14/12/2017 Kyle James Watson STC905

Assignment

Software Architectures

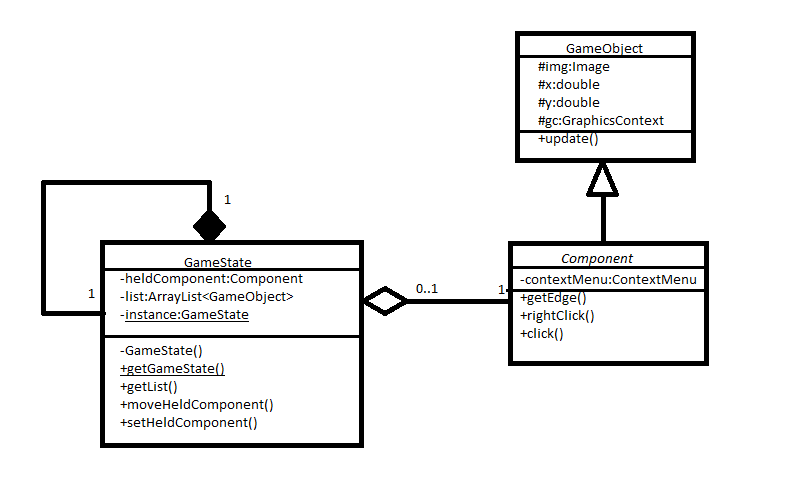
**Table of Contents**

[Design Patterns Used 1](#_Toc501047724)

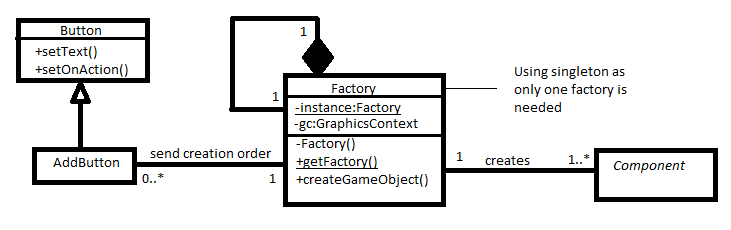
[Other Solutions 3](#_Toc501047725)

# **Introduction**

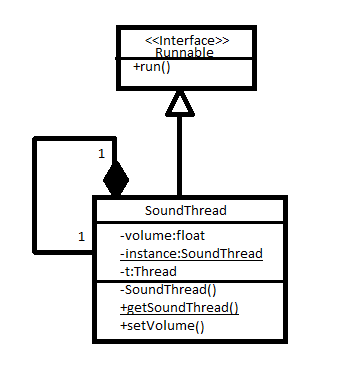
# **Design Patterns Used**

I decided to use a singleton design for 3 different classes where I felt it was only necessary to have one object initialised:

The *GameState* class would need to be accessed by multiple static classes as it is holding the display *list*. Having this as a singleton alows a static instance to be accessed by all the classes which keeps the display *list* synchronous for updates amongst the system.

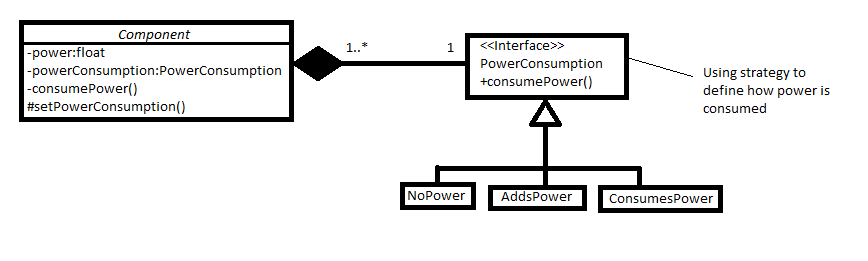


The *Factory* is accessed by multiple *AddButton* objects as they are the handlers used to create *Components*. There is no need to have multiple *Factories* for creating objects as they all would have the same state. The only field is *gc* which would only need to be different game wide if there were multiple *Canvasses*.

Due to the *Buzzer* being a component in the sandbox I decided a beep generating class would be needed.

After finding a solution from Stack Overflow (tangens 2009) I realised the generation would need to be multi-threaded otherwise the entire update would stutter whilst waiting for the sound to play.

