EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Tijuana

Departamento de Sistemas y Computación

Nombre de la Materia:

Patrones de Diseño

Actividad:

Proyecto Final

Profesor:

Maribel Guerrero Luis

Alumno(s):

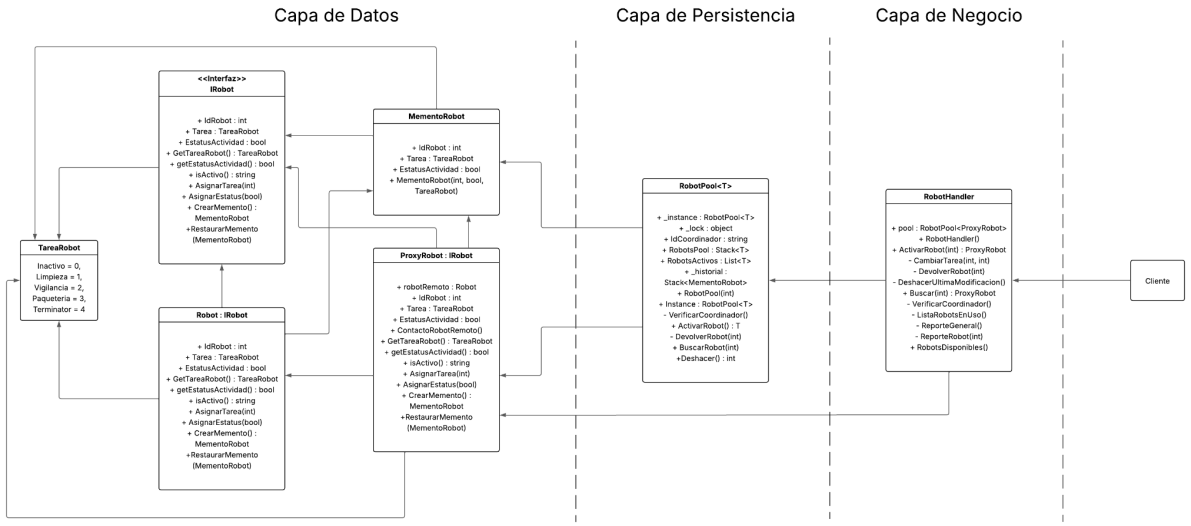
Jonathan Rafael Tamayo Ortiz

Fecha de entrega:

