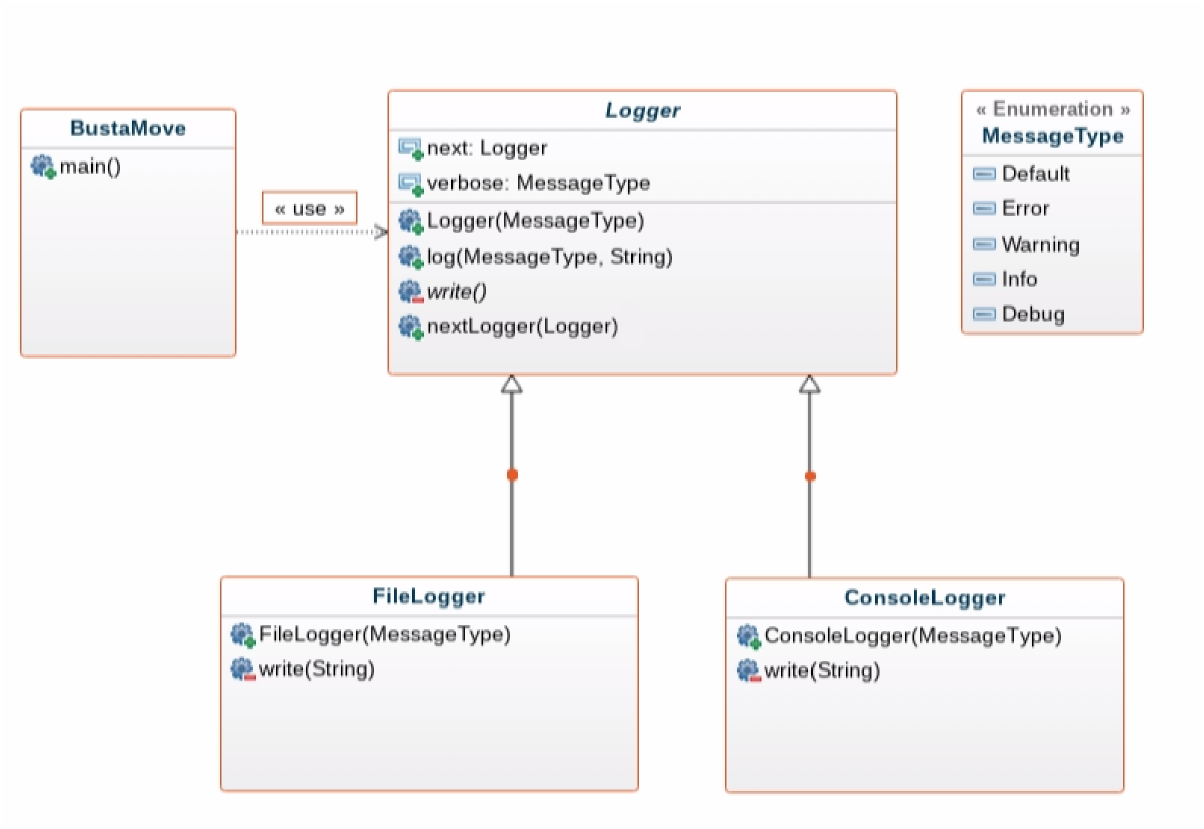
Logger Design Group 66

This document describes the design of a logger using responsibility driven design and UML.

Class Responsibility Collaborator (CRC)

Class	**Responsibility**	**Collaborators**
Logger	1. Handles incoming log messages and format them into the format specified in the requirements. 2. Cycles through loggers that are extended from this Logger class	
FileLogger	1. Extends Logger and adds a new *write* function which will write the message to a file 2. Loads the next *Logger* if present 3. Checks if the verbosity level of the message	Logger
ConsoleLogger	1. Extends Logger and adds a new *write* function which will write the message to the console 2. Loads the next *Logger* if present 3. Checks if the verbosity level of the message	Logger

UML diagram


BustaMove
main()
« use »
Logger
next: Logger
verbose: MessageType
Logger(MessageType)
log(MessageType, String)
write()
nextLogger(Logger)
« Enumeration »
MessageType
Default
Error
Warning
Info
Debug
FileLogger
FileLogger(MessageType)
write(String)
ConsoleLogger
ConsoleLogger(MessageType)
write(String)

Technical Design and Implementation

The created logger uses both a template pattern as a chain of command pattern. The template pattern is used for the abstract Logger class, this class implements the main message parsing of the logger and the output of a logger can be created by extending it. This is done for the FileLogger and ConsoleLogger. Here only the write function is implemented to write to the desired output (file or console).

The chain of command patterns is used to chain the loggers. At the start of the program in the BustaMove class the loggers that are needed (ConsoleLogger and FileLogger) are initialized and chained. The chaining is done by specifying a nextLogger within the log object (ConsoleLogger or FileLogger). When a log message is written the message is written to the first Logger, if it has a nextLogger this one will be loaded and the message will be written to that Logger. When no nextLogger is specified the logging chain is done.

A Logger is passed a verbosity level when created, these are in order of importance: Default, Error, Warning, Info, Debug. When a verbosity level is specified, that one and all the messages with a lesser verbosity level will be outputted by the logger.