Arquitectura de 3 capas

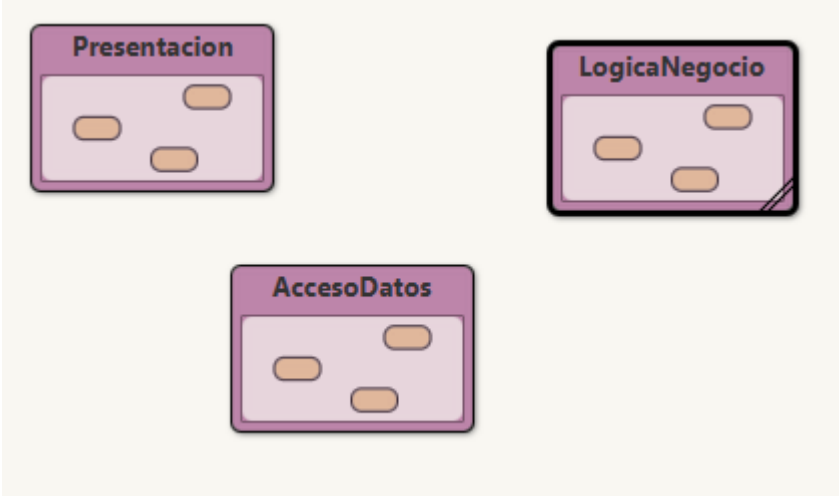
1. **Resumen de caso de uso y funcionalidades.**
   Usaré este caso de uso "**Reserva de Cancha**" del gestor de club de Tenis.

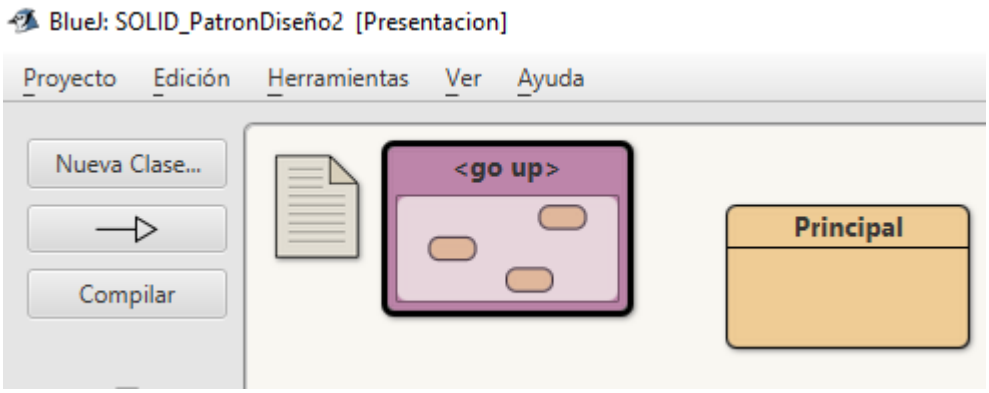
   Funcionalidades del caso de uso:
   - **Mostrar Canchas Disponibles:** Permite al usuario ver las canchas de tenis que están actualmente disponibles para su reserva.
   - **Reservar Cancha:** Permite al usuario reservar una cancha de tenis específica si está disponible.
   - **Liberar Cancha:** Permite al usuario reservar una cancha de tenis previamente reservada, cancelando la reserva y haciéndola disponible para otros miembros del club.
   - **Verificar Disponibilidad:** verifica si una cancha de tenis seleccionada está actualmente disponible para su reserva.
   - Agregar servicios adicionales a la reserva, como catering o alquiler de equipo especial.