10/12/2025

Proyecto Final

Diagrama



Código

```
namespace ProyectoV7.Capa_Datos
{
    public enum TareaRobot
    {
        Inactivo = 0,
        Limpieza = 1,
        Vigilancia = 2,
        Paqueteria = 3,
        Terminator = 4
    }
}
—
namespace ProyectoV7.Capa_Datos
{
    public interface IRobot
    {
        int IdRobot { get; set; }
        TareaRobot Tarea { get; set; }
        bool EstatusActividad { get; set; }
        TareaRobot GetTareaRobot();
        bool getEstatusActividad();
        string isActivo();
        void AsignarTarea(int tipo);
        void AsignarEstatus(bool estatus);
        MementoRobot CrearMemento();
        void RestaurarMemento(MementoRobot memento);
    }
}
```

```
—
namespace ProyectoV7.Capa_Datos
{
   public class Robot : IRobot
   {
      public int IdRobot { get; set; }
      public TareaRobot Tarea { get; set; }
       public bool EstatusActividad { get; set; } //El robot esta activo (true) o inactivo
(false)

       public TareaRobot GetTareaRobot() //Devuelve la tarea que esta realizando un
robot
      {
         return Tarea;
      }
      public bool getEstatusActividad() //Devuelve un valor booleano
      {
         return EstatusActividad;
      }
      public string isActivo() //Devuelve un valor string correspondiente a su estatus
      {
         if (EstatusActividad) //activo = (true) e inactivo = (false)
         {
            return "Activo";
         }
         return "Inactivo";
      }
      public void AsignarTarea(int tipo)
      {
         if (EstatusActividad) // Si el robot esta acivo, asigna la tarea que corresponde
         {
            Tarea = (TareaRobot)tipo;
         }
         else // Si el robot esta inactivo, lo marca como 0
         {
            Tarea = TareaRobot.Inactivo;
         }
      }
      public void AsignarEstatus(bool estatus)
      {
         EstatusActividad = estatus;
         if (!estatus)
         {
            Tarea = TareaRobot.Inactivo;
         }
      }
      //Metodos memento
      public MementoRobot CrearMemento()
      {
         return new MementoRobot(IdRobot, EstatusActividad, Tarea);
```

```
        }

        public void RestaurarMemento(MementoRobot memento)
        {
                if (memento.IdRobot != this.IdRobot) return; //Verificar que el memento
corresponde a este robot

            this.EstatusActividad = memento.EstatusActividad;
            this.Tarea = memento.Tarea;
        }
    }
}
—
namespace ProyectoV7.Capa_Datos
{
    public class ProxyRobot : IRobot
    {
        private Robot robotRemoto;

        public int IdRobot { get; set; }
        public TareaRobot Tarea { get; set; }
          public bool EstatusActividad { get; set; } //El robot esta activo (true) o inactivo
(false)

        private void ContactoRobotRemoto() //Conexion con el robot remoto
        {
            if (robotRemoto == null)
            {
                robotRemoto = new Robot(); //Nueva conexion
                robotRemoto.IdRobot = this.IdRobot;
            }
        }

          public TareaRobot GetTareaRobot() //Devuelve la tarea que esta realizando un
robot
        {
            ContactoRobotRemoto();
            Tarea = robotRemoto.GetTareaRobot();
            return Tarea;
        }
        public bool getEstatusActividad() //Devuelve un valor booleano
        {
            ContactoRobotRemoto();
            EstatusActividad = robotRemoto.getEstatusActividad();
            return EstatusActividad;
        }
        public string isActivo() //Devuelve un valor string correspondiente a su estatus
        {
            ContactoRobotRemoto();
            return robotRemoto.isActivo();
```

```
        }
        public void AsignarTarea(int tipo)
        {
            ContactoRobotRemoto();
            robotRemoto.AsignarTarea(tipo);
            Tarea = robotRemoto.Tarea;
        }
        public void AsignarEstatus(bool estatus)
        {
            ContactoRobotRemoto();
            robotRemoto.AsignarEstatus(estatus);
            EstatusActividad = estatus;
        }
        //Metodos memento
        public MementoRobot CrearMemento()
        {
            ContactoRobotRemoto();
            return robotRemoto.CrearMemento();
        }
        public void RestaurarMemento(MementoRobot memento)
        {
            ContactoRobotRemoto();
            robotRemoto.RestaurarMemento(memento);
            Tarea = robotRemoto.Tarea;
            EstatusActividad = robotRemoto.EstatusActividad;
        }
    }
}
—
namespace ProyectoV7.Capa_Datos
{
    public class MementoRobot
    {
        public int IdRobot { get; }
        public TareaRobot Tarea { get; }
        public bool EstatusActividad { get; }
        public MementoRobot(int idRobot, bool estatus, TareaRobot tarea)
        {
            IdRobot = idRobot;
            EstatusActividad = estatus;
            Tarea = tarea;
        }
    }
}
—
namespace ProyectoV7.Capa_Persistencia
{
    public class RobotPool<T> where T : class, IRobot, new()
    {
        //Singleton
```

```
    private static RobotPool<T> _instance; //Una sola instancia de la piscina
    private static readonly object _lock = new object();
    public string IdCoordinador { get; set; }

    //Object pool
      public readonly Stack<T> RobotsPool = new Stack<T>(); //Stack de todos los
robots a la espera de instrucciones
    public readonly List<T> RobotsActivos = new List<T>(); //Lista de robots en uso

    //Memento
                    public readonly Stack<MementoRobot> _historial = new
Stack<MementoRobot>(); // Historial

    private RobotPool(int maxPool)
    {
       IdCoordinador = "12345";
       //Instanciar los robots pertenecientes al pool
       for (int i = 1; i <= maxPool; i++)
       {
          var robot = new T();
          robot.IdRobot = i;
          robot.AsignarEstatus(false);
          robot.AsignarTarea(0);

          RobotsPool.Push(robot);
       }
    }
    public static RobotPool<T> Instance // Inicializa la instancia del coordinador
    {                                   // Lo cual inicializa la piscina de objetos
       get
       {
          if (_instance == null)
          {
             lock (_lock)
             {
                if (_instance == null)
                {
                   _instance = new RobotPool<T>(10);
                   Console.WriteLine("Nueva instancia creada");
                }
             }
          }
          else
          {
             Console.WriteLine("Instancia existente devuelta");
          }
          return _instance;
       }
    }
    public void VerificarCoordinador()
```

```
        {
            Console.WriteLine($"Coordinador ID: {IdCoordinador}");
        }
        public T ActivarRobot() //Activa un robot
        {
            if (RobotsPool.Count > 0) //Verifica que hay robots disponibles
            {
                var robot = RobotsPool.Pop();
                _historial.Push(robot.CrearMemento());

                robot.AsignarEstatus(true);
                RobotsActivos.Add(robot);
                return robot;
            }
            return null;
        }
        public void DevolverRobot(int id) //Devuelve el robot al pool
        {
            var robot = RobotsActivos.FirstOrDefault(r => r.IdRobot == id);
            if (robot == null) return;
            _historial.Push(robot.CrearMemento());
            robot.AsignarEstatus(false);
            robot.AsignarTarea(0);
            //Modifica las listas de robots
            RobotsActivos.Remove(robot);
            RobotsPool.Push(robot);
        }
        public T BuscarRobot(int id) //Busca un robot especifico
        {
            return RobotsPool.Concat(RobotsActivos).FirstOrDefault(r => r.IdRobot ==
id);
        }
        public int Deshacer() //Deshace la ultima accion
        {   //Devuelve el ID del robot afectado
            if (_historial.Count == 0)
            {
                Console.WriteLine("No hay historial para deshacer");
                return 0;
            }
            var memento = _historial.Pop();
            var robot = BuscarRobot(memento.IdRobot);
            if (robot == null)
            {
                Console.WriteLine($"Robot ID: {memento.IdRobot} no encontrado");
                return 0;
            }

            bool estadoActual = robot.getEstatusActividad();
            robot.RestaurarMemento(memento);
            bool estadoNuevo = robot.getEstatusActividad();
```

```
            //Modificar listas
            if (estadoActual && !estadoNuevo)
            {
                //De Activo a Inactivo
                var r = RobotsActivos.FirstOrDefault(x => x.IdRobot == robot.IdRobot);
                if (r != null)
                {
                    RobotsActivos.Remove(r);
                    RobotsPool.Push(r);
                    Console.WriteLine($"Robot ID {robot.IdRobot} desactivado");
                }
            }
            else if (!estadoActual && estadoNuevo)
            {
                //De Inactivo a Activo
                var temp = new Stack<T>();
                T encontrado = null;
                while (RobotsPool.Count > 0) //Almacena robots disponibles
                {
                    var r = RobotsPool.Pop();
                    if (r.IdRobot == robot.IdRobot && encontrado == null)
                        encontrado = r;
                    else
                        temp.Push(r);
                }
                while (temp.Count > 0) //Los regresa una vez encuentra el que busca
                    RobotsPool.Push(temp.Pop());

                if (encontrado != null)
                {
                    RobotsActivos.Add(encontrado);
                    Console.WriteLine($"Robot ID {robot.IdRobot} restaurado");
                }
            }
            return robot.IdRobot;
        }
    }
}
—
namespace ProyectoV7.Capa_Negocio
{
    public class RobotHandler
    {
        private readonly RobotPool<ProxyRobot> pool;
        public RobotHandler()
        {
            pool = RobotPool<ProxyRobot>.Instance;
        }
```

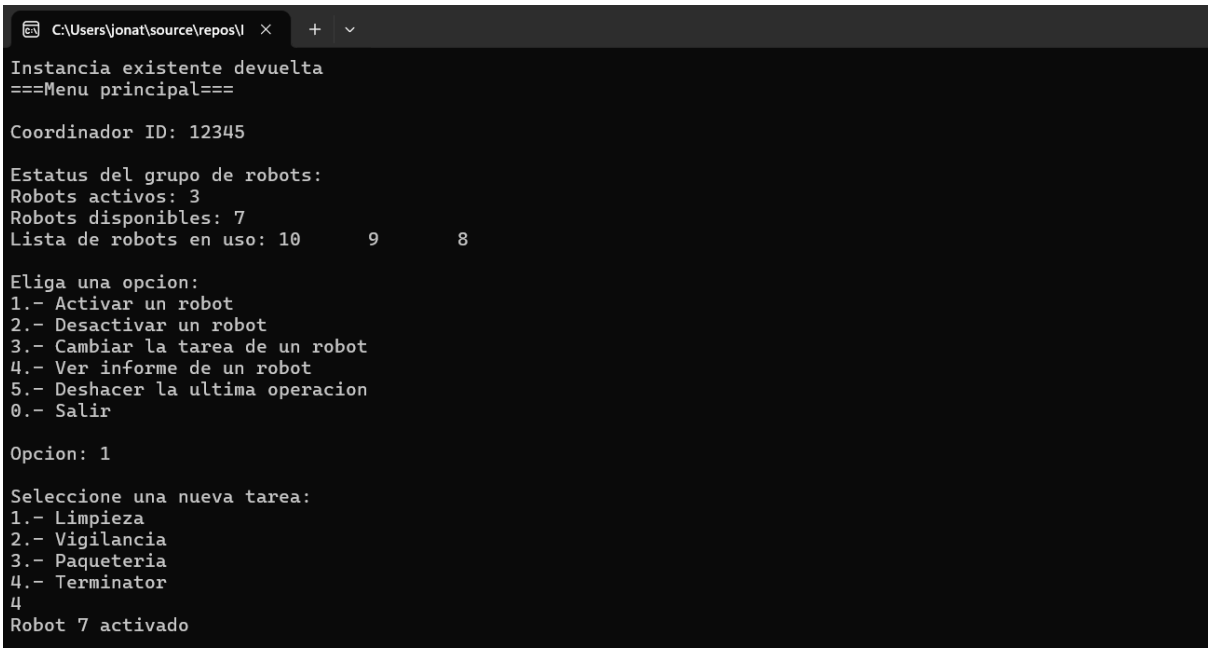
```
        public ProxyRobot ActivarRobot(int tarea)
        {
            var robot = pool.ActivarRobot(); //Crea memento
            if (robot == null) return null;

            robot.AsignarTarea(tarea); //No crea memento
            return robot;
        }
        public void CambiarTarea(int id, int tarea)
        {
            var robot = pool.BuscarRobot(id);
            if (robot == null) return;

            pool._historial.Push(robot.CrearMemento()); //Crea memento
            robot.AsignarTarea(tarea); //No crea memento
                                Console.WriteLine($"Tarea de la unidad {robot.IdRobot}:
{robot.GetTareaRobot()}");
        }
        public void DevolverRobot(int id)
        {
            pool.DevolverRobot(id); //Crea memento
        }
        public void DeshacerUltimaModificacion()
        {
            var id = pool.Deshacer();
            if (id == 0) //Error al deshacer
            {
                Console.WriteLine("La operacion no de pudo deshacer");
                return;
            }
            var robot = pool.BuscarRobot(id);
            Console.WriteLine($"Operación deshecha para robot ID {robot.IdRobot}");
            ReporteRobot(id);
        }
        public ProxyRobot Buscar(int id)
        {
            return pool.BuscarRobot(id);
        }
        public void VerificarCoordinador()
        {
            pool.VerificarCoordinador();
        }
        public void ListaRobotsEnUso()
        {
            Console.Write("Lista de robots en uso: ");
            foreach (var robot in pool.RobotsActivos)
            {
                Console.Write(robot.IdRobot + "\t");
            }
        }
```

