

Spike: 11**Title:** Messaging**Author:** Michael Williams, 7668481**Goals / deliverables:**

- Code
- To implement an easily expandable messaging system.
- To avoid refactoring old code as much as possible to support the new classes.
- Messaging Specification

Technologies, Tools, and Resources used:

- Visual Studio IDE
- www.cplusplus.com

Tasks undertaken:

- Decide on a method to implement the messaging system, including required functionality determined by the specification.
- Implement the required classes in order to support this.
- Research methods in reducing coupling between the objects and the messaging system.
- Implement these methods.

What we found out:

We found out how to implement a simple messaging system which can be easily expanded to add additional functionality such as incorporating an announcement or blackboard system through the use of a Message Handler to support sending messages.

The main goal we had in developing this was to have some way to reduce the coupling as much as possible so that a messageable object could just be one, and start sending and receiving messages as easily as possible. In the end, some methods need to be implemented by the child classes who decide to be a messageable object, but this is very minimal.

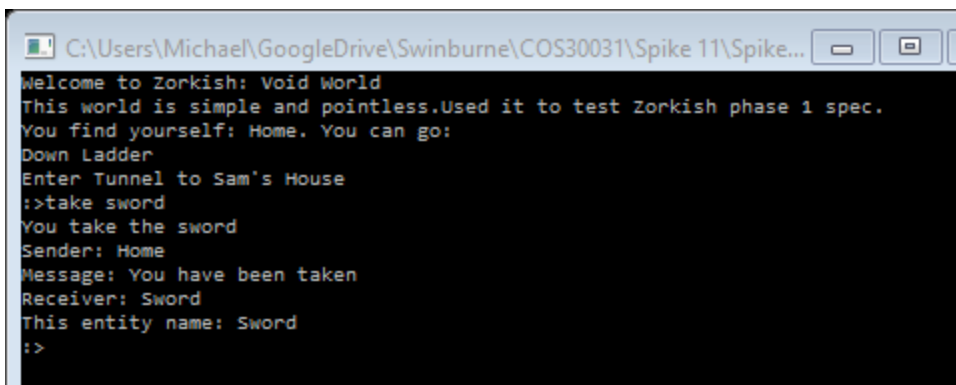
The MessageableEntity class handles every aspect to do with messaging, a child class need only provide a way to retrieve it's name (so it can be identified)

and a way to process a message. The default behaviour simply prints the contents of the message when it is received.

One challenge that we faced during this implementation was dealing with the circular dependency between the `MessageHandler` and the `Messageable_Entity` class. This is because the message handler needs to have a list of the registered entities, and the entities need to be able to register themselves with the handler. This issue was solved by using a forward declaration of the `Messageable_Entity` class.

Another issue we faced in achieving the coupling goal was, how do we allow each entity to know about the handler without passing a reference to each object through the constructor. If we didn't do this, then each time an entity is created we need to tell it who the handler is. Then we have to decide who is responsible for the handler. To solve this, we made a global `MessageHandler` declared in the `MessageHandler`. This way, anyone who has a need to use the message handler can simply include the header and have access to the handler directly. This approach even though it uses a global variable, is still controlled and is much more elegant than refactoring all of the old code.

The test harness set up works by sending a message to the entity who gets taken (using the `take _`) command. Below is the output showing the message contents:



```
C:\Users\Michael\GoogleDrive\Swinburne\COS30031\Spike 11\Spike...
Welcome to Zorkish: Void World
This world is simple and pointless.Used it to test Zorkish phase 1 spec.
You find yourself: Home. You can go:
Down Ladder
Enter Tunnel to Sam's House
:>take sword
You take the sword
Sender: Home
Message: You have been taken
Receiver: Sword
This entity name: Sword
:>
```