Ej1:

Se viola el LSP, porque la Animal tiene un método "Volar", que la subclase "Perro", al ejecutarla, lanza un error, lo cual hace que se deba saber si el animal es un perro o no antes de ejecutar el método.

Solución:

```
public abstract class Animal {
	public abstract void Comer();
}

public abstract class AnimalVolador {
	public abstract void Volar();
}

public class Perro: Animal {
	public override void Comer() {
		// El perro come
	}
}
```

Ej2:

Se viola el SRP, porque la clase Impresora tiene dos responsabilidades: Imprimir y Escanear.

Solución:

```
public class Documento {
      public string Contenido { get; set; }
}

public class Escaner {
      public void Escanear() {...};
}

public class Impresora {
      public void Imprimir() {...};
}
```

Ej 3:

Viola el SRP, porque la clase BaseDeDatos tiene dos responsabilidades: guardar un objeto y enviar un mensaje.

Solución

```
public class BaseDeDatos {
	public void Guardar(Object objeto) {
		// Guarda el objeto en la base de datos
	}
}

public class CorreoElectronico {
	public void EnviarMensaje(string correo, string mensaje) {
		// Envía un correo electrónico
	}
}
```

Ej4:

Viola el ISP, porque no hay forma de depender de uno de los métodos de la clase Robot sin depender de todos.

Solución:

```
public interface IEntidadQueCocina {
	public void Cocinar();
}

public interface IEntidadQueLimpia {
	public void Limpiar();
}

public interface IEntidadConBateria {
	public void RecargarBateria();
}

public class Robot: IEntidadQueCocina, IEntidadQueLimpia,
IEntidadConBateria {
	public void Cocinar();
	public void Limpiar();
	public void RecargarBateria();
}
```

Ej5:

(DIP) implementar interfaz pedido para crear el pedido después. El problema es que el pedido es una clase de alto nivel y el cliente es de bajo y al poder interactuar desde el cliente creando un pedido se viola el patrón.

Ej6:

Viola el LSP, porque no se puede utilizar una instancia de la clase PatoDeGoma como si fuera una instancia de la clase Pato, por el método Volar.

Solución:

```
public abstract class Pato {

}

public class PatoDeVerdad: Pato {
      public void Nadar() {}
      public void Graznar {}
      public void Volar() {}
}

public class PatoDeGoma: Pato {}
```

Ej7:

Viola el ISP, porque la clase ReadDatabase no precisa implementar el método WriteData de IDatabase, por lo que se debería separar la interfaz entre una que tiene los metodos de conexion y desconexión, y otra que tenga los métodos de lectura de datos.

Solución:

```
public interface IReadDatabase {
	void Connect();
	void Disconnect();
}

public interface IDatabase: IReadDatabase {
	void WriteData();
}

public class Database: IDatabase {
	public void Connect() {
		// Logic for connecting
	}

	public void Disconnect() {
		// Logic for disconnecting
	}

	public void WriteData() {
		// Logic for writing
	}
}

public class ReadDatabase: IReadDatabase {
	public void Connect() {
		// Logic for connecting
	}

	public void Disconnect() {
		// Logic for disconnecting
	}
}
```

Ej8:

Por el principio de Inversión de Dependencias. La clase AutoSave, en teoría, debería poder funcionar con cualquier tipo de “Saver”, pero aquí solo funciona con FileSaver. Si se quiere admitir otros ‘Saver’s seria mejor depender de una interfaz ISaver que requiera el método Save, lo cual permite crear otros ‘Saver’s a partir de esta interfaz.

Solución:

```
public interface ISaver {
	void Save(Document doc);
}

public class FileSaver: ISaver {
	public string FileName { get; init; }

	public FileSaver(string fileName) {
		if(string.isNullOrEmpty(fileName))
			throw new ArgumentException();
		this.FileName = fileName;
	}

	public void Save(Document doc) {
		// Logic for saving the document
	}
}

public class AutoSave {
	private ISaver _saver { get; init; }

	public AutoSave(ISaver saver) {
		this._saver = saver;
	}
}
```

Ej9:

Viola el SRP, porque la clase User se encarga de almacenar información de sí mismo, y de determinar si puede editar un Post.

Solución:

```
public class User {
	public bool IsAdmin { get; set; }
}

public class Post {
	public User Author { get; set; }
}

public static class System {
	public bool UserCanEditPost(User user, Post post) {
		return user.IsAdmin || post.Author == user;
	}
}
```

Ej10:

Se viola el OCP, porque si quisiéramos añadir un nuevo formato de música, habría que modificar la clase MusicPlayer para añadir el método correspondiente al mismo.

Solución:

```
public interface IMusicPlayer {
      // Returns false if the file can’t be played by this player
      public bool Play(string fileName);
}

public class Mp3Player: IMusicPlayer {
      public bool Play(string fileName) {
            // Checks the file format
            // Lógica para reproducir archivos MP3
      }
}

public class WavPlayer: IMusicPlayer {
      public bool Play(string fileName) {
            // Checks the file format
            // Lógica para reproducir archivos WAV
      }
}

public class PlayFlac: IMusicPlayer {
      public bool Play(string fileName) {
            // Checks the file format
            // Lógica para reproducir archivos FLAC
      }
}

public class MultiOptionMusicPlayer: IMusicPlayer {
      public IMusicPlayer[] Players { get; set; }

      public bool Play(string fileName) {
            for (IMusicPlayer player : Players) {
                  if (player.Play(fileName)) return true;
            }
            return false;
      }
}
```