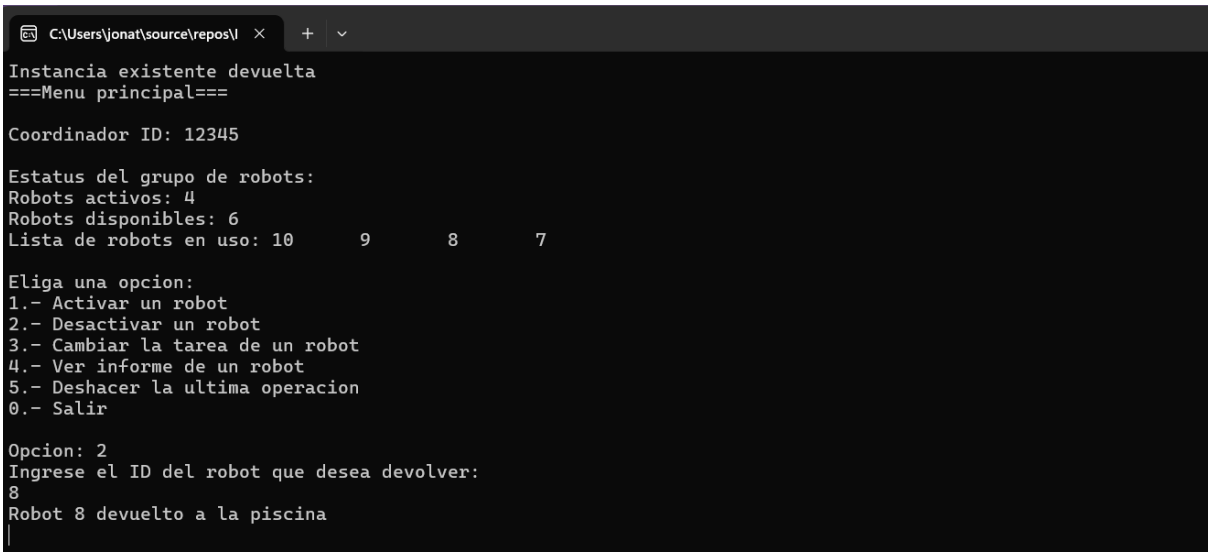
```
        public void ReporteGeneral()
        {
            Console.WriteLine("\nEstatus del grupo de robots:\n" +
                $"Robots activos: {pool.RobotsActivos.Count}\n" +
                $"Robots disponibles: {pool.RobotsPool.Count}");

            if (pool.RobotsActivos.Count > 0)
                ListaRobotsEnUso();
        }
        public void ReporteRobot(int id)
        {
            var robot = Buscar(id);
                                Console.WriteLine($"\nRobot   ID:   {robot.IdRobot}\nTarea:
{robot.GetTareaRobot()}\nEstatus: {robot.isActivo()}");
        }
        public int RobotsDisponibles()
        {
            return pool.RobotsPool.Count;
        }
    }
}
—
namespace ProyectoV7
{
    internal class Program
    {
        static void Main(string[] args)
        {
            string opc = "";
            int id = 0;

            do
              {   //Intenta crear una nuava instancia con cada ciclo. Singleton devuelve la
ya existente
                var coordinador = new RobotHandler();

                Console.WriteLine("===Menu principal===\n");
                coordinador.VerificarCoordinador();
                coordinador.ReporteGeneral();
                Console.WriteLine("\n\nEliga una opcion:");
                Console.WriteLine("1.- Activar un robot");
                Console.WriteLine("2.- Desactivar un robot");
                Console.WriteLine("3.- Cambiar la tarea de un robot");
                Console.WriteLine("4.- Ver informe de un robot");
                Console.WriteLine("5.- Deshacer la ultima operacion");
                Console.WriteLine("0.- Salir\n");
                Console.Write("Opcion: ");
                opc = Console.ReadLine();

                switch (opc)
```

