The assignment for the second iteration consists of two parts:

1) You shall implement Undo and Redo functionality in the Blockr application: the user shall be able to undo any modification of the program in the Program Area and any change to the execution state of the program (together with the corresponding change in the game world) by pressing Ctrl+Z. As in most applications, the user shall be able to undo multiple changes by pressing Ctrl+Z repeatedly; the user shall also be able to undo the most recent Undo operation (this is known as a Redo) by pressing Shift+Ctrl+Z; and the user shall be able to Redo all of the modifications that were undone, except for the ones that happened before the most recent new original modification, by pressing Shift+Ctrl+Z repeatedly.

2) You shall modularize Blockr.
Specifically, you shall factor the block programming logic and the robot-wall-grid logic into separate components, separated by a Game World API.
The Game World API shall not involve any block programming concepts. It shall be possible to implement this API with different kinds of game worlds, and to consume this API from different kinds of applications. To prove this, you shall develop, in addition to the Blockr application.

Correspondingly, instead of delivering a single monolithic blockr.jar executable, you shall design and deliver the following:

- a **Game World API**, probably consisting mostly of interfaces and perhaps enums, compiled, with no dependencies, into gameworldapi.jar.

- a **Robot Game World** component that implements the Game World API and depends only on gameworldapi.jar, and does not depend in any way on block programming concepts or artifacts. Compiled into robotgame.jar with only gameworldapi.jar in the classpath.

- the **Blockr application**. Its only compile-time dependency is the Game World API. It takes the class name of the root class of a Game World API implementation (which implements an appropriate Game World API interface) as a command-line argument. It uses Class.forName and Constructor.newInstance to create an instance and call the Game World API interface methods on it. It does not depend in any way on robot, wall, or grid concepts or artifacts. Compiled into blockr.jar with only gameworldapi.jar in the classpath.

Develop each component in its own Eclipse/IntelliJ project. Make sure each project declares only permitted dependencies on other projects in its Build Path. Specifically: gameworldapi shall have no dependencies, and the other components shall have only gameworldapi as a dependency.

The Game World API defines a GameWorldType interface, that offers methods to:

- retrieve the list of Actions supported by the GameWorldType. For Robot, this is: Turn Left, Turn Right, Move Forward.

- retrieve the list of Predicates supported by the GameWorldType. For Robot, this is: WallInFront.

- create a new game world instance, which implements interface GameWorld.

The GameWorld interface offers methods to:

- perform one of the supported Actions. It returns a result indicating successful execution, failure to execute because the action is illegal in the current state, or end of the game due to reaching the goal state.

- evaluate one of the supported Predicates.

- create an (opaque, i.e. non-inspectable) snapshot of the game world state and to restore the game world state to a given snapshot.

- paint the current state of the game world, given a graphics object (either java.awt.Graphics or another graphics API of your own design).

A regular assignment document with details on what to hand in, etc., follows later.

Deadline: Friday, 24 April 2020, 15:30pm.