# Team Liza
# Milestone 5

Sam Kim
Kevin Geisler
Michael Williamson
Brian Collins

2-10-12

# Contents

# 1 Introduction

Minecraft [1] is a sandbox computer game where players can create and remove blocks in a simulated world. These blocks can be arranged in a nearly unlimited number of ways and is only limited by the player's imagination. Despite being recently officially released, it has gained immense popularity with a user base exceeding 10 million people.

Modifying Minecraft [1] has become increasingly popular, as a number of changes can be made to suit each users needs. There are currently two methods of modifying Minecraft [1]. Mods require a user to directly modify their game files in order to add or alter functionality in Minecraft [1]. For a server to effectively use a mod, the server requires each user to install that mod. On the other hand, plugins enable developers to make changes to Minecraft [1] without needing each user to directly modify the platform. Only the server needs to be modified.

A common way of introducing plugins to a Minecraft [1] server is to utilize a tool called Bukkit [2]. Bukkit wraps around the official server application and exposes an easy application programming interface (API) for developers to create plugins.

Bukkit [2] currently does not possess any means of testing, which makes debugging plugins a tedious process. Liza intends to provide a unit testing framework for plugin developers to programmatically test their code. This document will briefly summarize the problem with the current system, and provide details on how Liza intends to solve them. This will be a compilation of all of the previous milestones, and include the need, features, requirements, and details about the development of the system.

