PV286 - Panbyte application

Team members
- Michal Badin - 485517
- Dominik Dubovský - 485020
- Andrea Jonásová - 485243

Current progress

The first task we completed was setting up commit signing for each team member according to the provided instructions.
When it comes to the Panbyte application itself, we have already implemented functionality as it is described in the following paragraph.
Our program supports the following commands for converting input:
- -f FORMAT and --from=FORMAT
- --from-options=OPTIONS
- -t FORMAT --to=FORMAT
- --to-options=OPTIONS
- -h --help

We have to finish implementation for the following commands for converting inputs:
- -i FILE --input=FILE
- -o FILE --output=FILE
- -h --help Print help

Our program supports the following formats:
- bytes
- hex
- int – with options for input and output:
    - big
    - little
- bits – with options for input:
    - left
    - right

We have to finish implementation for the following formats of inputs:
- array - with all required options for output

Currently, our program passes provided examples of the supported formats.

We have already set CI GitHub for automated build and test checks. The pipeline runs every time when a new commit is merged/pushed into the master branch. In the foreseeable future, we plan to separate the build and test into two different actions and add some format for the output of the test results.
Tests should cover all functionality of our program. We tried to find all possible edge cases which could possibly cause errors in our application. As it is mentioned below, there are many classes for which we have created unit tests. There are also functional tests for simulating user inputs and program outputs.

Program design

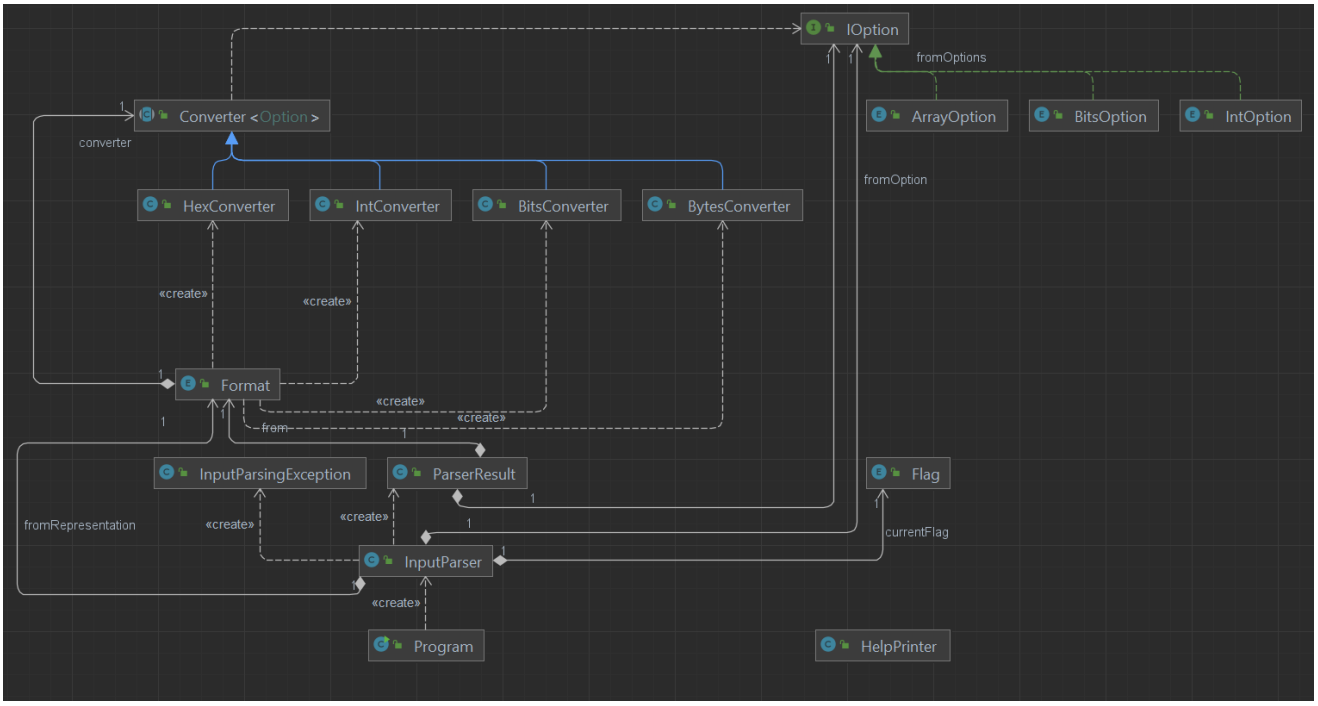
General info:

- Programming language: Java
- JDK: Java 17
- Build tool: Gradle
- Additional library (**we used only for tests**): JUnit

How to build and run the application:

- Ensure that you have set environment variables: JAVA_HOME and PATH to the java executable - version JDK 17 is needed
- From the project directory call the command:
  - *./gradlew build jar* -> it will build and create a new directory called appBuilds and JAR file called panbyte.jar
- From the current directory navigate to the appBuilds directory with command:
  - *cd appBuild*
- From appBuilds you can execute our panbyte application with commands e.g.:
  - *echo 2345 | java -jar ./panbyte.jar -f int -t hex*

Project structure



Our project is divided into two folders - main and tests. The Tests folder contains all tests which we split into 2 categories - tests for input parsing and tests for value converting. Main folder contains all functionality.

For converting from one format to the second one we created a converter for each format. These converters implement a common interface that defines two functions, one for converting input into inner representation (bit string) and the second one which takes this inner representation and converts it into concrete type. Both functions take 2 parameters: Option and String which is processed according to this option. If format has no options, then it is possible to pass into the function any value which implements IOption interface, this is mainly for the future possible extension, but has no impact on functionality. To make sure that into functions are passed just options corresponding to this format we created several enum classes to handle these options. Extended abstract class is then provided with specific enum of type IOption. Each format has its own option enum, if it is necessary. These enums inherit from a common interface that defines that values have to at least contain text representation according to which are values created and descriptions for HelpPrinter.

For parsing of arguments, we created InputParser class which contains 1 public method for parsing. A Whole list of arguments is passed to this method. If this list of arguments contains any invalid values, an exception is thrown. For this purpose, we created a custom exception - InputParsingException. When all arguments in the list are valid, method parse returns an instance of ParserResult class containing all necessary information (from the format, to format, from options, to options, input file, output file, delimiter and info whether help should be printed). Since there are two versions of arguments (long and short versions) that should be supported, to make things easier we created Flag enum. For each possible argument this enum contains a short version of flag, a long version of flag and description.

For help printing we created a class HelpPrinter containing 1 static public method which prints information about all supported functionality. To avoid putting long chunks of text into this class to describe all supported functionality, we used description fields in enums which contain information about each item separately.

Problems during implementation

- From the beginning, we encountered problems with setting up the commit signing. It was something that none of us had used before. The given tutorial was not proper and we had to find another one. There were other steps which had to be executed (especially on Windows) in order to make it work.
- Another small problem was to come up with a way to execute java program as it is stated in the assignment, i.e. echo input | ./panbyte -f int -to hex. At first, we didn't know whether it had to be executable with this exact command or if we had to only replicate the functionality of it.
- One of the challenges was to write a method that parses all the arguments. Since we could not use any external libraries and Java does not have any support for custom arguments, except for when they start with -D (at least as far as I am aware), we had to write our own parser. This was quite time-consuming and it was difficult to cover all the possible arguments because there were 2 types of arguments which have a different structure (-f FORMAT and –from=FORMAT).
- During investigation of possible edge cases for args inputs, we were not 100% sure whether the input is correct or invalid. It was not mentioned in the assignment. So there are specific conditions that have to be met to pass the input check and we are

hoping that they are correct. E.g: duplicity of args (./panbyte.jar –help –help) is invalid input.