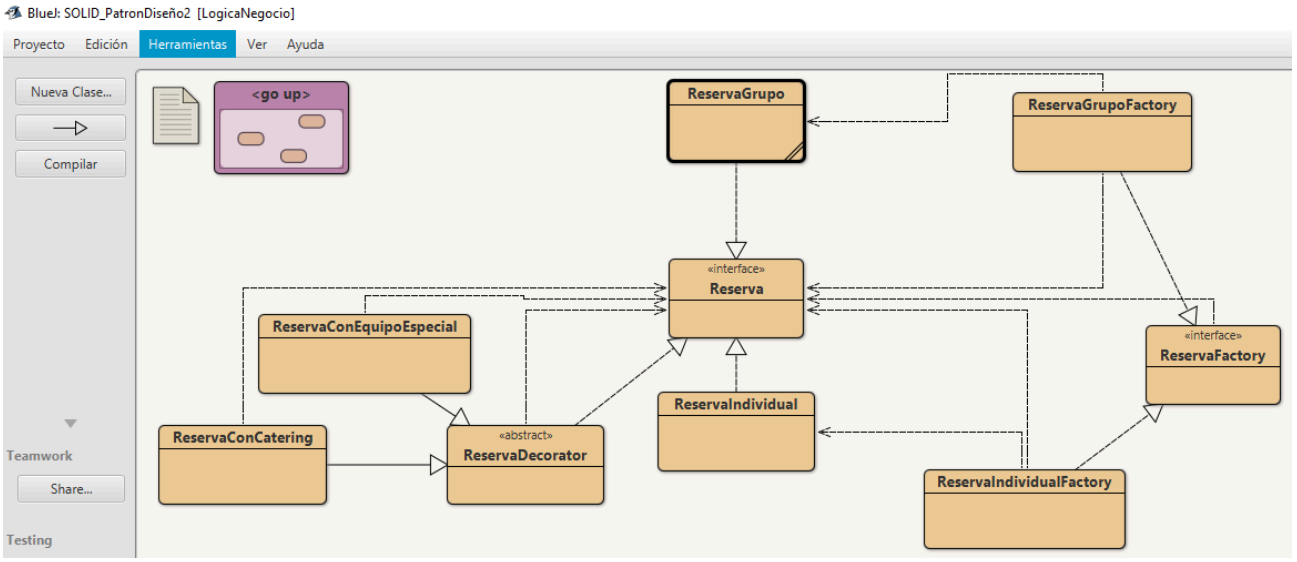
   Funcionalidades de la Arquitectura de 3 capas:
   - **Capa de Presentación:** Se comunica con el usuario, mostrando la interfaz y recogiendo sus entradas. Luego envía estas entradas a la capa de lógica de Negocio para procesarlas y recibe los resultados para mostrarlos al usuario.
   - **Capa Lógica de Negocio:** contiene la esencia de la aplicación, gestionando las operaciones y la lógica relacionada con las canchas y las reservas. Se comunica con la capa de presentación para recibir comandos y con la capa de acceso a datos para interactuar con información subyacente.
   - **Capa de Acceso a Datos:** Se encarga de interactuar con la información almacenada, como las canchas y su disponibilidad. Define cómo se estructuran y manipulan los datos, ocultando los detalles de implementación de la base de datos y permitiendo una fácil escalabilidad y mantenimiento del sistema.