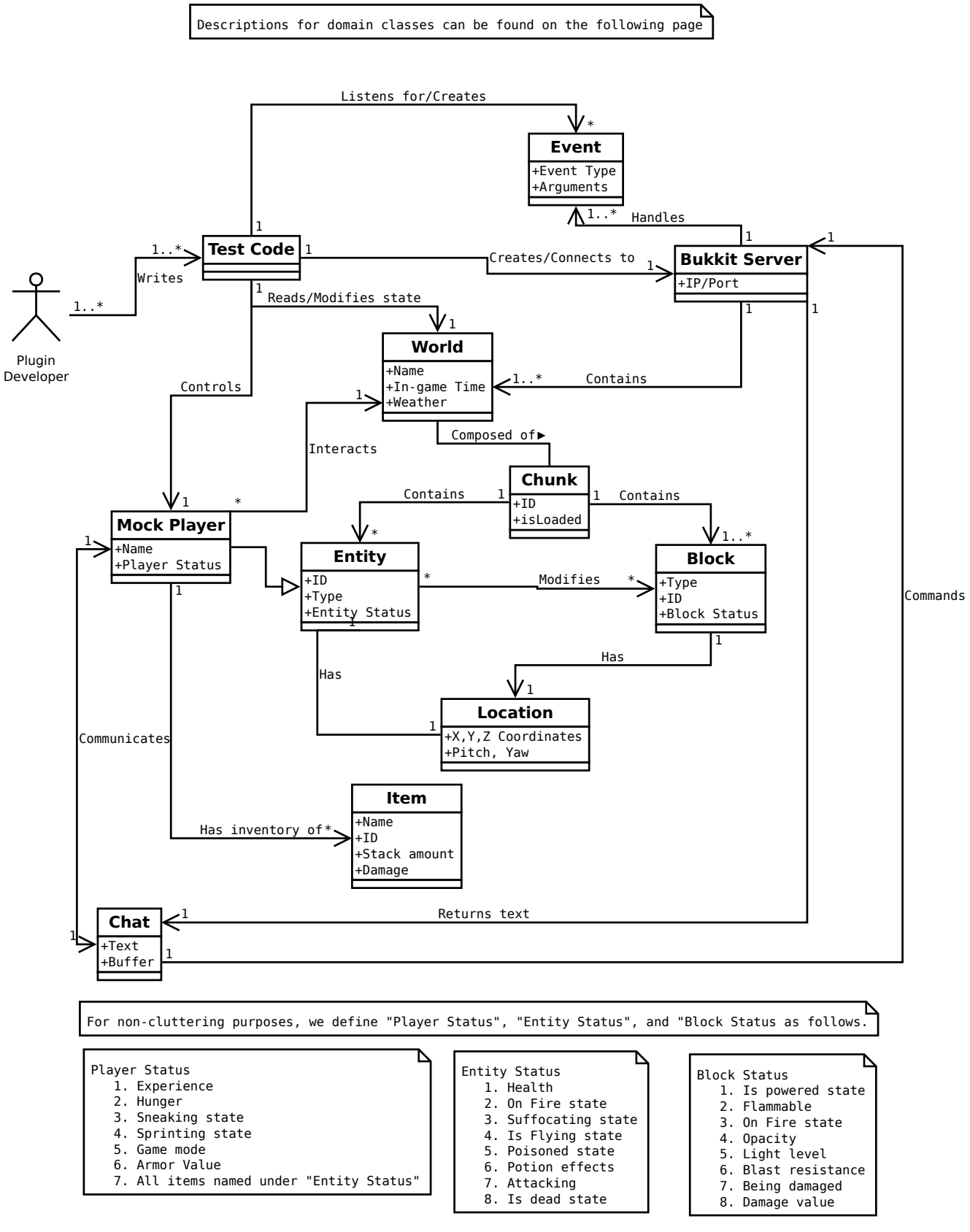
# 2 Analysis Models

## 2.1 Domain Model

This is the final design for the Liza Unit Testing Framework. Our domain model shows the communication between the developers test code and Liza Classes which will interact with Bukkit. This shows the basic design of Liza without the complexity of the complete Bukkit-Liza Class Diagram.

The diagram is depicted below:

