EJERCICIO 1

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

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

  public override void Volar() {
    throw new NotImplementedException();
  }
}
```

LSP: Si se sustituye al hijo con el padre se va a romper el código, Perro con Animal
ISP: La clase Animal implementa el método Volar, lo cual obliga a que Perro implemente el método Volar() y eso incumple el principio.

Parte 2:
Para solucionar esto, se podría separar el metodo 'Volar' de la clase abstracta 'Animal'.
Se podría crear otra clase abstracta AnimalVolador que contenga dicho método y todos los futuros que correspondan a un animal volador.

EJERCICIO 2

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

public class Impresora {
  public void Imprimir(Documento documento) {
    Console.WriteLine(documento.Contenido);
  }

  public void Escanear(Documento documento) {
    // Código complejo para escaneo...
  }
}
```

Parte 1:
SRP: Se debería tener una clase para Imprimir y otra para Escanear.

Parte 2:
Clase Escanner que tendrá el método Escanear.

EJERCICIO 3

```
1 public class BaseDeDatos {
2   public void Guardar(Object objeto) {
3     // Guarda el objeto en la base de datos
4   }
5
6   public void EnviarCorreo(string correo, string
        mensaje) {
7     // Envía un correo electrónico
8   }
9 }
```

Parte 1:
SRP: La clase BaseDeDatos contiene más de una razón para cambiar. El método EnviarCorreo no corresponde a esta clase.

Parte 2: Se podría crear otra clase 'ServicioEmail' para todos los metodos que correspondan a los métodos de Email y remover el mismo de la clase 'BaseDeDatos'.

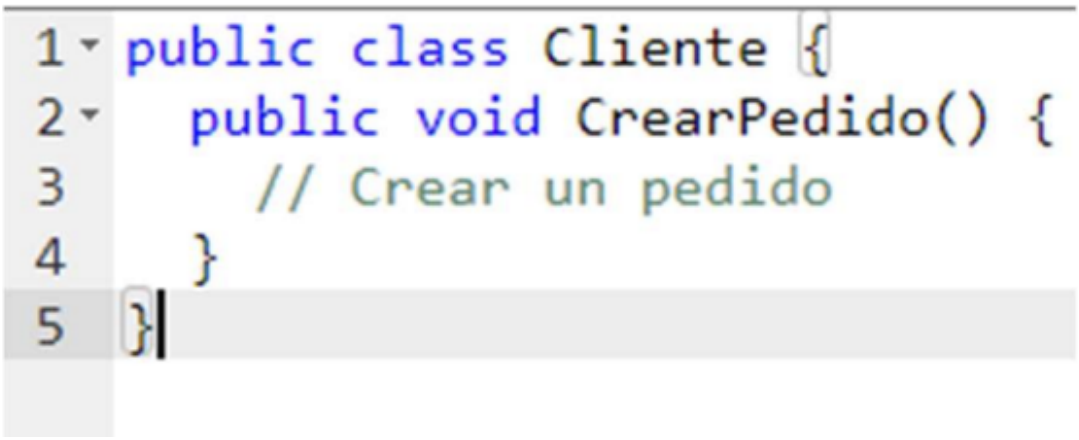
EJERCICIO 4:

```
1  public class Robot {
2    public void Cocinar() {
3      // Cocina algo
4    }
5
6    public void Limpiar() {
7      // Limpia algo
8    }
9
10   public void RecargarBateria() {
11     // Recarga la batería
12   }
13 }
```

Parte 1:
SRP: Robot tiene más de una responsabilidad que corresponda a un Robot, podría dividirse en más clases para cada funcionalidad.

Parte 2:
Se podría mejorar agregando interfaces donde cada una implementa cada funcionalidad del robot.
A la vez, en el caso de que el robot pueda tener otras funcionalidades, se podría también implementar una abstracción para la acción que realiza, y dejar en robot un método el cual se encarga de ejecutar dicha acción, con el fin de cumplir OCP.

EJERCICIO 5



```
1 public class Cliente {
2   public void CrearPedido() {
3     // Crear un pedido
4   }
5 }
```

Parte 1:
xd

Parte 2:
Xd2

EJERCICIO 6:

```
1  public class Pato {
2    public void Nadar() {
3      // Nada
4    }
5
6    public void Graznar() {
7      // Grazna
8    }
9
10   public void Volar() {
11     // Vuela
12   }
13 }
14
15 public class PatoDeGoma: Pato {
16   public override void Volar() {
17     throw new NotImplementedException();
18   }
19 }
```

Parte 1:
LSP: la clase pato de goma no define los otros métodos de la clase padre.
ISP: La clase pato de goma no define el método Volar porque no lo necesita.

Parte 2: