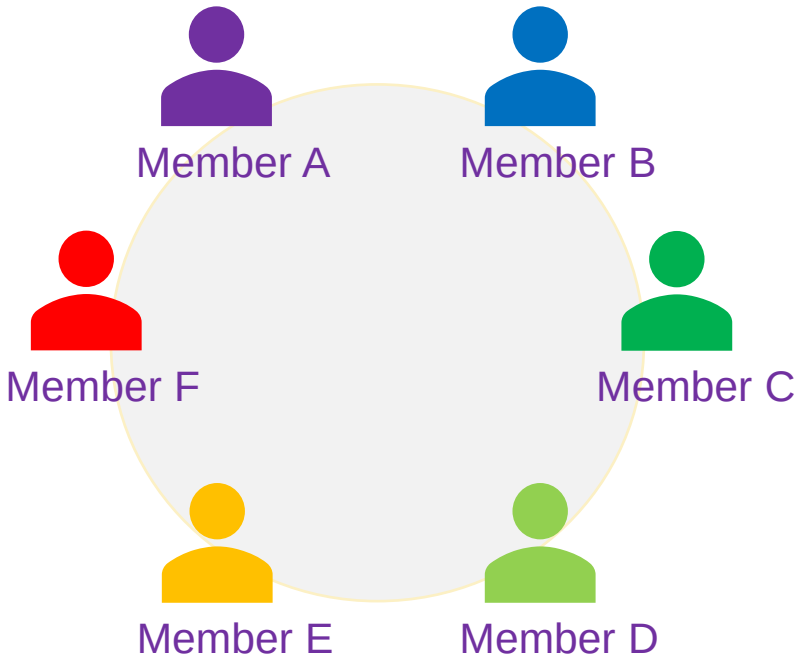
Break (10 Minutes)

Circular Arrays

https://www.geeksforgeeks.org/circular-array/

Circular Arrays

- An array is a circular, if we consider the first element to be the next of the last element.



For example

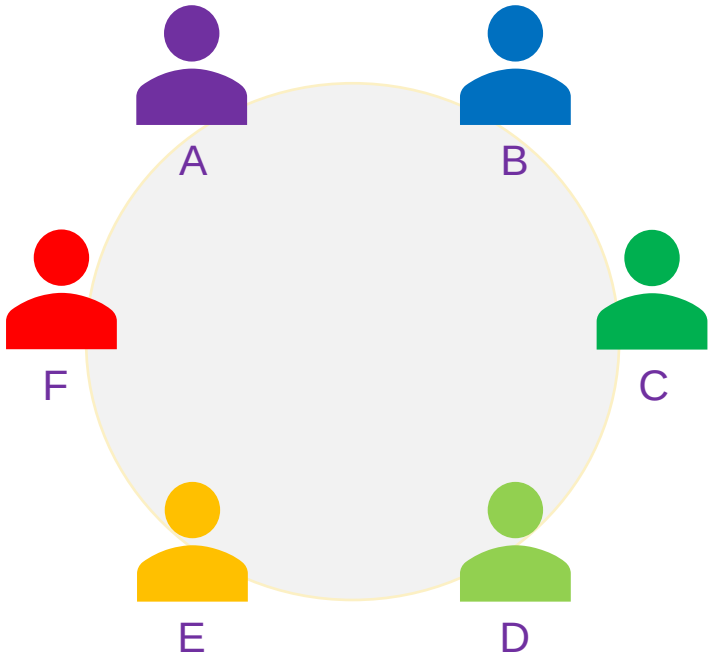
- Consider 6 people with names A, B, C, D, E, F sitting in a circular table,
- Starting from D, estimate all people sitting starting from D.