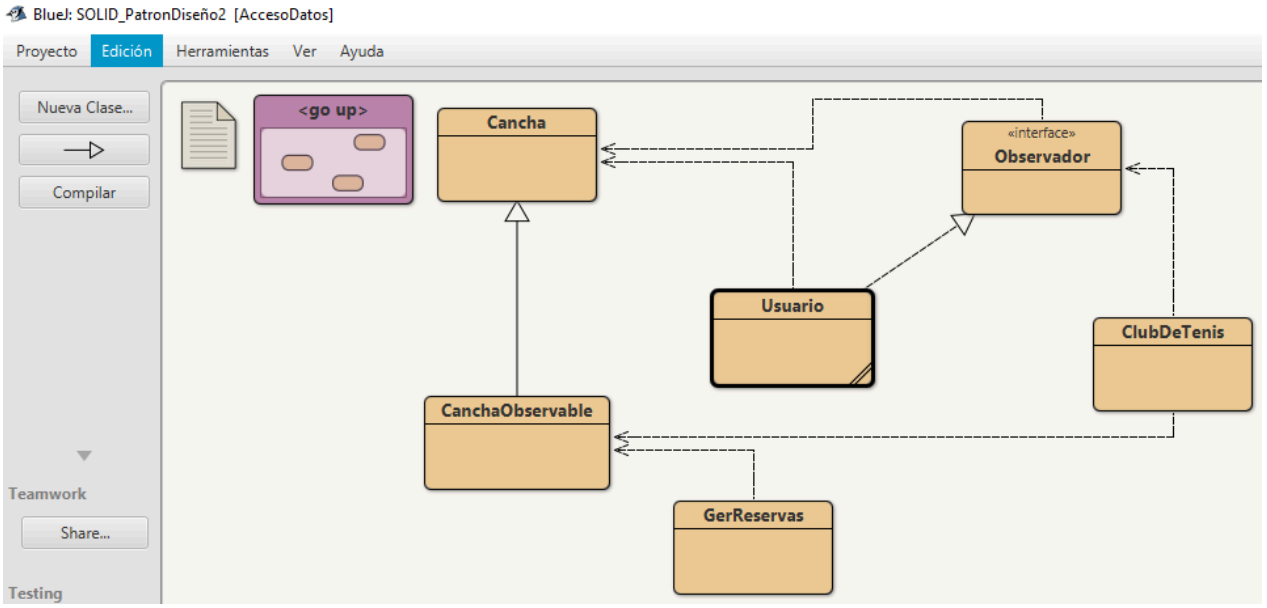
2. **Tecnologías usadas.**
   Lenguaje Java, bluej

3. **Captura de pantalla de la Arquitectura de 3 capas.**









4. Código fuente documentado por clases y rutas (path)

Principal.java
Ubicación: PatronDiseño2/Presentacion/Principal.java

```
package Presentacion;
import AccesoDatos.ClubDeTenis;
import AccesoDatos.Usuario;
import LogicaNegocio.Reserva;
import LogicaNegocio.ReservaFactory;
import LogicaNegocio.ReservaIndividualFactory;
import LogicaNegocio.ReservaGrupoFactory;
import LogicaNegocio.ReservaConCatering;
import LogicaNegocio.ReservaConEquipoEspecial;


import java.util.Scanner;

public class Principal {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ClubDeTenis club = new ClubDeTenis(5);

        Usuario user1 = new Usuario("Juan");
        Usuario user2 = new Usuario("Maria");
        Usuario user3 = new Usuario("Ana");
        Usuario user4 = new Usuario("Miguel");
        Usuario user5 = new Usuario("Pedro");
        Usuario user6 = new Usuario("Miriam");
```

```
        club.agregarCancha(user1, 1);
        club.agregarCancha(user2, 2);
        club.agregarCancha(user3, 3);
        club.agregarCancha(user4, 4);
        club.agregarCancha(user5, 5);

        ReservaFactory reservaIndividualFactory = new
ReservaIndividualFactory();
        ReservaFactory reservaGrupoFactory = new ReservaGrupoFactory();

        while (true) {
            System.out.println("        Club de Tenis   ");
            System.out.println("1. Mostrar canchas disponibles");
            System.out.println("2. Reservar cancha");
            System.out.println("3. Liberar cancha");
           System.out.println("4. Salir");
            int opcion = scanner.nextInt();

            switch (opcion) {
                case 1:
                    club.mostrarCanchasDisponibles();
                    break;
                case 2:
                    System.out.println("Ingrese el ID de la cancha a reservar:");
                    int idReservar = scanner.nextInt();
                    System.out.println("Seleccione tipo de reserva: 1. Individual 2.
Grupal");
                    System.out.println(" 1. Individual ");
                    System.out.println(" 2. Grupal");
                    int tipoReserva = scanner.nextInt();
                    Reserva reserva;
                    if (tipoReserva == 1) {
                        reserva = reservaIndividualFactory.crearReserva(idReservar);
                    } else {
                        reserva = reservaGrupoFactory.crearReserva(idReservar);
                    }

                    System.out.println("Desea agregar servicios adicionales? ");
                    System.out.println("1. Ninguno ");
                    System.out.println("2. Catering ");
                    System.out.println("3. Equipo Especial ");
                    System.out.println("4. Ambos");
```

```
                int tipoServicio = scanner.nextInt();
                if (tipoServicio == 2) {
                    reserva = new ReservaConCatering(reserva);
                } else if (tipoServicio == 3) {
                    reserva = new ReservaConEquipoEspecial(reserva);
                } else if (tipoServicio == 4) {
                    reserva = new ReservaConCatering(new
ReservaConEquipoEspecial(reserva));
                }

                reserva.mostrarDetalles();
                club.reservarCancha(idReservar);
                break;
            case 3:
                System.out.println("Ingrese el ID de la cancha a liberar:");
                int idLiberar = scanner.nextInt();
                club.liberarCancha(idLiberar);
                break;
            case 4:
                return;
            default:
                System.out.println("Opción no válida.");
        }
    }
  }
}
```