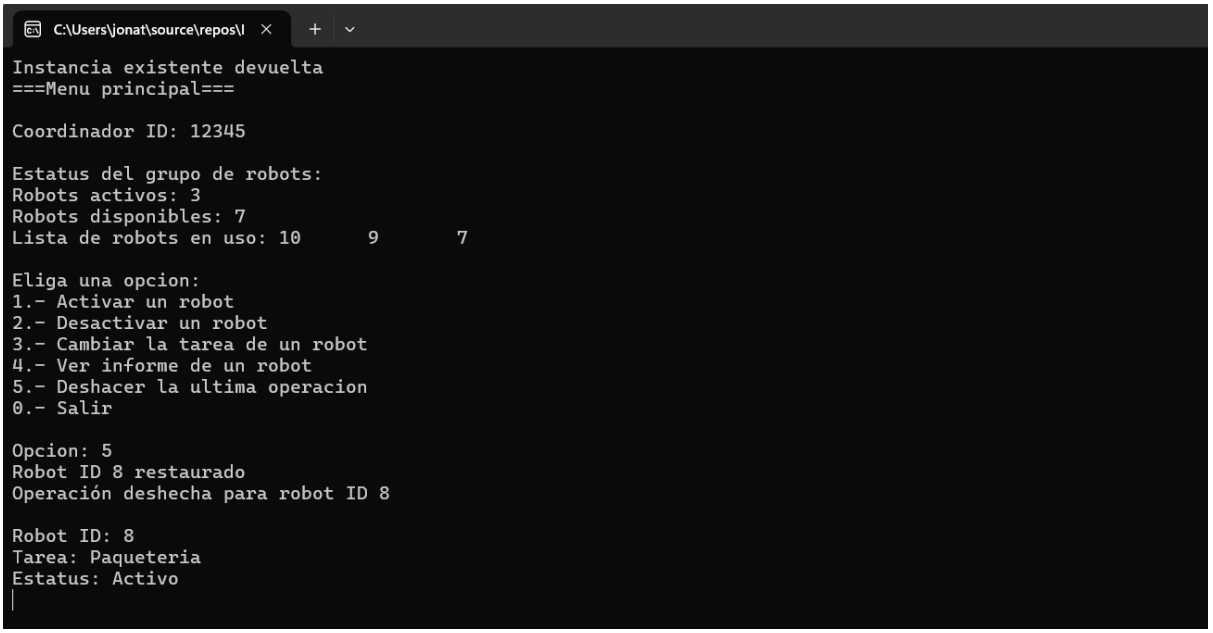
```
            {
                case "1": //Activar robot
                    if (coordinador.RobotsDisponibles() == 0)
                    {
                                Console.WriteLine("No es posible activar un robot en este
momento");
                        break;
                    }
                    Console.WriteLine("\nSeleccione una nueva tarea:\n" +
                          $"1.- Limpieza\n2.- Vigilancia\n3.- Paqueteria\n4.- Terminator");
                      if (int.TryParse(Console.ReadLine(), out int tarea) && (tarea >= 0 &&
tarea <= 4))
                    {
                        var r1 = coordinador.ActivarRobot(tarea);
                        if (r1 != null)
                            Console.WriteLine($"Robot {r1.IdRobot} activado");
                    }
                    else
                        Console.WriteLine("Esa opcion no es valida");
                    break;

                case "2": //Devolver robot
                    Console.WriteLine("Ingrese el ID del robot que desea devolver:");
                    int.TryParse(Console.ReadLine(), out id);
                    var r2 = coordinador.Buscar(id);
                    if (r2 == null)
                    {
                        Console.WriteLine("No se pudo encontrar al robot");
                        break;
                    }
                    coordinador.DevolverRobot(id);
                    Console.WriteLine($"Robot {r2.IdRobot} devuelto a la piscina");
                    break;

                case "3": //Cambiar la tarea de un robot
                      Console.WriteLine("Ingrese el ID del robot al que desea asignar una
nueva tarea:");
                    int.TryParse(Console.ReadLine(), out id);
                    Console.WriteLine("\nSeleccione una nueva tarea:\n" +
                          $"1.- Limpieza\n2.- Vigilancia\n3.- Paqueteria\n4.- Terminator");
                    var r3 = coordinador.Buscar(id);
                    if (r3 == null)
                    {
                        Console.WriteLine("No se pudo encontrar al robot");
                        break;
                    }
                            if (int.TryParse(Console.ReadLine(), out int nuevaTarea) &&
(nuevaTarea >= 0 && nuevaTarea <= 4))
                    {
                        coordinador.CambiarTarea(id, nuevaTarea);
```

