Software Systems Document

Team A: Bryson, Eli, Jack, and Abby

4/23/2025

Requirements:

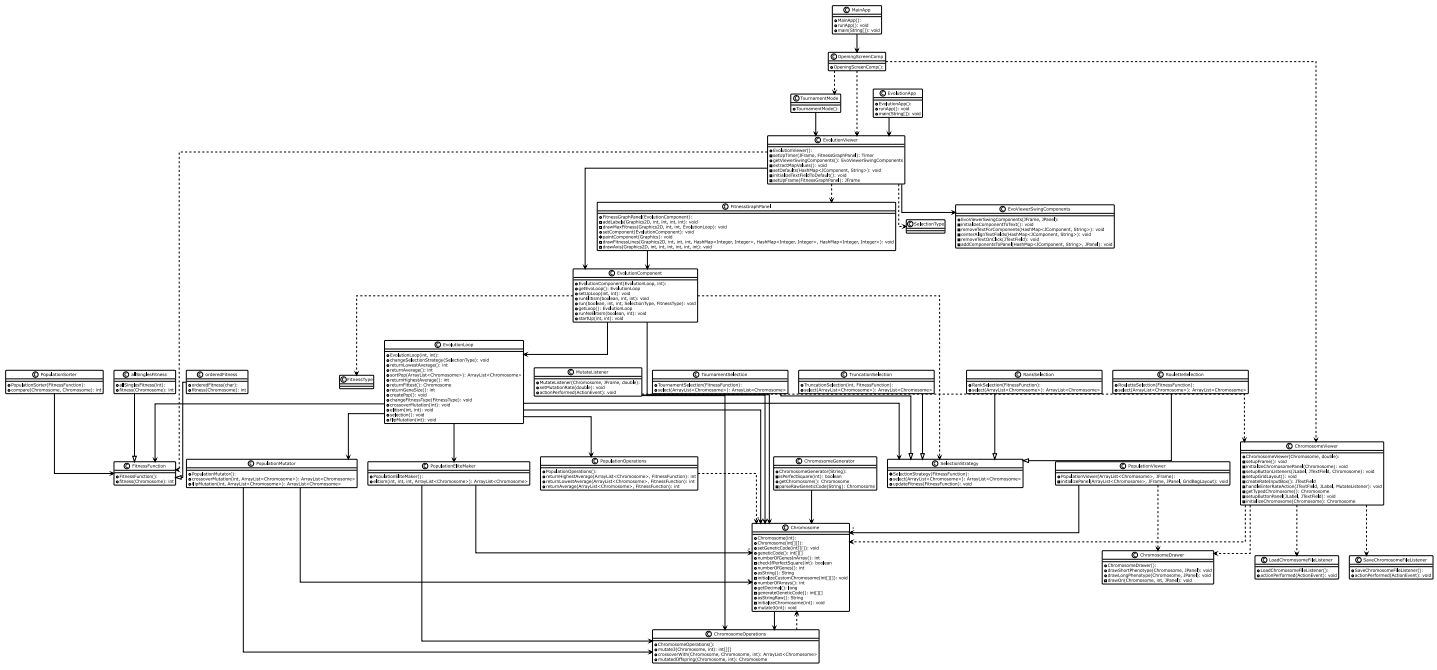
The GARP is a program meant to simulate artificial evolution of chromosome fitness. Thus, its base requirements must allow a use to generate a population of chromosomes (represented as binary arrays), determine each individual's fitness in some way, introduce some form of variation in individuals/generations (mutation/crossover), and have a way to select for the individuals that will be propagated in further generations. This base representation, however, does not allow the user to experiment with artificial evolution to the extent that is desired, so further requirements are as follows:

1. A GUI that allows user control over the program
2. A visual representation of each generation's fitness in relation to the last (a fitness over time graph)
3. Multiple fitness functions that determine how to determine the most fit individual
4. Mutation, elitism, and crossover between generations
5. User input to control hyperparameters for the experiment
6. Individual chromosome viewing and editing
7. Modes to compare different fitness functions
8. The ability to save and load chromosomes from files

Software Architecture:

Given that the refactoring was meant to preserve and improve existing design and functionality, changes were made to improve design where applicable, but large design flaws could not be resolved without fully rewriting/rethinking the project from the start. Thus, the most notable improvements were implementation of two strategy patterns (FitnessFunction and SelectionStrategy), the distribution of large classes into single-purpose classes, and the breaking of inter-class dependencies into encapsulated classes which follow the Hollywood Principle.

GARP UML:



Design Specifications:

As mentioned above, to fulfill our requirements while staying within the scope of the project, design patterns were implemented where applicable to simplify poor existing logic, while not fully rewriting the codebase. To supplement this, individual class logic was heavily changed to promote encapsulation and have each method be single-purpose and function within the needs of its class. Specifically, the most common pattern found was that fields were being called in classes where that logic was not specifically needed this. This was most commonly remedied by moving fields and methods into the appropriate class and having other classes call them, encapsulating those fields and preventing feature envy. More specific examples can be seen through GitHub's version tracking but, as a whole, design was significantly improved to further reflect good code practice and make further improvements easier.