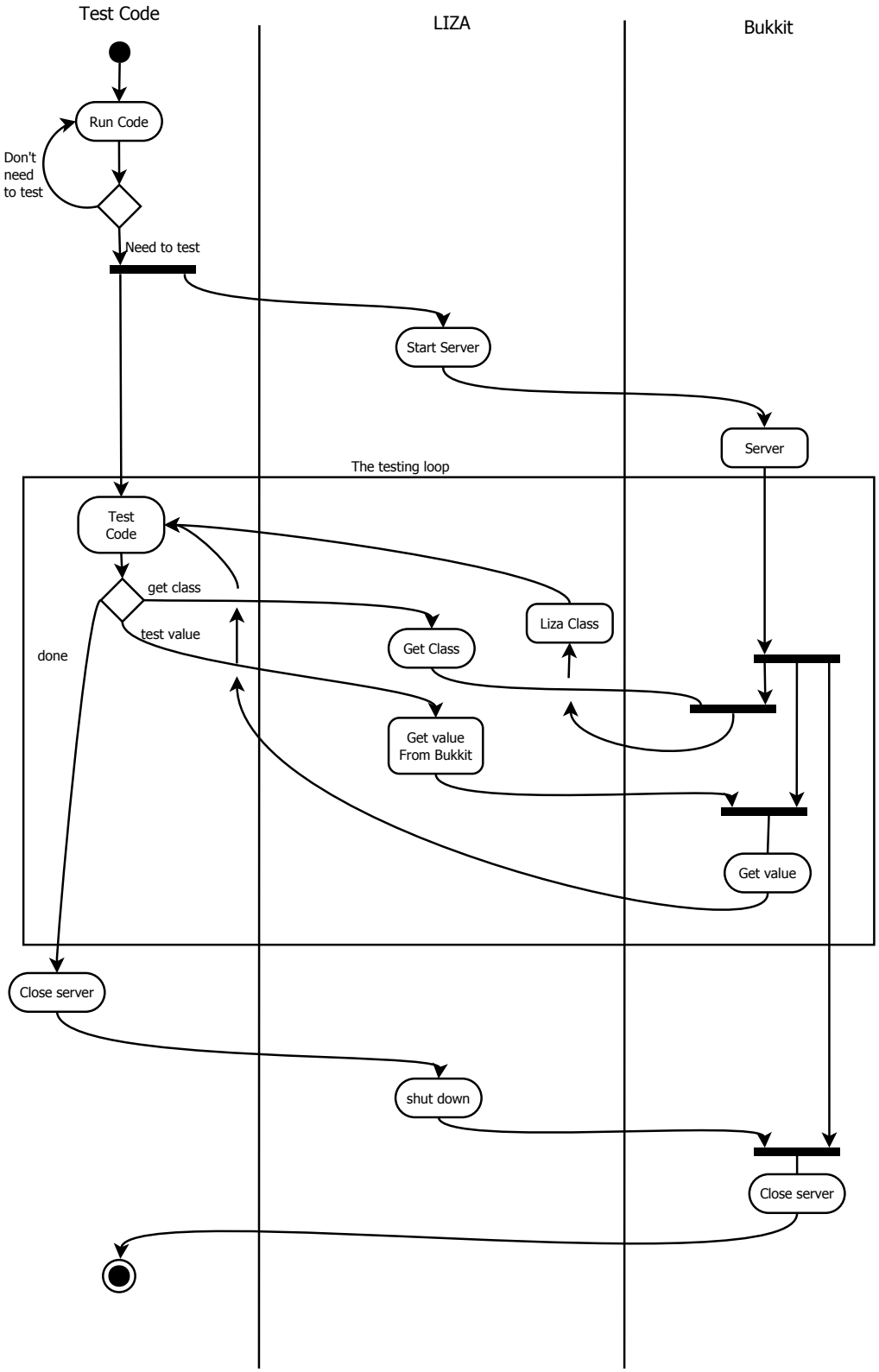
# Liza Domain Model



## 2.2 Activity Diagram

This diagram shows how the Developer's test code, the Liza interface, and Bukkit interactall together. There is a large amount of activity that goes through a massive loop, shown in the rectangular box. This loop is the main guts of what Liza does when running.