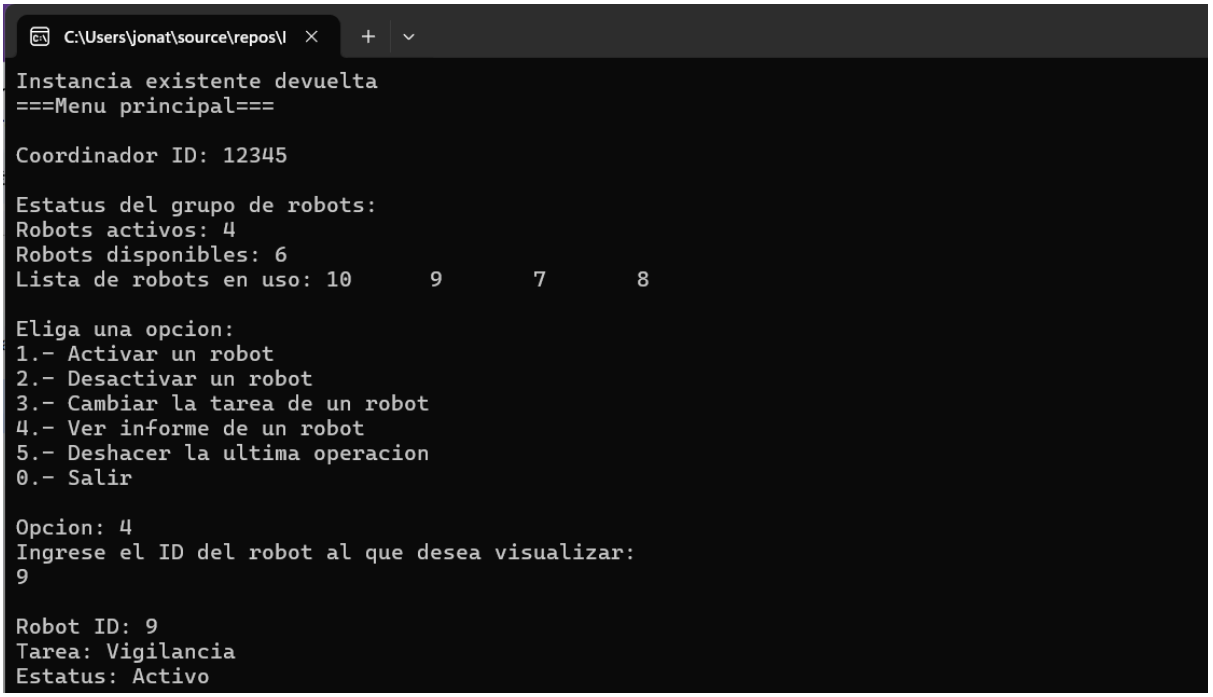
```
                    }
                    else
                        Console.WriteLine("Esa opcion no es valida");
                    break;

                case "4": //Muestra el estado de un solo robot
                    Console.WriteLine("Ingrese el ID del robot al que desea visualizar:");
                    int.TryParse(Console.ReadLine(), out id);
                    var r4 = coordinador.Buscar(id);
                    if (r4 == null)
                    {
                        Console.WriteLine("No se pudo encontrar al robot");
                        break;
                    }
                    coordinador.ReporteRobot(r4.IdRobot);
                    break;

                case "5": //Deshacer la ultima operacion
                    coordinador.DeshacerUltimaModificacion();
                    break;

                case "0": //Salir
                            Console.WriteLine("Saliendo. Presione cualquier tecla para
continuar...");
                    break;

                default:
                    Console.WriteLine("Esa opcion no es valida.");
                    break;
            }
            Console.ReadKey();
            Console.Clear();
        } while (opc != "0");
    }
  }
}
```