Reserva.java (Service Interface)
Ubicación: PatronDiseño2/LogicaNegocio/Reserva.java

```
package LogicaNegocio;
public interface Reserva {
   void mostrarDetalles();
}
```

ReservaFactory.java (Service Interface)
Ubicación: PatronDiseño2/LogicaNegocio/ReservaFactory.java

```
package LogicaNegocio;
public interface ReservaFactory {
   Reserva crearReserva(int idCancha);
}
```

ReservaIndividualFactory.java
Ubicación: PatronDiseño2/LogicaNegocio/ReservaIndividualFactory.java

```
package LogicaNegocio;
public class ReservaIndividualFactory implements ReservaFactory {
    @Override
    public Reserva crearReserva(int idCancha) {
        return new ReservaIndividual(idCancha);
    }
}
```

ReservaIndividual.java
Ubicación: PatronDiseño2/LogicaNegocio/ReservaIndividual.java

```
package LogicaNegocio;
public class ReservaIndividual implements Reserva {
    private int idCancha;

    public ReservaIndividual(int idCancha) {
        this.idCancha = idCancha;
    }

    @Override
    public void mostrarDetalles() {
        System.out.println("Reserva individual para la Cancha " + idCancha);
    }
}
```

ReservaGrupoFactory.java
Ubicación: PatronDiseño2/LogicaNegocio/ReservaGrupoFactory.java

```
package LogicaNegocio;
public class ReservaGrupoFactory implements ReservaFactory {
    @Override
    public Reserva crearReserva(int idCancha) {
        return new ReservaGrupo(idCancha);
    }
}
```

ReservaGrupo.java
Ubicación: PatronDiseño2/LogicaNegocio/ReservaGrupo.java

```
package LogicaNegocio;
public class ReservaGrupo implements Reserva {
    private int idCancha;
    public ReservaGrupo(int idCancha) {
        this.idCancha = idCancha;
    }

    @Override
    public void mostrarDetalles() {
        System.out.println("Reserva grupal para la Cancha " + idCancha);
    }
}
```

ReservaDecorator.java (Service Abstract)
Ubicación: PatronDiseño2/LogicaNegocio/ReservaDecorator.java

```
package LogicaNegocio;
public abstract class ReservaDecorator implements Reserva {
    protected Reserva reservaDecorada;

    public ReservaDecorator(Reserva reserva) {
        this.reservaDecorada = reserva;
    }

    public void mostrarDetalles() {
        reservaDecorada.mostrarDetalles();
    }
}
```

ReservaConCatering.java
Ubicación: PatronDiseño2/LogicaNegocio/ReservaConCatering.java

```
package LogicaNegocio;
public class ReservaConCatering extends ReservaDecorator {
    public ReservaConCatering(Reserva reserva) {
        super(reserva);
    }

    @Override
    public void mostrarDetalles() {
        super.mostrarDetalles();
        System.out.println("Incluye servicio de catering.");
    }
}
```