SDU

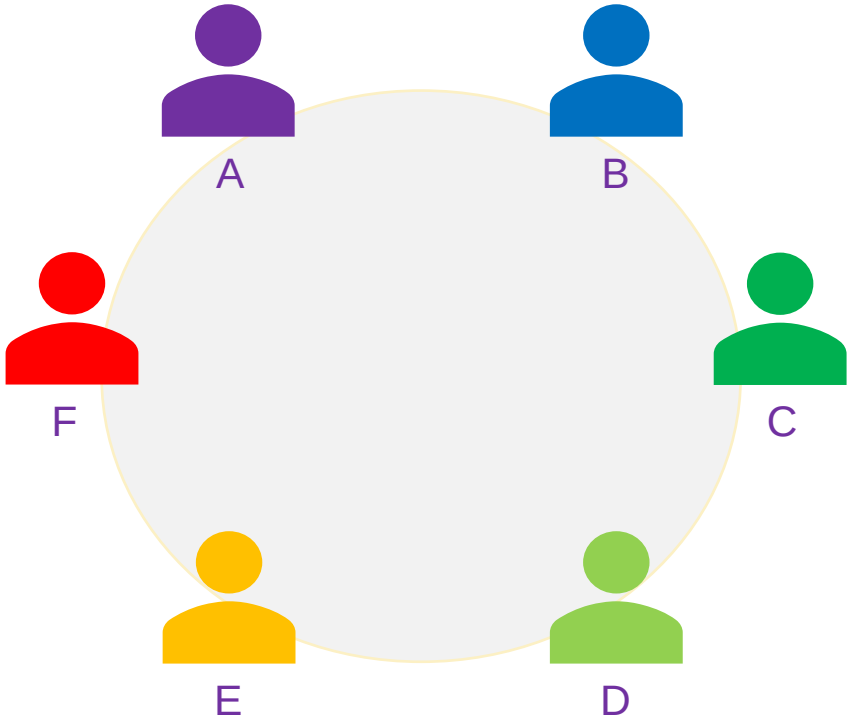
Circular Arrays

- We can use the modulo operator (%)
- A modulo operator `(%)` gives the remainder of a division `(÷)` operator
- It can be used to compute a `circularIndex` as follows

```
circularIndex = index % length
```



Circular Arrays



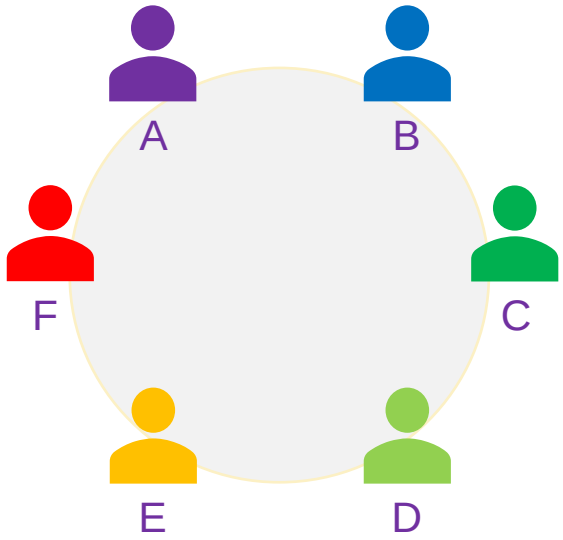
- `circularIndex = index % length`
- E.g. `char[] a =`

0	1	2	3	4	5
A	B	C	D	E	F

- Given that `a.Length = 6`
- We compute `circularIndex` of array `a`:
  - `0 % 6 = 0`
  - `1 % 6 = 1`
  - `2 % 6 = 2`
  - `3 % 6 = 3`
  - `4 % 6 = 4`
  - `5 % 6 = 5`
  - `6 % 6 = 0`
  - `7 % 6 = 1`
  - `8 % 6 = 2`
  - .
  - .
  - .

