UNIDAD TEMÁTICA 5 – Patrones de diseño– Trabajo de Aplicación 1

Para cada uno de los siguientes ejercicios, en equipo:

1- Determine que anti-patrón se ajusta mejor al código presentado.
2- Agregue las clases, interfaces, métodos que considere necesarios para remediar la situación.

EJERCICIO 1

```
public class TODOController
{
    private List<Todo> todos;

    public TODOController()
    {
        this.todos = new List<Todo>();
    }

    public void Add(Todo todo)
    {
        todos.Add(todo);
    }

    public void Delete(int id)
    {
        Todo todo = todos.Find(t => t.Id == id);
        if (todo != null)
            todos.Remove(todo);
    }

    public void Update(Todo todo)
    {
        Todo oldTodo = todos.Find(t => t.Id == todo.Id);
        if (oldTodo != null)
        {
            oldTodo.Title = todo.Title;
            oldTodo.Description = todo.Description;
            oldTodo.Completed = todo.Completed;
        }
    }

    // Aquí hay más métodos relacionados a la gestión de tareas (TODOs)
    // ...
    // Y después, también hay métodos de gestión de usuarios.
    // ...
    // Y aún más, métodos para la gestión de permisos.
    // ...
}
```

EJERCICIO 2

```
public class UserManager
{
    // A lot of code here
    // ...
    public void CreateUser(string name, string email, string password)
    {
        // Validation code
        // ...
        // Save user to database
        // ...
    }

    public void DeleteUser(int id)
    {
        // Validation code
        // ...
        // Delete user from database
        // ...
    }

    public void UpdateUser(int id, string name, string email, string password)
    {
        // Validation code
        // ...
        // Update user in database
        // ...
    }

    public void ChangePassword(int id, string oldPassword, string newPassword)
    {
        // Validation code
        // ...
        // Update password in database
        // ...
    }

    // More methods about user management
    // ...
}
```

EJERCICIO 3

```
public void ProcessData()
{
    // Lots of logic here...
    int x = GetData();
    int y = x + 10;
    // More code...
    if (y > 50)
    {
        // Lots of logic here...
    }
    else
    {
        // Lots of logic here...
    }
    // Even more code...
    SaveData(y);
}
```

EJERCICIO 4

```
public void ValidateUserInput(string input)
{
    // Code for validating input
    // ...
}

public void ValidateUserPassword(string password)
{
    // Code very similar to input validation
    // ...
}
```

EJERCICIO 5

```
public class OldUnusedClass
{
    // Lots of unused code here...
}
```

EJERCICIO 6

```
public class Superclass
{
    public virtual void DoWork()
    {
        // Do some work
    }
}

public class Subclass : Superclass
{
    public override void DoWork()
    {
        // Do completely different work, unrelated to the superclass's work
    }
}
```

EJERCICIO 7

```
public class DataProcessor
{
    // This is a specific library or tool used everywhere
    SpecificLibrary library = new SpecificLibrary();

    public void ProcessData(List<int> data)
    {
        library.Method1(data);
        library.Method2(data);
        library.Method3(data);
    }
}
```

EJERCICIO 8

```
public class MyClass
{
    public void DoManyThings()
    {
        // Lot of code for task 1
        // ...
        // Lot of code for task 2
        // ...
        // Lot of code for task 3
        // ...
        // Lot of code for task 4
        // ...
    }
}
```

EJERCICIO 9

```
public double CalculateNetSalary(double grossSalary)
{
    double taxRate;
    double netSalary;

    if (grossSalary > 10000)
    {
        taxRate = 0.3;
    }
    else if (grossSalary > 5000)
    {
        taxRate = 0.2;
    }
    else
    {
        taxRate = 0.1;
    }

    netSalary = grossSalary - (grossSalary * taxRate);

    if (netSalary < 0)
    {
        netSalary = 0;
    }

    return netSalary;
}
```

EJERCICIO 10

```
public class ShoppingCart
{
    private Dictionary<Product, int> _items = new Dictionary<Product, int>();

    public void AddProduct(Product product, int quantity)
    {
        if (_items.ContainsKey(product))
        {
            _items[product] += quantity;
        }
        else
        {
            _items.Add(product, quantity);
        }
    }

    // ... Otros métodos ...
}
```

Luego de unas iteraciones dev-test:

```
public class ShoppingCart
{
    private Dictionary<Product, int> _items = new Dictionary<Product, int>();
    private const int MAX_QUANTITY = 10;

    public void AddProduct(Product product, int quantity)
    {
        if (quantity > MAX_QUANTITY)
        {
            throw new ArgumentException($"Can't add more than {MAX_QUANTITY} of a product at once");
        }

        if (_items.ContainsKey(product))
        {
            _items[product] += quantity;
        }
        else
        {
            _items.Add(product, quantity);
        }
    }

    // ... Otros métodos ...
}
```