# Refactoring Documentation for Project “ Balloons Pops”

1. Redesigned the project structure:

* Renamed the project and the solution to BalloonsPopsGame
* Renamed the namespace to BalloonsPopsGame
* Removed useless empty destructors
* Renamed GameState class to Game
* Renamed ballonsState class to BalloonsContainer
* Added new BalloonsPopsGame.Balloons namespace where is all the creational, storage and popping logic for the balloons is helded

1. Reformatted the source code:

* Removed all unneeded empty lines.
* Inserted empty lines between the methods.
* Formatted the curly braces { and } according to the best practices for the C# language.
* Put { and } after all conditionals and loops (when missing).
* Inserted the usings into the namespace
* Character casing: variables and fields made camelCase; types and methods made PascalCase.

1. Renamed variables:

* In class BalloonsContainer: poleto -> ballons.
* In class BalloonsContainer:cnt -> numberOfTurn.
* In class Game: \_st -> ballons

1. Renamed methods:

* In class BalloonsContainer: kraj -> IsContainerEmpty.
* In class BalloonsContainer: printArray -> DispayBalloons.
* In class BalloonsContainer: pr -> ConvertBallonToChar.
* In class BalloonsContainer: PopBallon -> PopBallons.
* In class Game: sendCommand -> PopBallons.

1. Introduced constants:

* In class BalloonsContainer: NumberOfRows.
* In class BalloonsContainer: NumberOfColumns.
* In class BalloonsContainer: NumberOfBalloonColors.

1. Introduced Properties:

* In class BalloonsContainer: NumberOfTurn.
* In class BalloonsContainer: Balloons
* In class Game: Scoreboard.
* In class Game: Balloons.
* In class Game: UIGenerator.

1. Moved the property NumberOfTurn into the Game class.
2. Removed unnecessary comments:
3. Added access modifiers to all methods:
4. Changed methods:

* In class BallonsContainer -> PopBallons: Changed the return type of the method to void. Removed the logic for checking if the container is empty from him. Added logic for checking if the balloon position is valid.
* In class Game -> PopBalloons: Removed the logic for validating the balloon position. Made the method to update the scoreboard only after the end of the game.
* In class Game -> ExecuteCommand: Fixed the parsing of the input when entering balloon position.

1. Extracted methods:

* In class BallonsContainer placed the logic for filling the container into method Fill.

1. Made new methods:

* In class Game -> Start: moved the endless loop and command execution logic in it.

1. Introduced new class ScoreBoard and moved all related functionality in it.
2. Introduced new interface UIGenerator and new class ConsoleUIGenerator and moved the logic for creating UI into them:

* Moved the DisplayBalloons method from BalloonsContainer class.
* Created DispayMessage method for writing messages to the user.

1. Introduced new interface IRandomNumberProvider and new class StandardRandomNumberProvider and moved the functionality for creating random numbers into them.
2. Introduced new InvalidRowOrColumnExceptionand MissingBalloonException:

* Made PopBallons method in BallonsContainer to throw these exceptions instead of writing on the console.

1. Implemented the Singleton design pattern:

* Made the Game class a singleton one.
* Added Instance property for returning the static instance of Game class.

1. Implemented the Factory Method design pattern:

* Made balloon creation with Factory Method.
* Created new Balloon, BalloonCreator, BlueBalloon, BlueBalloonCreator, RedBalloon, RedBalloonCreator, GreenBalloon, GreenBalloonCreator, YellowBalloon, YellowBalloonCreator, PoppedBalloon and PoppedBalloonCreator classes.
* Created new namespace BalloonsPopsGame.Balloons.Creator.

1. Implemented the Flyweight design pattern:

* Made balloon storage with Flyweight pattern.
* Create new IBalloonFactory interface, BalloonFactory class and BalloonType enumeration.

1. Implemented the Iterator pattern:

* Made BalloonsContainer to implement IEnumerable<Balloon>.

1. Implemented the Observer pattern:

* Added ContainerChanged event to the BalloonsContainer and protected void OnContainerChanged method.
* UIGenerator subscribes to the ContainerChanged event.