SDU

Circular Arrays




```
// C# program to demonstrate use of circular
public class CircularArray {

    // function to print circular list starting from given index ind.
    public static void print(char[] a, int n, int ind)
    {
        // print from ind-th index to (n+i)th index.
        for (int i = ind; i < n + ind; i++)
            Console.Write(a[(i % n)] + " ");
    }
    // driver code
    public static void Main()
    {
        char[] a = new char[] { 'A', 'B', 'C', 'D', 'E', 'F' };
        int n = 6;
        print(a, n, 3);
    }
}
```

In-Class Revision (5 Minutes)
Slides (24-30)

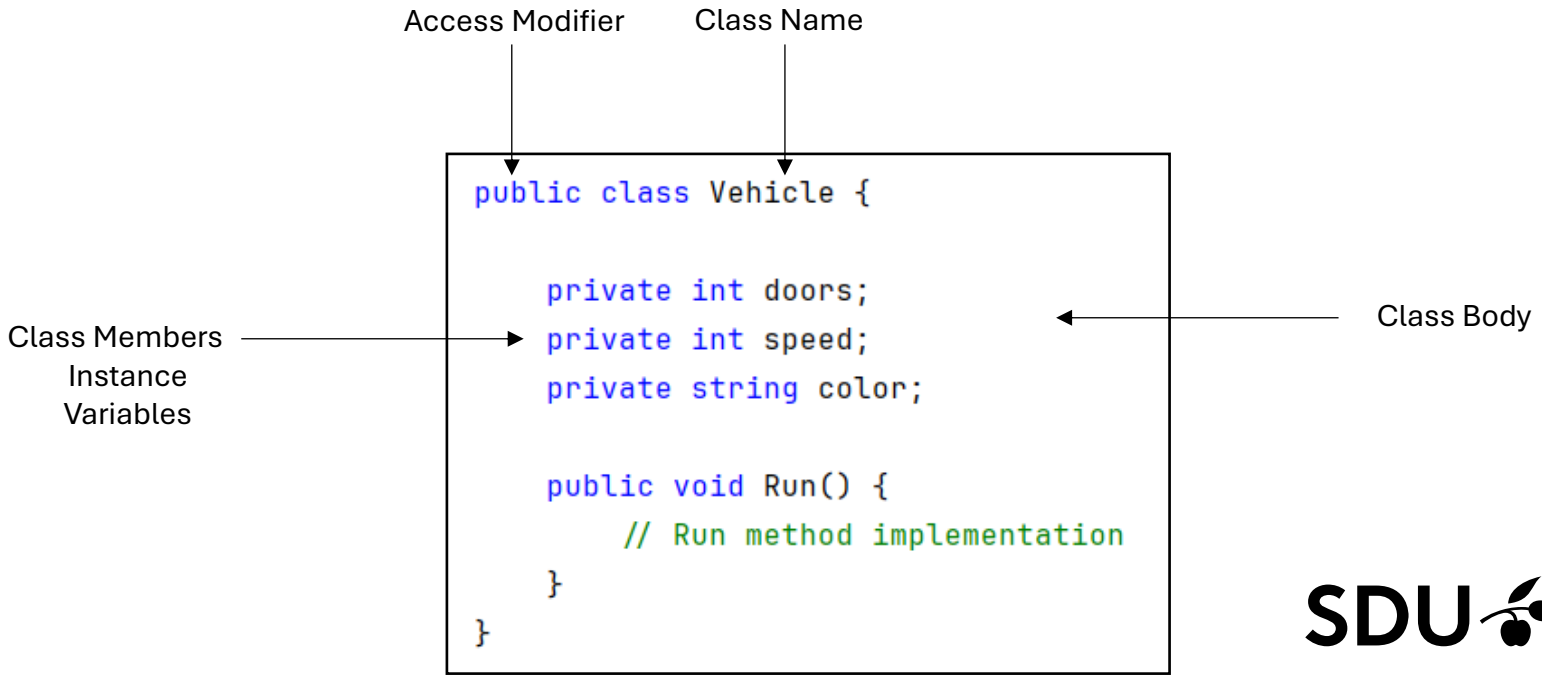
Fields/Properties in Class Definition (revision)

- Instance variables
- Class variables

SDU

Instance Variables (revision)

- Variables associated with each object
- A separate value for each instance of a class
- For example, doors, speed, color etc.