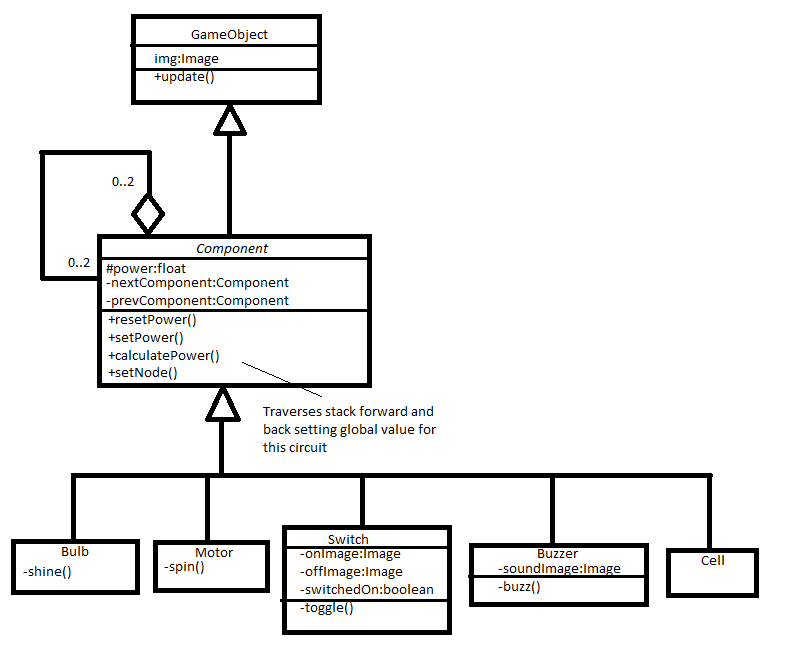
To stop multiple threads playing over eachother and crackling I created the *SoundThread* as a singleton that controlled the looping of the playback.



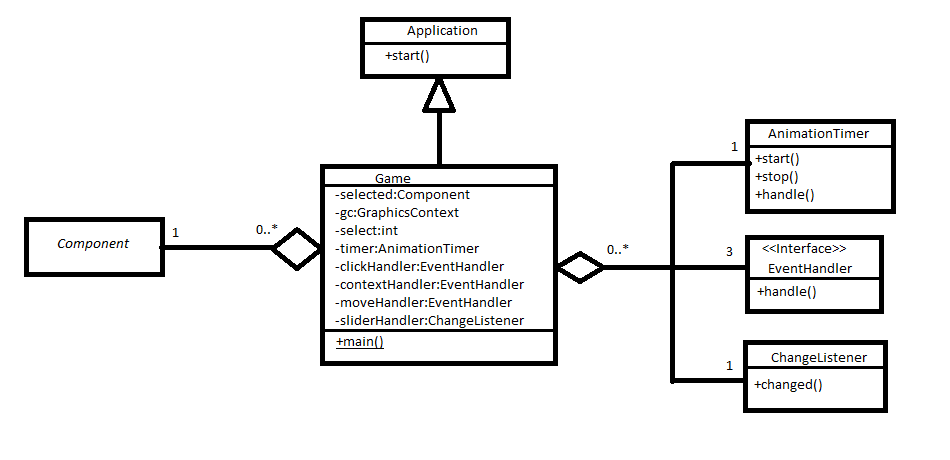
All *Components* in the circuit manipulated the power in one way or another so I decided to use the strategy design pattern to abstract this responsibility.

# 

# **Other Solutions**



Using an algorithm loosely based off a doubly linked list I created recursive functions to calculate, and then set, power for any potential circuits. This allows the *Components* to handle their own circuits resulting in multiple circuits being calculated regardless of their setup.

It was ideal for the *Game* class containing the entry point to the program to set up the UI as well as handle events fired by the UI. Any UI elements that needed to access components could do this through the *GameState* singleton.

# **Limitations of the Game**

The sandbox does not allow for parallel circuits which would be possible with a real life set of components. I felt implementation of this was not needed as, according to STEM Learning Limited (2016), primary school science focusses on how changing elements in a single series circuit can cause other changes for example “the brightness of bulbs, the loudness of buzzers and the on/off position of switches” (STEM 2016, para. 1).