Abstraction - core of the bridge design pattern and defines the crux. Contains a reference to the implementor.

Concrete Implementation - Implements the implementor by providing the concrete implementation.

Implementor - It defines the interface for implementation classes. This interface doesnot need to correspond directly to the abstraction interface and can be very different.

Concrete Abstraction - Extends the abstraction takes the finer detail one level below.

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Develop a music streaming application that can play music from various sources, such as local files, online streaming services, and radio stations. Implement the following design patterns to achieve this functionality:

Adapter Pattern: Adapt different music sources to a common interface.

Bridge Pattern: Decouple the music playback functionality from the music source.

Decorator Pattern: Add additional features (e.g., equalizer, volume control) to the music playback.

Your task is to design and implement the application using these design patterns to ensure flexibility, scalability, and maintainability.

Procedure/Program:

```
public class BridgePatternDemo {
public static void main (String[] args) {
DrawAPI redCircleAPI=new RedCircle();
Shape redCircle=new Circle(5, 10, 15, redCircleAPI);
  redCircle.draw();
DrawAPI greenCircleAPI=new GreenCircle();
Shape greenCircle=new Circle(8, 25, 30, greenCircleAPI);
   greenCircle.draw();
   }
}
public class Circle extends Shape{
  private int radius;
  private int x;
```

```
private int y;
public Circle(int radius, int x, int y, DrawAPI
                                    drawAPI) {
    super(drawAPI);
    this.radius = radius;
    this.x = x;
    this.y = y;
} public void draw() {
    drawAPI.drawCircle(radius, x, y);
    }
}
public interface DrawAPI {
    void drawCircle(int radius, int x, int y);{ }
public class GreenCircle implements DrawAPI
{
public void drawCircle(int radius, int x, int y) {
System.out.println("Drawing green circle with
radius" + radius + " at ("+x+", "+y+")");
    }
}
```

```
public abstract class Shape {
    protected DrawAPI drawAPI;
    protected Shape (DrawAPI drawAAI) {
        this.drawApI = drawAPI;
    }
    public abstract void draw ();
}
```

		Student ID	
Experiment#		Student Name	
Date			

✓ **Data and Results:**

Playing music from different sources:
Playing music from local file.
Playing music from online streaming service.
Playing music from radio station.

✓ **Analysis and Inferences:**

Adapter pattern used to adapt different music sources to a common interface. The Bridge pattern is employed by defining the MusicPlayer interface and providing BasicMusic Player and AdvancedMusicPlayer.

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab)

1) State at which situation that we are in need of Bridge Design Pattern.

Bridge design pattern is used when we need to decouple an abstraction from its implementation so that the two can vary independently.

2) Illustrate the difference between Decorator and Bridge pattern.

Decorator pattern focuses on adding responsibilities to objects dynamically and extending functionality.
Bridge pattern focuses on separating an abstraction from its implementation so that both can evolve independently.