Class Variables/Static Variables (revision)

- Also called as static fields
- Declared using the keyword static
- Exists even if no object has been created
- Shared among all objects of a class
- If value is changed, it is reflected for all objects.

```
class A
{
    static int i;       // Class Variable

    int k;              // Instance Variable

    static string s1;   // Class Variable

    string s2;          // Instance Variable

}
```

SDU

Methods in Class Definition (revision)

- Instance Methods
- Class Methods

SDU

Instance Methods (revision)

- Can execute when objects exist

Example: public double CalculateArea()

Method of class Shape

```
Shape s = new Shape();
Console.WriteLine(s.CalculateArea());
```

instance method is called using object name

SDU

Class Methods (revision)

- Execute class methods even when no objects of a class exist

Example:

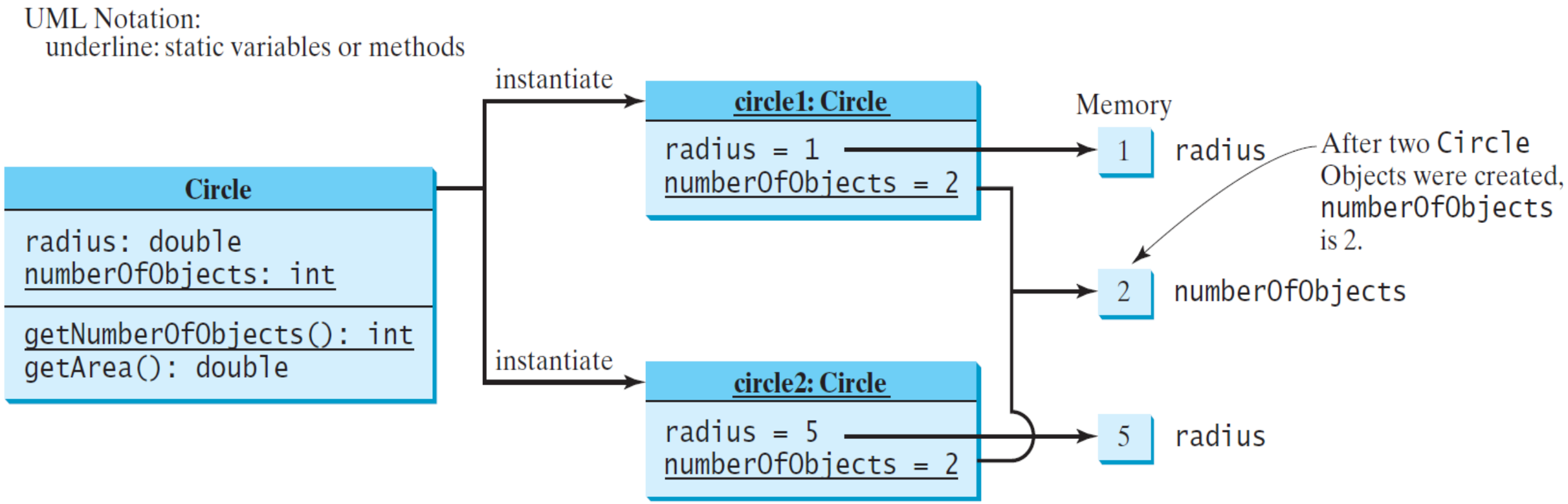
- Built in methods in standard class Math

```
double rootPi = Math.Sqrt(Math.PI);
```


class method is called using class name

SDU

Static/Instance Variables and Methods (revision)




```
Console.WriteLine(circle1.getArea());
Console.WriteLine(circle2.getArea());
Console.WriteLine(Circle.getNumberOfObjects());
```

Instance methods

Class method

SDU

Static/Instance Variables and Methods

1. Console.WriteLine(“hello”);
2. Console.WriteLine(Math.PI);
3. Console.WriteLine(Math.SQRT(25));
4. Console.WriteLine(student1.GetName());

Can you identify **class/instance** *variable*

or

class/instance *method* in the above statements ?