LIZA
Activity Diagram



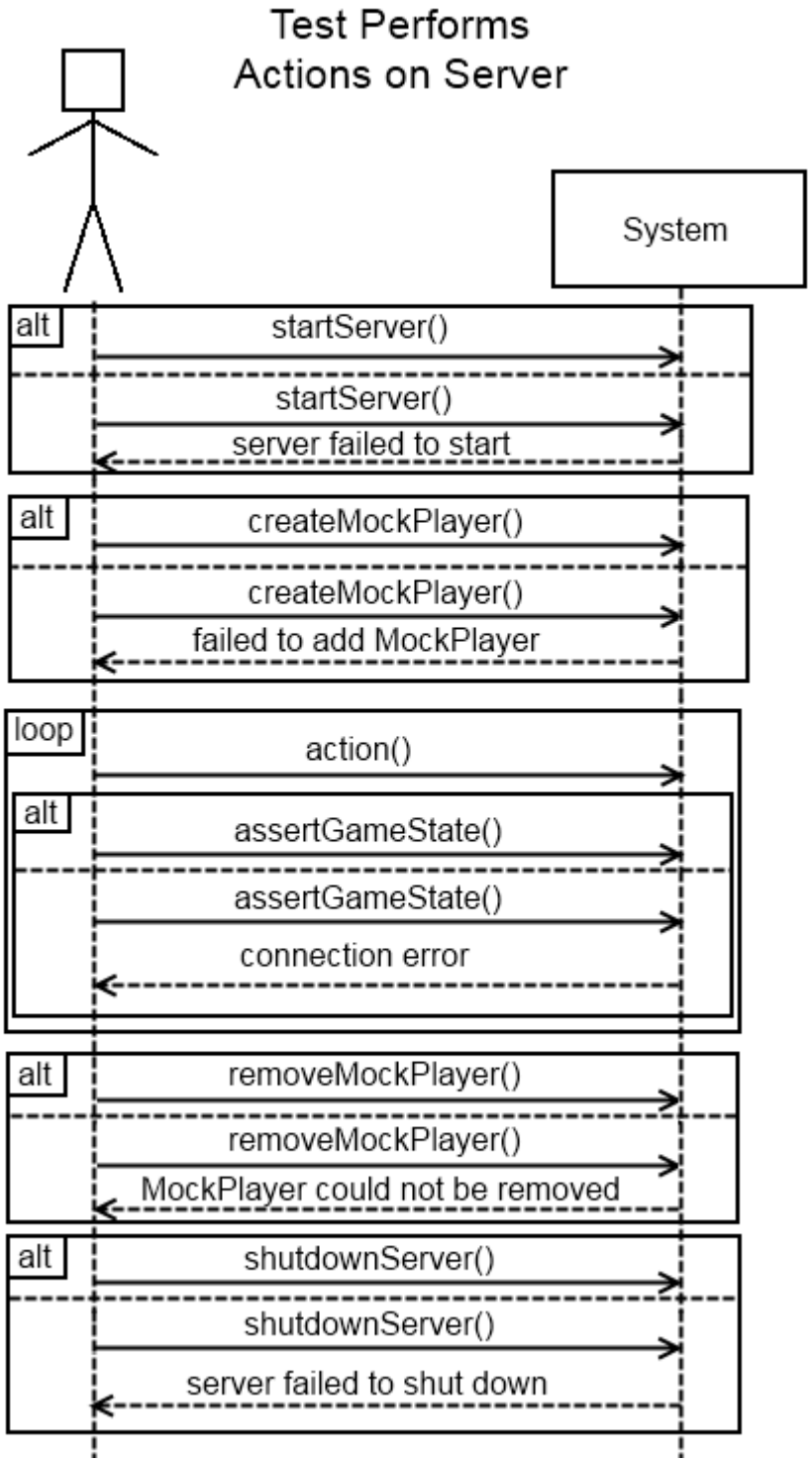
# 3 Logical Architecture

## 3.1 System Sequence Diagrams (SSD)

Description:

There are many SSDs that will apply to this project – one for each assertion. However, each of these will follow the exact same format. Instead, we produced a single SSD that details the flow of a test case that a developer may write. This includes general exception cases and alternate paths which could be produced.

The SSD is shown below:


Test Performs
Actions on Server
System
alt
startServer()
startServer()
server failed to start
alt
createMockPlayer()
createMockPlayer()
failed to add MockPlayer
loop
action()
alt
assertGameState()
assertGameState()
connection error
alt
removeMockPlayer()
removeMockPlayer()
MockPlayer could not be removed
alt
shutdownServer()
shutdownServer()
server failed to shut down

## 3.2 Operational Contracts (OC)

There will be a controller in Liza to start the server that it will control. Here is an operational contract which describes the startup process.:

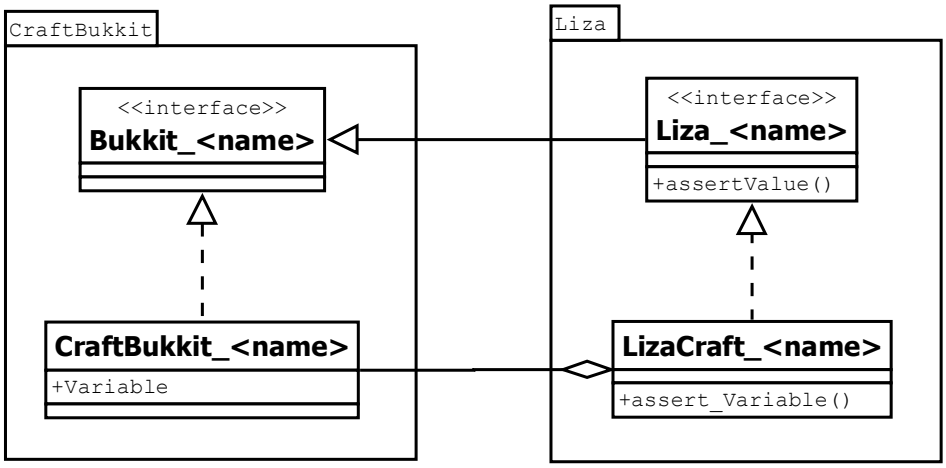
**Operation name:** startServer()
**Cross-References:** SSD1
**Preconditions:** LizaCraftController has been created.
**Postconditions:**

1. CraftBukkitThread was created.
2. CraftBukkitThread was run
3. CraftServer object was retrieved from CraftBukkitThread
4. EventListener was created
5. Association between EventListener and CraftServer established

## 3.3 Package Diagram

There exists a degree of separation between our system and Bukkit, in the sense that our system interacts with Bukkit, yet Bukkit is not aware of our system. However, all of the classes in our system will all belong to the same package, due to the high amount of interaction between classes. Because of this, a package diagram is not applicable. However, we have included a diagram which depicts the relationship between Liza and Bukkit in general. For each Bukkit class that is relavent, Liza will extend a new interface and create a new LizaCraft class that will implement it.