ReservaConEquipoEspecial.java
Ubicación: PatronDiseño2/LogicaNegocio/ReservaConEquipoEspecial.java

```
package LogicaNegocio;

public class ReservaConEquipoEspecial extends ReservaDecorator {
    public ReservaConEquipoEspecial(Reserva reserva) {
        super(reserva);
    }

    @Override
    public void mostrarDetalles() {
        super.mostrarDetalles();
        System.out.println("Incluye alquiler de equipo especial.");
    }
}
```

ClubDeTenis.java
Ubicación: PatronDiseño2/AccesoDatos/ClubDeTenis.java

```
package AccesoDatos;
import java.util.ArrayList;
import java.util.List;

public class ClubDeTenis {
    private List<CanchaObservable> canchas;
    private GerReservas gerReservas;

    public ClubDeTenis(int numCanchas) {
        canchas = new ArrayList<>();
        for (int i = 1; i <= numCanchas; i++) {
            CanchaObservable cancha = new CanchaObservable(i);
            canchas.add(cancha);
        }
        gerReservas = new GerReservas(canchas);
    }

    public void agregarCancha(Observador observador, int idCancha) {
        for (CanchaObservable cancha : canchas) {
            if (cancha.getId() == idCancha) {
                cancha.addObserver(observador);
                return;
            }
        }
        System.out.println("La cancha " + idCancha + " no existe.");
    }

    public void mostrarCanchasDisponibles() {
        System.out.println("Canchas disponibles:");
        for (CanchaObservable cancha : canchas) {
            if (cancha.isDisponible()) {
                System.out.println("Cancha " + cancha.getId());
            }
        }
    }

    public void reservarCancha(int idCancha) {
        gerReservas.reservarCancha(idCancha);
    }

    public void liberarCancha(int idCancha) {
```

```
        gerReservas.liberarCancha(idCancha);
    }
}
```

GerReservas.java
Ubicación: PatronDiseño2/AccesoDatos/GerReservas.java

```
package AccesoDatos;
import java.util.List;

public class GerReservas {
    private List<CanchaObservable> canchas;

    public GerReservas(List<CanchaObservable> canchas) {
        this.canchas = canchas;
    }

    public void reservarCancha(int idCancha) {
        for (CanchaObservable cancha : canchas) {
            if (cancha.getId() == idCancha && cancha.isDisponible()) {
                cancha.reservar();
                System.out.println("La cancha " + idCancha + " ha sido
reservada.");
                return;
            }
        }
        System.out.println("La cancha " + idCancha + " no está disponible.");
    }
    public void liberarCancha(int idCancha) {
        for (CanchaObservable cancha : canchas) {
            if (cancha.getId() == idCancha && !cancha.isDisponible()) {
                cancha.liberar();
                System.out.println("La cancha " + idCancha + " ha sido liberada.");
                return;
            }
        }
        System.out.println("La cancha " + idCancha + " no ha sido reservada.");
    }
}
```

Cancha.java
Ubicación: PatronDiseño2/AccesoDatos/Cancha.java

```
package AccesoDatos;
import LogicaNegocio.Reserva;
public class Cancha implements Reserva {
    private int id;
    private boolean disponible;

    public Cancha(int id) {
        this.id = id;
        this.disponible = true;
    }

    public int getId() {
        return id;
    }

    public boolean isDisponible() {
        return disponible;
    }

    public void reservar() {
        disponible = false;
    }

    public void liberar() {
        disponible = true;
    }

    @Override
    public void mostrarDetalles() {
        System.out.println("Cancha #" + id + " - Disponible: " + disponible);
    }
}
```