SDU

SDU

Singleton and Facade Design Patterns

https://csharpindepth.com/articles/singleton
https://dotnettutorials.net/lesson/singleton-design-pattern/
https://dotnettutorials.net/lesson/facade-design-pattern/#google_vignette

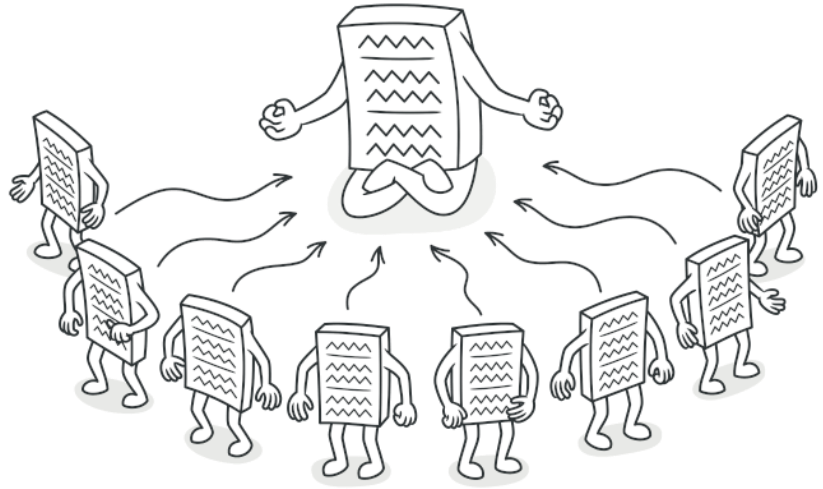
Design Patterns and Types

Design patterns are solutions to general problems that software developers faced during software development.

1. **Creational**: These patterns are designed for class instantiation.
2. **Structural**: These patterns are designed with regard to a class's structure and composition.
3. **Behavioral**: These patterns are designed depending on how one class communicates with others.

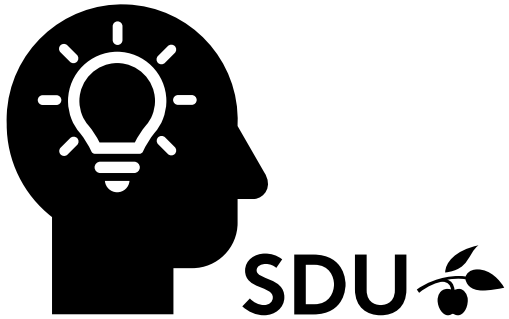
SDU

Singleton Pattern

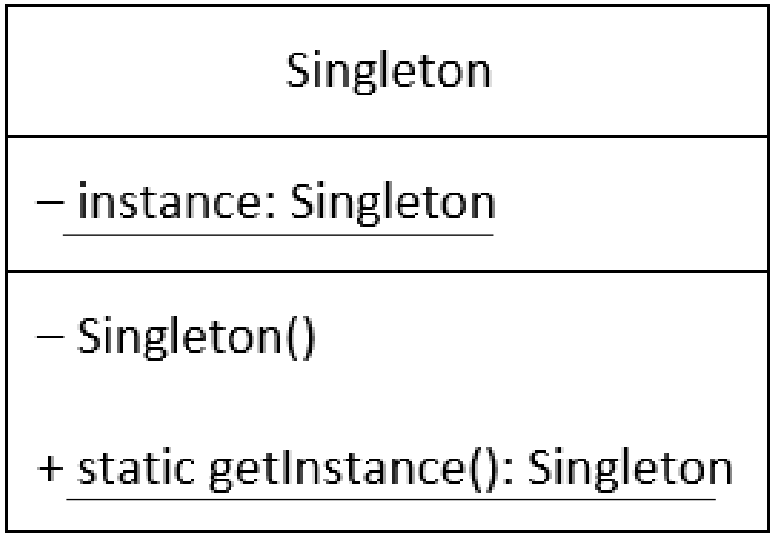


- The singleton pattern is a creational pattern.
- This pattern involves a single class and it ensures that only a **single object gets created**.
- It provides a **way to access its only instance** and **restricts instantiating** the class from outside.
- We need three things to create a Singleton design pattern

Any idea, how can you use C# language constructs to implement Singleton pattern???