# Bukkit-Liza Relationship



### 3.4 Class Diagram

Description:

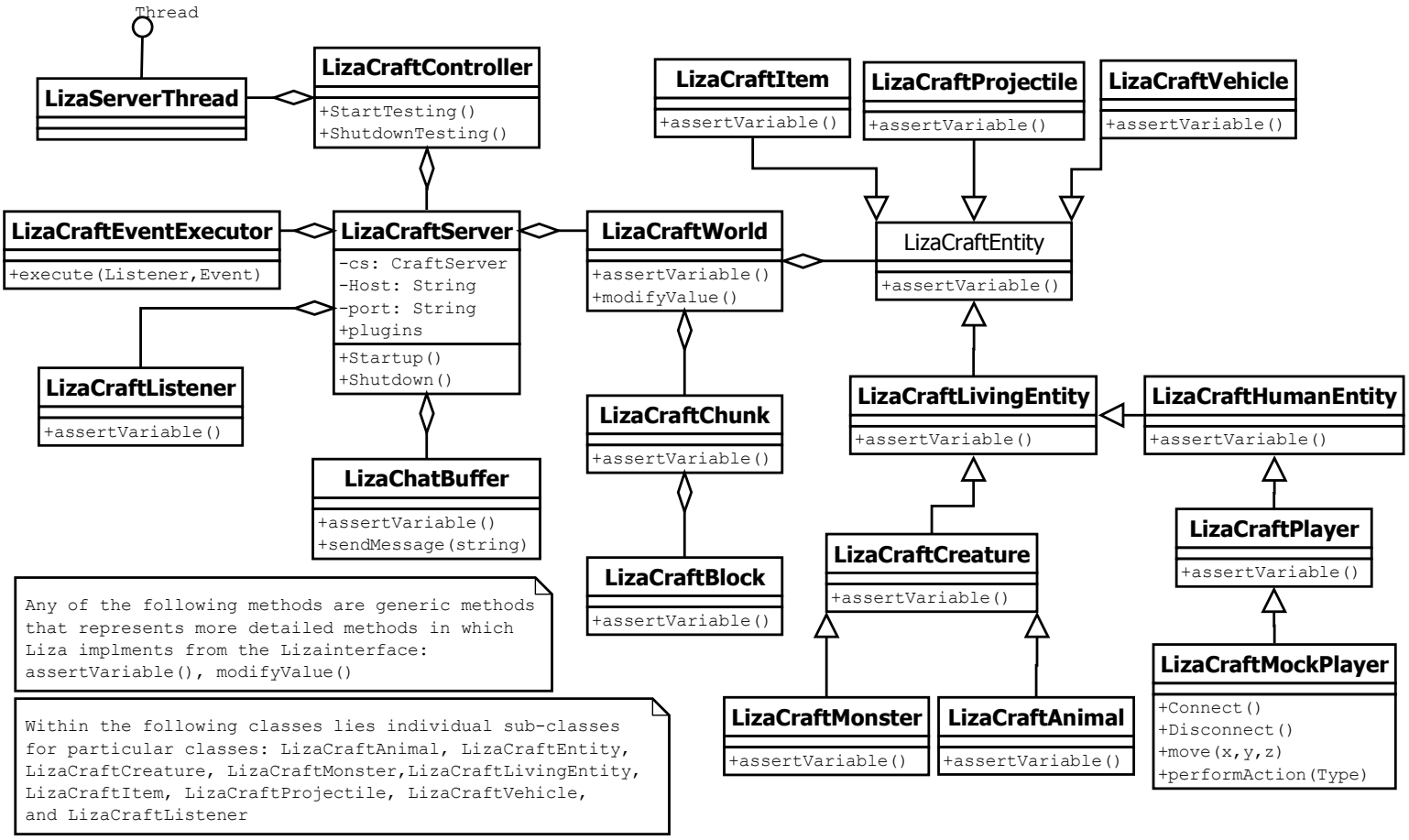
Bukkit has two main layers: Bukkit, which is a collection of interfaces that a plugin developer utilizes, and CraftBukkit, which is the implementation of those interfaces. Our system will mirror this design. There is Liza, which is a set of interfaces that inherit the Bukkit interfaces, and LizaCraft, which is the implementation of the Liza interfaces and extends the CraftBukkit classes. By extending the existing Bukkit and CraftBukkit, we present the API that the developers are accustomed to, along with any new methods that may be of use for testing, such as asserting properties.

