Design Rationale (REQ 2)

HighGround class created and extended by *Tree* class and *Wall* class. Since the different stages of the tree and wall allow the player to jump onto it. To follow the Don't *Repeat Yourself Principle,* all the implementation that relates to the jump action will be implemented inside the *HighGround* class. By doing this, it will also be easy for maintenance or extension if there is another high ground object added into this game.

HighGround---<<*creates*>>---> *JumpAction*. In this game, there are different high ground objects such as trees and walls that allow the player to perform jump action on them. Based on object-oriented, the tree and wall are the objects. We can simply create JumpAction inside the Player class, however by doing this, we need to know which object the player wants to jump, it will require additional dependency between Player and Tree and Wall.

Besides, we also need to check whether the ground allows the player to jump onto it or not, checking the object classes using the if-else statement will also increase dependency. To align our design with the Reduce Dependency Principle, we discard this alternative. *HighGround*---<<*creates*>>---> *JumpAction* and *HighGround* implements *Jumpable* can avoid checking the objects that the player wants to jump using the if-else statement.

Different stages of Tree and Wall have different success rates that Player can jump onto it and if the player fails to jump onto it, they will cause different amounts of damage on the player. Based on the open-closed principle, classes should be open for extension but closed for modification. This is the rationale behind the presence of the Jumpable interface. By doing this, the classes implementing it will have the same method but different implementation, even though there is another high ground that will be added into our game in future.