Singleton Pattern

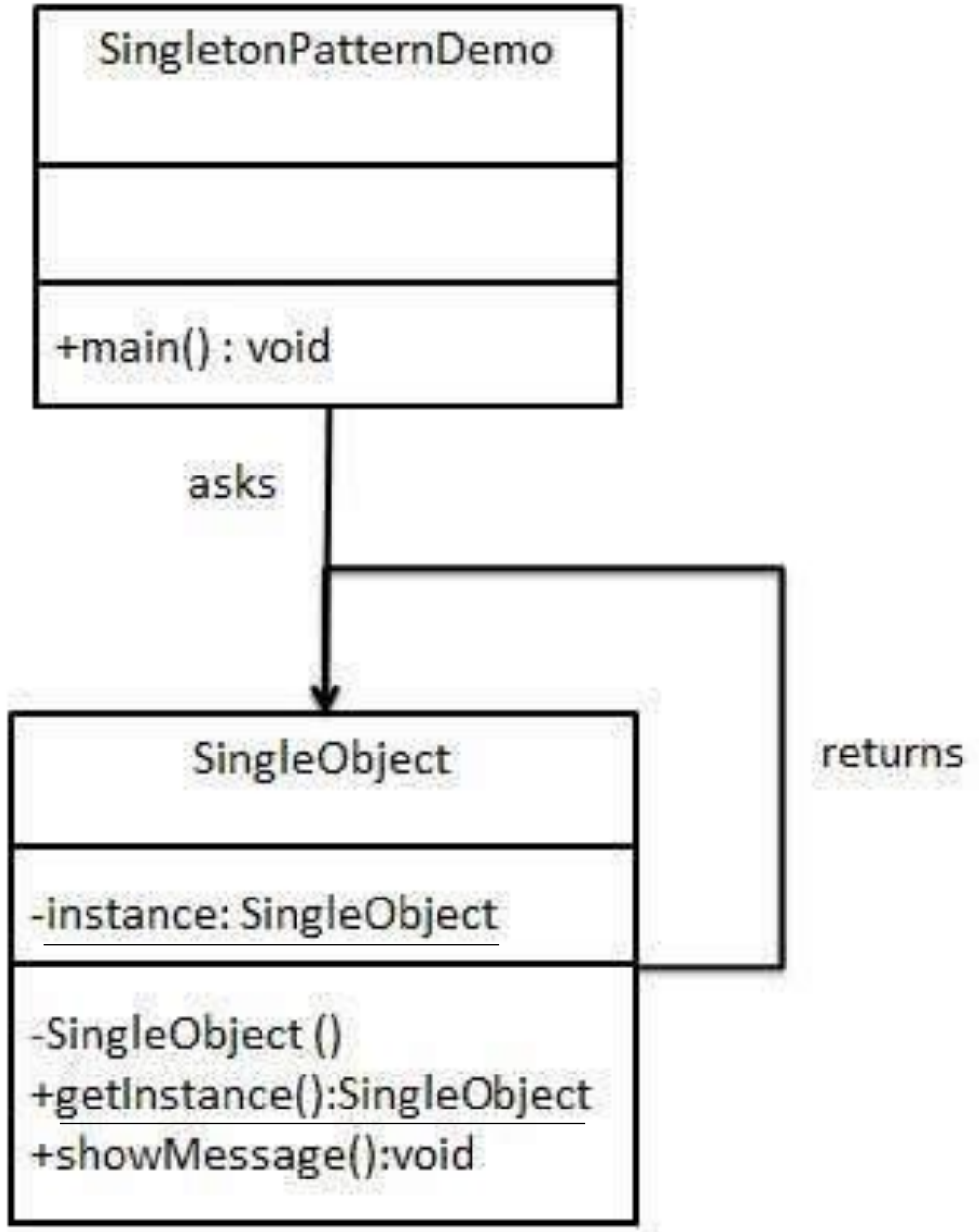


Singleton Class Diagram

1. **static member:** it contains the instance of the Singleton class.
2. **private constructor:** It will prevent to instantiate the Singleton class from outside the class.
3. **public static method:** This provides the global point of access to the Singleton object.