There are a few new classes, however. LizaMockPlayer and its associated implementation represent the automated player that the developer commands. LizaMockPlayer and LizaPlayer are separated because the latter asserts properties of any player, while the former controls only the automated player.

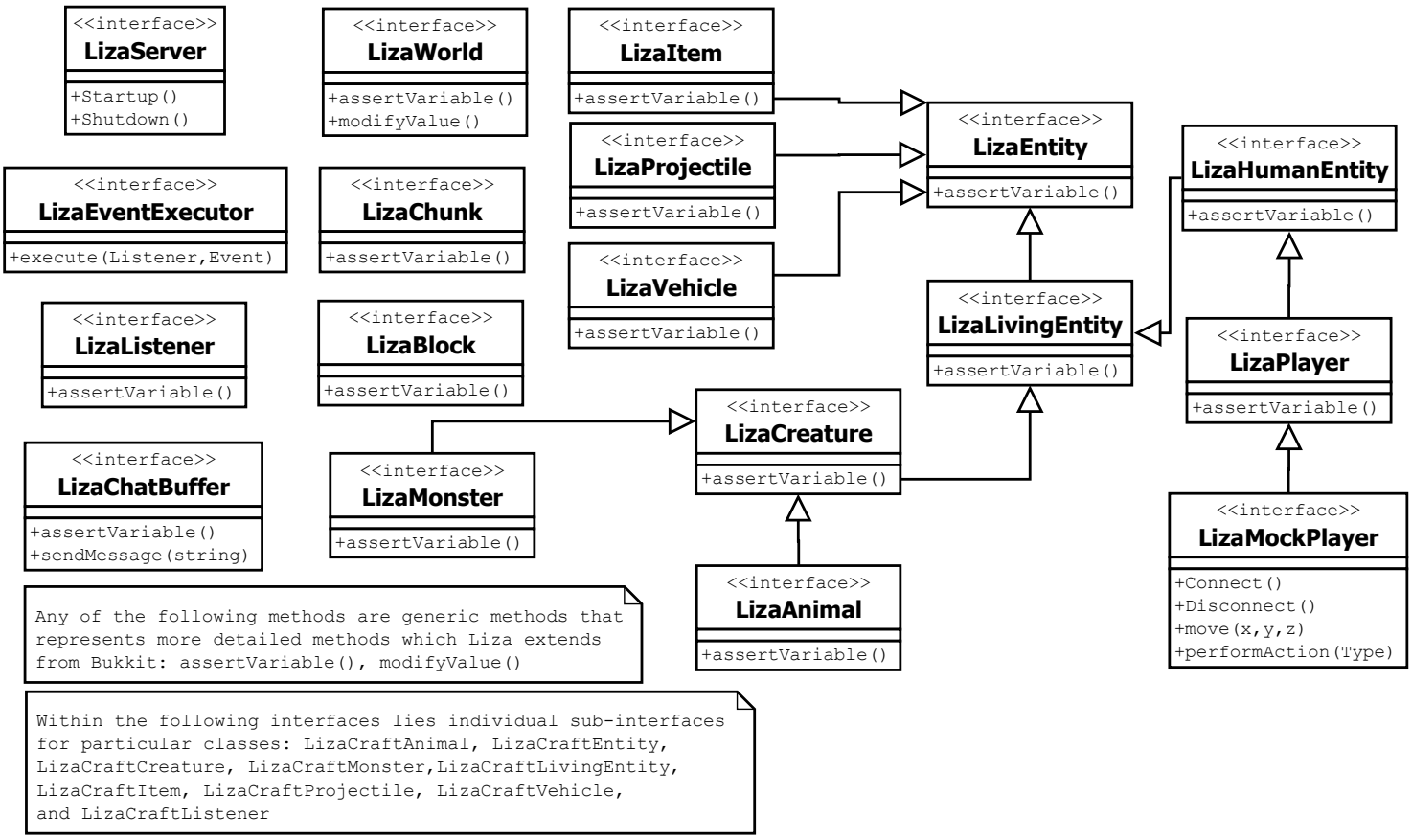
LizaListener and LizaEventExecutor handle the listening and spoofing of events, respectively.