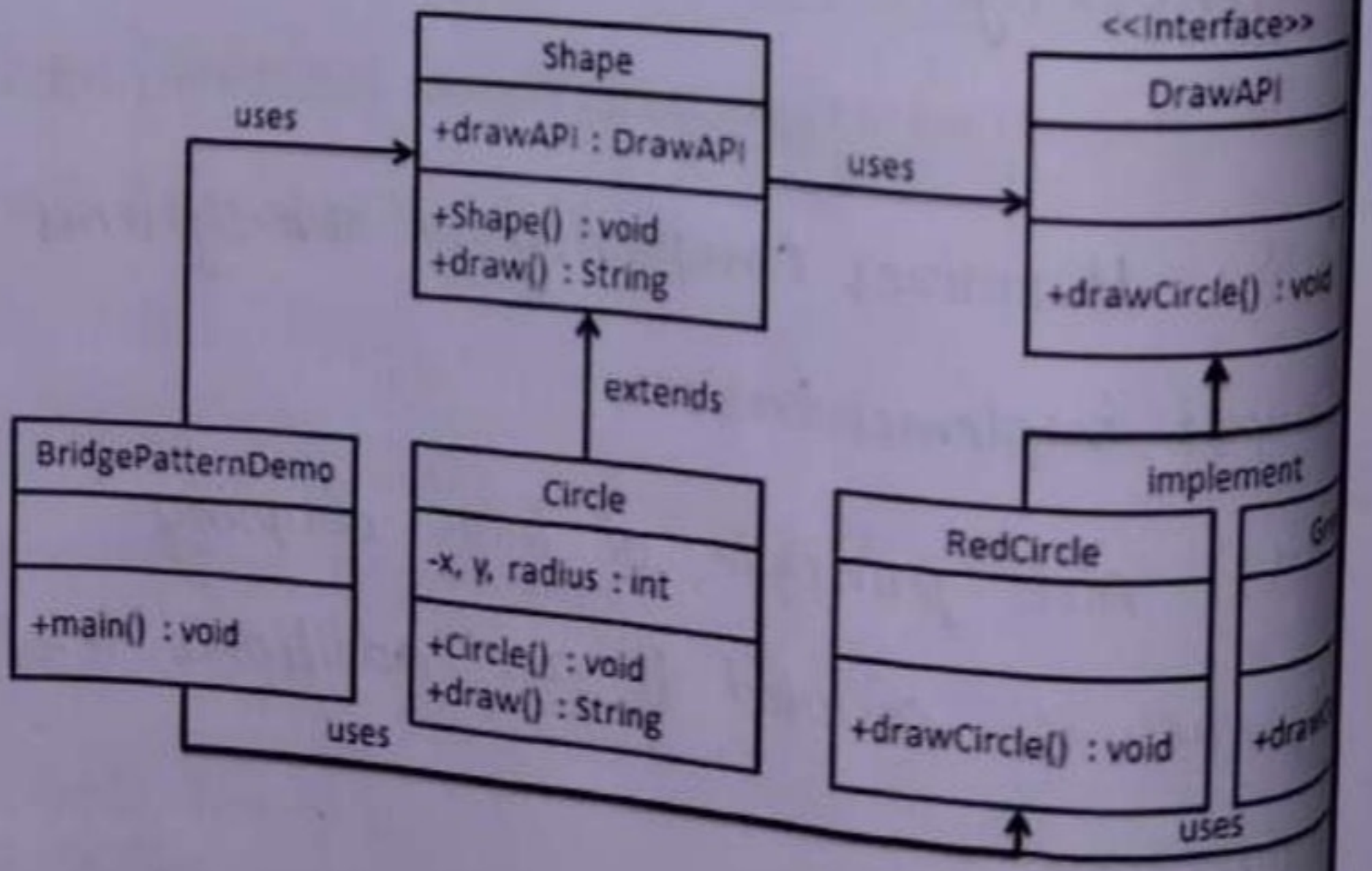
3) Discuss the Pros and Cons of Facade Design Pattern.

Pros: Minimizes complexity of sub-systems
Complex implementation

Cons: Aids principle of loose coupling
Approach is coupled to an additional level of indirection.

Post-Lab:

1) Implement the DrawAPI interface which is acting as a bridge implementer a concrete classes RedCircle, GreenCircle implementing the DrawAPI interfa is an abstract class and will use object of DrawAPI. BridgePatternDemo, ou class will use Shape class to draw different coloured circle.



Procedure/Program:

```
public interface DrawAPI {
void drawCircle(int radius, int x, int y);
}
public class RedCircle implements DrawAPI {
public void drawCircle(int radius, int x, int y) {
System.out.println("Drawing Circle [ color: red, radius:
" +radius+", x:" + x +", y: "+y+"]");
    }
}
public class GreenCircle implements DrawAPI {
public void drawCircle(int radius, int x, int y) {
System.out.println("Drawing Circle [color: green, radius:
" +radius+", x:" + x +", y: "+y+"]");
    }
}
  public abstract class Shape {
protected DrawAPI drawAPI;
protected Shape(DrawAPI drawAPI) {
```

```
this.drawAPI = drawAPI; }

public abstract void draw(); }

public class Circle extends Shape {
private int x, y, radius;
public Circle (int x, int y, int radius, DrawAPI
                                    drawAPI) {
   super(drawAPI);
   this.x = x;
   this.y = y;
   this.radius = radius;
}
  public void draw(){
drawAPI.drawCircle (radius, x, y);
   }
} public class BridgePatternDemo {
public static void main (string[] args) {
Shape redCircle = new Circle (100, 100, 10);
Shape greenCircle = new Circle (100, 100, 10);
   redCircle.draw();
   greenCircle.draw(); } }
```

Experiment#		Student ID	
Date		Student Name	

✓ **Data and Results:**

Drawing Circle[color: red, radius: 10, x: 100, y: 100]

Drawing Circle [color: green, radius: 10, x: 100, y: 100]

✓ **Analysis and Inferences:**

Bridge Pattern allows to separate the abstraction (circle drawing) from its implementation (color), enabling both to vary independently.

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.