SDU

Singleton Pattern



Singleton Pattern

Step 1

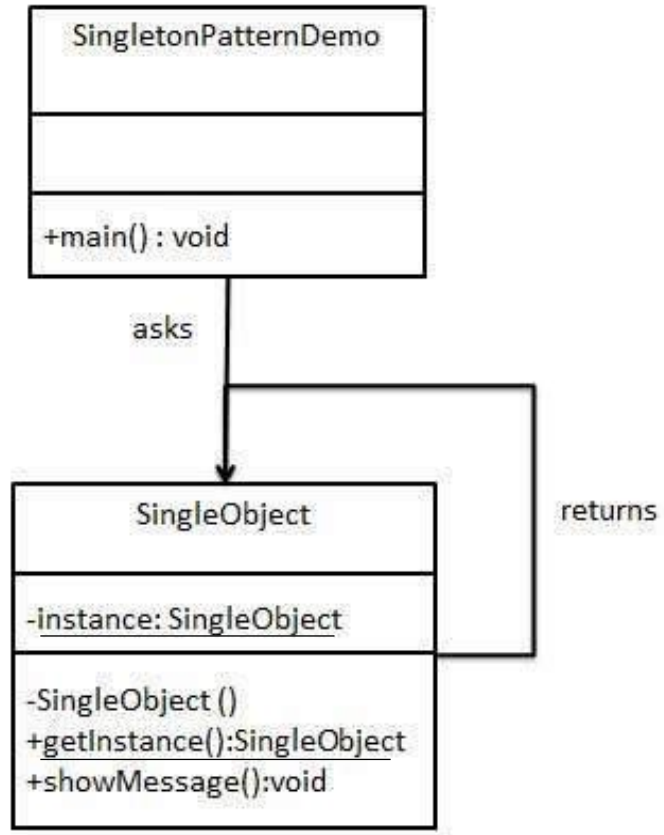
Create a Singleton Class.

```
public class SingleObject {
    private static SingleObject _instance = new SingleObject();

    private SingleObject(){}

    public static SingleObject GetInstance()
    {
        return _instance;
    }

    public void ShowMessage()
    {
        Console.WriteLine("Hello World!");
    }
}
```

static member + instantiation

private constructor

static method



SDU

Singleton Pattern

```
//illegal construct
//Compile Time Error: The constructor SingleObject() is not visible
//SingleObject object = new SingleObject();
```

Step 2

Get the only object from the singleton class.

```
public class SingletonPatternDemo {
    public static void Main(string[] args)
    {
        SingleObject singleObject = SingleObject.GetInstance();

        singleObject.ShowMessage();
    }
}
```