The diagram is found on the following pages.

# LizaCraft Class Diagram



# Interface Class Diagram



# 4 Gang of Four Patterns

In this section, we will discuss which of the Gang of Four patterns apply to our design.

## 4.1 Adapter

As Liza is an extension of Bukkit, we felt it necessary to mirror Bukkit's design to our own. This involved making Liza versions of many classes, as noted in the package overview diagram. To reduce coupling with Bukkit, we instead chose to make each of our classes an adapter to the appropriate Bukkit class.

## 4.2 Façade

Liza is responsible for starting up the server, retrieving the server object and enabling event listening, among other tasks. These are rather complex control sequences. Therefore, to hide this away, we've created a *LizaTestModule* class, an application of the Façade pattern. The code to the developer simply becomes

```
LizaTestModule test = new LizaTestModule();
test.start();
test.enableEvents();
// code work here
test.disableEvents();
test.end();
```

## 4.3 Observer

Minecraft, and by association Bukkit, has nearly every action driven by player input. The game handles these actions by using event handlers.

We feel that a test developer will also be interested in these events. Therefore, we have added the *LizaPlugin* class. It itself is an observer – it is able to register events with Bukkit's event dispatcher, and gets notified when such events occur.

However, we also open up the test developer to observe the *LizaPlugin*. A developer will register an event to listen for through the class. Then the *LizaPlugin* will register that event to Bukkit. When the event occurs, we relay that event to the developer.

# 5 Acceptance Test Plan

It is vital for any testing framework to work reliably. Because of this, we will be using Bukkit's API to assert that the mock player's states are correct through a testing plugin. In general, we will be using Bukkit's API (which is assumed to be correct) to check against Liza's values. For controls sake, the test plugin will be run on a world that is generated to be completely flat, with no obstacles or pitfalls, and natural entity spawning disabled. If a test case would require such things, the testing plugin will be able to spawn them in manually.

In the descriptions, the terms testing plugin (that will be used to test Liza) and Bukkit will be used interchangeably.

The Acceptance Test Plan is as shown below:

| Test Case No. | Description and Expected Result | Support |
|---|---|---|
| 1 | Bukkit [2] can retrieve a list of all connected players. The test will begin by retrieving this list. Liza will attempt to log in the mock player. The list is retrieved again. The mock player is expected to be added to this list. | Yes |
| 2 | Through Bukkit [2], a message can be sent directly to the mock player. Liza will search through the buffer and attempt to find this message. | No |
| 3 | Liza will attempt to send a command through the chat. Bukkit [2] has a callback function which retrieves any commands that are sent along with the sender. We can assert that the command was sent correctly and by the mock player. | No |
| 4 | Liza will send a command through the chat to the server. The testing plugin will read this command, and send out a predetermined event. It will be asserted that this event was received, and that all the values match. | Yes |
| 5 | The testing plugin will also come with an additional plugin which makes dirt indestructible. Liza will attempt to destroy this block. Bukkit [2] will assert that the dirt block still exists. Liza will attempt to disable this plugin, and destroy the block again. Bukkit [2] will assert that the block is destroyed, and will replace it. Liza will attempt to re-enable the plugin, and destroy the block. Bukkit [2] will once again assert that the block still exists. | No |
| 6 | Bukkit [2] will retrieve the current location of the mock player. Liza will attempt to move the player in a specified direction or teleport to a location. The testing plugin will assert that the mock player is in the correct location. This test will be repeated multiple times for every direction. | No |
| 7 | Bukkit [2] keeps track of the pitch and the yaw of a player. Given the player's location along with the target point, the expected viewing angles can be calculated. Bukkit [2] will retrieve the player's viewing angles and compare them to the expected values. | Yes |
| 8 | Bukkit will encase the mock player in water. Then the testing plugin will assert swimming much like it does with regular movement. | No |
| 9 | Liza will attempt to send the command to sneak. Bukkit can read the player's sneaking state. A similar process follows for sprinting. | Yes |
| 10 | Bukkit [2] will enable "creative mode" for the mock player. This will allow the mock player to fly. Bukkit [2] will retrieve the current location of the mock player. Liza will attempt to fly the player in a specified direction. The testing plugin will assert that the mock player is in the correct location. This test will be repeated multiple times for every direction. | No |