CanchaObservable.java
Ubicación: PatronDiseño2/AccesoDatos/CanchaObservable.java

```
package AccesoDatos;

import java.util.ArrayList;
import java.util.List;
public class CanchaObservable extends Cancha {
    private List<Observador> observadores = new ArrayList<>();

    public CanchaObservable(int id) {
        super(id);
    }

    public void addObserver(Observador observador) {
        observadores.add(observador);
    }

    @Override
    public void reservar() {
        super.reservar();
        notificarObservadores();
    }

    @Override
    public void liberar() {
        super.liberar();
        notificarObservadores();
    }

    private void notificarObservadores() {
        for (Observador observador : observadores) {
            observador.actualizar(this);
        }
    }
}
```

Observador.java (Service Interface)
Ubicación: PatronDiseño2/AccesoDatos/Observador.java

```
package AccesoDatos;
public interface Observador {
    void actualizar(Cancha cancha);
}
```

Usuario.java
Ubicación: PatronDiseño2/AccesoDatos/Usuario.java

```
package AccesoDatos;

public class Usuario implements Observador {
    private String nombre;

    public Usuario(String nombre) {
        this.nombre = nombre;
    }

    @Override
    public void actualizar(Cancha cancha) {
        System.out.println("Hola " + nombre + ", la Cancha " + cancha.getId() +
                           " está " + (cancha.isDisponible() ? "disponible." :
"reservada."));
    }
}
```

5. Resumen de cumplimiento de SOLID, Patrones y Arquitectura.

1) Principio SOLID:

- **Single Responsibility Principle(SRP):** En el código implementado, cada clase tiene una responsabilidad clara y única.
- **Open/Closed Principle(OCP):** Las entidades deben estar abiertas para exención pero cerradas para modificación. En el código implementado, este principio se sigue al utilizar patrones como el **'Factory Method'** y **'el Decorator'.**
- **Liskov Substitution Principle(LSP):** Las subclases deben poder ser sustituidas por sus clases base sin alterar el comportamiento del programa. En el código proporcionado, **'CanchaObservable'** es una subclase de **'Cancha',** pero agregar comportamiento adicional para notificar a los observadores cuando el estado de la cancha cambia.
- **Interface Segregation Principle(ISP):** Los clientes no deben verse obligados a depender de interfaces que no utilizan. En el código implementado, las interfaces **'Reserva'** y **'ReservaFactory'** son interfaces pequeñas y cohesivas que definen contratos específicos para clases relacionadas.
- **Dependency Inversion Principle(DIP):** Los módulos de alto nivel no deben depender de módulos de bajo nivel, ambos deben depender de abstracciones**.** El código implementado, **'ClubTenis'** y **'GerReservas'** dependen de abstracciones como **'Cancha'** y **'Reserva',** en lugar de depender de implementaciones concretas.

2) Patrones de Diseño:

- **Creacionales(Factory Method):** Este patrón se utilizará para crear diferentes tipos de reservas**('ReservaIndividual' y 'ReservaGrupo')** de manera flexible, si acoplar la lógica de creación a las clases que las utilizan.
- **Estructurales(Decorator):** Este patro se utiliza para agregar funcionalidades adicionales a las reservas, como servicios de **catering** o **alquiler de equipo especial,** de manera dinamica y transparente para el cliente.
- **Comportamiento(Observer):** Este patrón se implementa para permitir que los observadores **(en este caso, los usuarios)** reciban notificaciones cuando cambia el estado de una cancha. La clase **'CanchaObservable',** que notifica a sus observadores cuando se reserva o se libera una cancha.

3) **Arquitectura de 3 capas:**
    - **Capa de Presentación:** Esta capa está siendo representada por la clase **‘Principal’,** que se encarga de interactuar con el usuario y presentar las opciones disponibles.
    - **Capa Lógica de Negocio:** Esta capa esta siendo representada por las clases **‘ClubTenis’** y **‘GerReservas’,** que contienen la lógica para gestionar las operaciones relacionadas con las canchas y las reservas.
    - **Capa Acceso a Datos:** Esta capa esta representada por las clases **‘Cancha’** y **‘CanchaObservable’,** que define los modelos de datos y la lógica para interactuar con ellos.