Step 3

Verify the output.

```
Hello World!
```

SDU

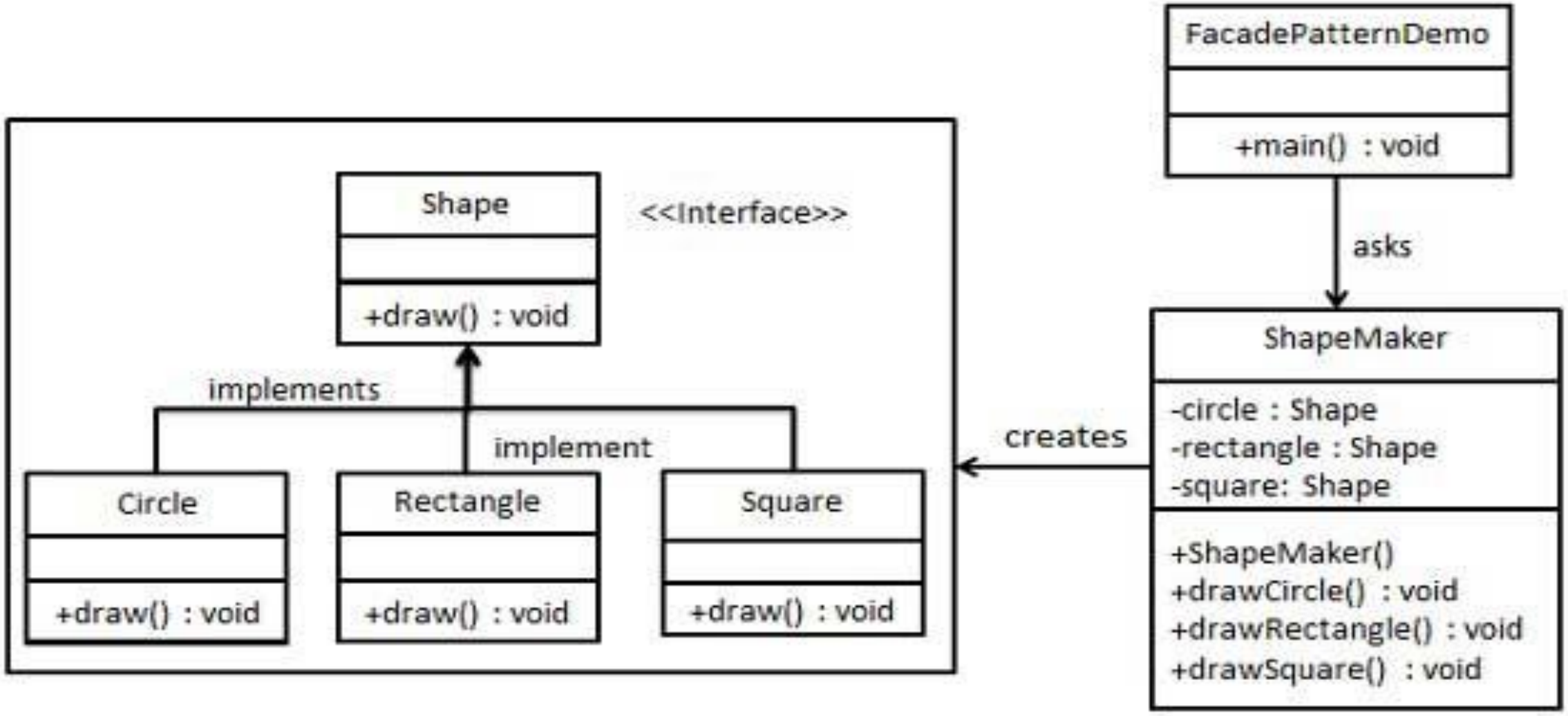
Singleton Pattern: Application

- Singleton pattern is mostly used in multi-threaded and database applications.

SDU

Facade Pattern

- The Facade pattern is a structural pattern
- A Facade hides the complexities of a system and it provides an interface to the client.



Step 1

Create an interface.

Shape.cs

```
public interface IShape
{
    void Draw();
}
```

Step 2

Create concrete classes implementing the same interface.

Rectangle.cs

```
public class Rectangle : IShape {
    public void Draw() {
        Console.WriteLine("Rectangle::Draw()");
    }
}
```

Square.cs

```
public class Square : IShape {
    public void Draw() {
        Console.WriteLine("Square::Draw()");
    }
}
```

Circle.cs

```
public class Circle : IShape {
    public void Draw() {
        Console.WriteLine("Circle::Draw()");
    }
}
```

Step 3

Create a facade class.

ShapeMaker.cs

```
public class ShapeMaker {
    private IShape _circle;
    private IShape _rectangle;
    private IShape _square;

    public ShapeMaker() {
        _circle = new Circle();
        _rectangle = new Rectangle();
        _square = new Square();
    }
    public void DrawCircle() {
        _circle.Draw();
    }
    public void DrawRectangle() {
        _rectangle.Draw();
    }
    public void DrawSquare() {
        _square.Draw();
    }
}
```


SDU

Step 4

Use the facade to draw various types of shapes.

FacadePatternDemo.cs

```
public class FacadePatternDemo
{
    public static void Main(string[] args)
    {
        ShapeMaker shapeMaker = new ShapeMaker();

        shapeMaker.DrawCircle();
        shapeMaker.DrawSquare();
        shapeMaker.DrawRectangle();
    }
}
```

Step 5

Verify the output.

```
Circle::Draw()
Square::Draw()
Rectangle::Draw()
```


SDU