| | | |
|---|---|---|
| 11 | Liza will attempt to place a block. The testing plugin will assert that the block exists. Liza will attempt to destroy that block. The testing plugin will assert that the block does not exist. | Yes |
| 12 | The testing plugin will spawn an entity at the mock player's location. Liza will proceed to attack this entity. This will trigger an event, which the testing plugin will retrieve. | No |
| 13 | The testing plugin will retrieve the mock player's inventory, and sets it so that a target item is not currently possessed by the mock player. The testing plugin will then proceed to drop that item onto the ground near the player. The mock player's inventory is once again retrieved, and the presence of the item is asserted. | No |
| 14 | The testing plugin will give the mock player a specific item, such as a bow and arrow. Liza will attempt to use this item. Bukkit [2] will respond by throwing an event, which the testing plugin will receive. | No |
| 15 | Bukkit [2] will retrieve a list of entities and their location on a server. Liza will attempt to add an entity (such as a cow) at a certain location. The testing plugin will assert that the list of entities has been modified and an entity is located at the determined place. Liza will then attempt to despawn the entity, and the testing plugin will assert that the list has been modified and that there is no entity at the specific location. | Yes |
| 16 | Liza will attempt to retrieve an entity's health and location. Bukkit [2] will retrieve the same entity's health and the testing plugin will assert that they are the same. | Yes |
| 17 | Liza will attempt to retrieve a list of all entities in the Minecraft [1] world. Bukkit [2] will retrieve the list off of the server. The two lists will be compared and asserted to have the same values. | Yes |
| 18 | Liza will attempt to retrieve a list of all entities in a predetermined radius from the mock player. Bukkit [2] will retrieve the list of all entities and their locations, and will calculate and find those that are within the same specified radius of the mock player. The two lists will be compared and asserted to have the same values. | No |
| 19 | Liza will attempt to remove all entities within a predetermined radius from the mock player, while generating a list of removed entities. The testing plugin will retrieve the list of all entities on the server, and assert that none of the removed entities are present. | No |
| 20 | The testing plugin will give a predetermined special condition (such as being on fire) to a specified entity. Liza will attempt to retrieve this information, and the testing plugin will assert that they are the same. | Yes |
| 21 | Both Bukkit [2] and Liza will retrieve the list of equipped items on the mock player, and run calculations to find the armor value. The two are compared. Then the mock player will change its | No |

| | | |
|---|---|---|
| | armor configuration by removing/equipping armor. Bukkit [2] and Liza will once again retrieve the equipped items and compare computed armor values. | |
| 22 | The mock player will wait until hunger is partially depleted. Bukkit [2] will retrieve the hunger value of the player, and assert that Liza's value matches. This hunger value will be altered, either by eating food or waiting longer, and Bukkit [2] and Liza compare values again. | No |
| 23 | Before killing any entities, the testing plugin will assert that the mock player has zero experience. Then, after killing some entities, the testing plugin will retrieve the player's experience and assert that it matches Liza's value. | No |
| 24 | The testing plugin will clear the mock player's inventory. Liza will give the mock player a predetermined set of items. Bukkit [2] will read the item values in the player's inventory and assert that each item is present. | No |
| 25 | Liza will attempt to create a block at a given location. The testing plugin will assert that the block exists. Removal is achieved by creating an air block at a location. | Yes |
| 26 | Liza and the testing plugin will retrieve a block at a given location. The two are compared and asserted to be the same. | Yes |
| 27 | Bukkit [2] is able to retrieve blocks that a player is facing. This value is compared to Liza's and asserted to be the same. | Yes |
| 28 | Given a block, the value that the testing framework and Liza retrieves are compared and asserted to be the same. | Yes |
| 29 | Given a block, Liza will attempt to set some material properties. The testing plugin will retrieve the block and read its properties, and assert that they have been set. | Yes |
| 30 | Liza will attempt to set the time of day in the world. The testing plugin will assert that the time matches what was set. | Yes |
| 31 | Liza will attempt to adjust the current weather at a location. Bukkit [2] will assert that it has been done. | Yes |