Capturas


C:\Users\jonat\source\repos\I
Instancia existente devuelta
===Menu principal===
Coordinador ID: 12345
Estatus del grupo de robots:
Robots activos: 3
Robots disponibles: 7
Lista de robots en uso: 10 9 8
Eliga una opcion:
1.- Activar un robot
2.- Desactivar un robot
3.- Cambiar la tarea de un robot
4.- Ver informe de un robot
5.- Deshacer la ultima operacion
0.- Salir
Opcion: 1
Seleccione una nueva tarea:
1.- Limpieza
2.- Vigilancia
3.- Paqueteria
4.- Terminator
4
Robot 7 activado


C:\Users\jonat\source\repos\I
Instancia existente devuelta
===Menu principal===
Coordinador ID: 12345
Estatus del grupo de robots:
Robots activos: 4
Robots disponibles: 6
Lista de robots en uso: 10 9 8 7
Eliga una opcion:
1.- Activar un robot
2.- Desactivar un robot
3.- Cambiar la tarea de un robot
4.- Ver informe de un robot
5.- Deshacer la ultima operacion
0.- Salir
Opcion: 2
Ingrese el ID del robot que desea devolver:
8
Robot 8 devuelto a la piscina





Conclusión

Este proyecto sirvió para integrar los conocimientos adquiridos a lo largo de todo el semestre. Se incluyen patrones de todo tipo: creacionales, estructurales, de comportamiento y de arquitectura. La integración de estos patrones permite la creación de código más eficiente, reutilizable y mantenible. Saber distinguir las necesidades de nuestra aplicación y juzgar cuál patrón es el más acorde a estas es de gran ayuda en nuestra carrera profesional.