Programming Assignment

Regina Ibragimova

The aim of this Java project is to provide the opportunity for bioinformatics team to work together on a nucleotide multiple sequence alignment. All the users of the bioinformatics team and an optimal alignment are stored in a repository.

The design of repository

Repository is a directory with 3 types of files:

- alignment.txt file, which is a fasta file with an optimal alignment,
- repository.txt file, which is a space separated file with first and second name and experience of each bioinformatician,
- personal fasta files with the alignments of each bioinformatician, named as “first name second name.alignment.txt”.

Class Descriptions

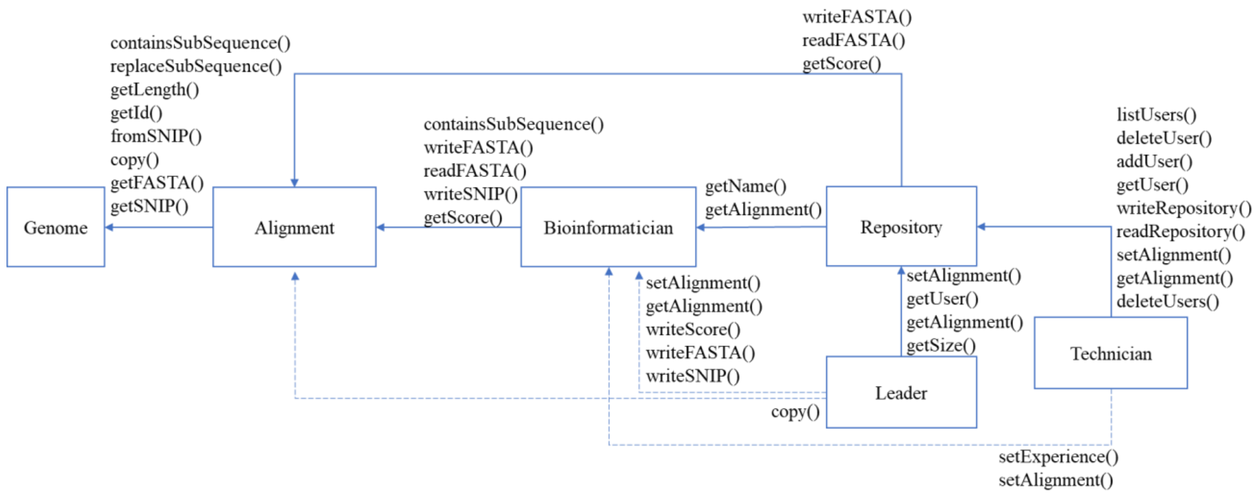
There are 7 classes in the project: Genome, Alignment, Repository, Bioinformatician, Leader, Technician and CLI.

- **Genome**: since there are operations that need to be done in a given genome, a Genome class has been created as a smallest unit that makes up an alignment. It represents a genome sequence with its id, so a Genome class object has ***id*** and ***sequence attributes***. A Genome object can be constructed by 2 ways: from id combined with a sequence and from SNiP(existing genome can restore dots in a given SNiP sequence to produce a new genome). This class has methods to *get id, sequence, length, FASTA, SNIP*, as well as methods to *replace* subsequence, check whether a genome *contains* a subsequence and *copy* a genome.
- **Alignment**: this class represents a list of aligned genomes and has 3 attributes: ***genomes*** (a list of genomes), ***genomeIds*** (mapping between a genome id and index of a genome in genomes attribute for quick access to a genome by id) and ***genomeLength*** (needed for quick access of stored genomes` length). It is possible to construct an empty alignment object. Also, an alignment object can be created either from a list of genomes or by parsing FASTA or SNiP files by using the corresponding *read* methods. As for other methods, this class also has *copy*, which produce a deep copy of alignment and stored genomes, *get size* and *length* to determine the number of genomes and sequence length of genome as it might be useful for genomes iterating, *get score* for an alignment score calculating. An alignment object can also generate FASTA and SNiP file using corresponding *write* methods and manipulate with genomes list by *list, get, add, replace*(all occurrences of a given sequence in a given genome), *delete* methods. For batch processing, methods that check whether genome *contains* a subsequence and *replace* all occurrences of a given sequence in the entire alignment have been created.
- **Repository**: this class represents repository, where all the bioinformaticians and an optimal alignment are stored. It has ***alignment*** and ***users*** attributes, which present an optimal alignment and a list of bioinformaticians respectively. The class has methods to *read* and *write* repository, *list users* (bioinformaticians) and their alignments *scores*, *get* and *set* an optimal alignment, *get* and *delete users*.

- **Bioinformatician**: this class represents bioinformaticians and has ***firstName***, ***lastName***, ***alignment*** (as every bioinformatician has own personal alignment) and ***experience*** attributes. It has methods for an *alignment* to be *set*, to *get* and *set experience*, to *write FASTA*, *SNiP* and *score*, *read FASTA* and *SNiP*, *add* and *replace genome* and check whether a bioinformaticians` alignment *contains* a *subsequence*.
- **Leader**: this class represents team leaders and has a ***repository*** attribute. The methods of this class are aimed to *set*, *promote* an alignment of a bioinformatician, *write* a *score* of an alignment, *write FASTA* and *SNiP*.
- **Technician**: this class represents technical support members and has a ***repository*** attribute. The class has methods to *list*, *add*, *delete users* (bioinformaticians), *set an alignment* and *experience* of the bioinformaticians; to *back up* and *restore* repository data; to *restore user* and *alignment*.
  Since a repository is just a folder with files, clearing the repository data can be done just by deleting the folder, so a method for this task has not been created. However, deleting of a specific bioinformatician can be done by a technician using corresponding method mentioned above.
- **CLI** (Command Line Interface): this class contains the entry point (the main method) with a common line parser, which is made by using argparse4j library.

Class Relationships

The figure below demonstrates the relationship between the classes (solid arrows represent association, dashed arrows represent class methods usage, the text next to the arrows means used methods).



Usage Workflow

In workflow.sh a set of commands with comments for manipulation with repository is shown. It includes technician, bioinformatician and team leader related commands.

Strengths and weaknesses of the project

The **strength** of the project is that project repository based on plane text files, so it is easier to restore information if something goes wrong or repository can be tracked by version control systems. Besides, a list of a project-specific exception classes is implemented for advanced errors handling.

The **weakness** of the project is that there is no safety, no account access restrictions. Moreover, there is a risk of data corruption during parallel execution.

Dependencies: argparse4j and jetbrains.annotations.

Difficulties

It took me some time to understand if inheritance needs to be used in this project. It could be used only for describing team members` common attributes and methods. However, as in the current design there was no need to use authentication and logging, there is no reason to implement different technician and leader accounts, and because of that both technician and leader do not have name and experience attributes unlike bioinformaticians.