# Redesigned the project structure:

* Separate code in **business logic** and **UI** (client) logic.
* Rename class **Person** to **Player.**
* Rename **GameBoard** to **BalloonBoard**.
* Remove **Command** class and implement **Command** **Pattern.**
* Rename **Coordinates** class to **Balloon**.
* Remove class **PersonScoreComparer**.
* Rename class **TopScore** to **ScoreBoard**.
* Remove class **baloncheta** and redesign the code to separate files.
* Rename solution from **baloncheta** to **BalloonsPop**.
* Create project **BalloonsPop.Common** which holds the business logic.
* Create folder **Components** in **BalloonsPop.Common** which holding the base components in the game.
* Create folder **Contracts** in **BalloonsPop.Common** which holding all interfaces in the game.
* Create folder **Entities** in **BalloonsPop.Common** which holding real world class objects.
* Create folder **Utilities** in **BalloonsPop.Common** which holding helping classes.
* Create project **BalloonsPop.ConsoleUI** which holds the console UI logic.
* Create project **BalloonsPop.UnitTest** which holds the unit test for the application.

# Used design patterns:

* **Structural:**
* Decorator
* Proxy
* **Behavior:**
* Strategy
* Command
* **Creational:**
* Singleton
* Prototype

# Introducing new classes by namespace:

## BalloonsPop.Common

* Components
* **BalloonGameEngine** is a default game engine for the game.
* **BalloonTypes** is an enumeration of balloon types.
* **CommandTypes** is an enumeration of different commands.
* **Directions** is an enumeration of all possible directions.
* **ScoreBoard** is internal class which holds all of the data for the current score.
  + Patterns
* **BalloonGameCommand** is a default specific command class which is responsible to call the receiver of the command.
* **BalloonPrototype** is abstract class who gives clone method to the derived classes.
* **Decorator** is class who derived the base class **Playground** and holds in property the information about it. It’s part of the decorator pattern.
* **Playground** is an abstract base class which has property field.
* **ScoreBoardProxy** is a part of Proxy pattern and it’s visible outside of the library.
* **Shootable** is a current implementation of **Playground** who has methods that gives options for shooting the board.
* Contracts
* **IBalloon** gives information about creating balloon.
* **ICommand** gives information about creating command.
* **ICommandInvoker** gives information about creating specific command **invoker.**
* **IGameEngine** gives option to create your own game engine.
* **IGameReader** gives option to create specific game reader which readsuser’s inputs.
* **IGameRender** gives option to implement different presentation of the game etc. Windows Forms, WPF, ASP.NET.
* **IPlayer** gives option to create your own implementation of player.
* **IPlayGroundItem** gives option to create your own playground item to do not depend of Balloon at all.
* **IRandomGenerator** gives option to create own random generator.
* **IScoreBoard** gives option to create your own score board.
* Entities
* **Balloon** is a current implementation of **IBalloon** which is an internal default playground item.
* **BalloonBoard** is a current implementation of **IPlayground** and is a default Playground.
* **Player** is a current implementation of **IPlayer** and is a default player for the game.
* Utilities
* **RandomGenerator** is a default current implementation of **IRandomGenerator** and he cares for generate random numbers.
* **Utils** is a static class with specific static methods which are helping in different cases.
  + Extensions
* **BalloonExtensions** is an extension class for **Balloon** class which holds methods who shouldn’t be in the Balloon.
* **ShootableExtensions** is an extension class for **Shootable** class and holds methods who shouldn’t be in **Shootable**.

## BalloonsPop.ConsoleUI

* **CommandInvoker** is a specific invoker created by the client.
* **ConsoleReader** is a class created by the client which is specific for Console UI.
* **ConsoleRender** is a specific console render class created by the client.
* **Engine** is the main class which start the program.
* **ScoreHandler** is a helper class with responsibilities to get and set the result.

# Reformatted the source code:

The entire code was reformatted with best practices. We used StyleCope and JustCode to validate all of the misses.

# Renaming:

All classes, namespaces, variables, constants, fields, properties was correctly reformatted following the best practices.

# Introduced constants:

All magic strings and numbers were moved to constants with proper name and reused.

# Followed principles:

* **SOLID**
* **DRY**
* **KISS**
* **YAGNI**