Exercise 1

- Create a method for loading plugins to a program
  - (incomplete) method signature
  - **`static List loadPlugins(Class pluginInterface, String... pluginNames);`**
    - pluginInterface – an interface that the plugin class must implement
    - pluginNames – names of the classes to be loaded
  - update the signature so the returning type can be a List, from which instances of the pluginInterface can be directly taken (without casting)

Exercise 1

- correct signature
  - `static <T> List<T> loadPlugins(Class<T> pluginInterface, String... pluginNames);`

Exercise 2

- Create an extensible text processor
  - program reads from the std input and prints out to std. output
  - as an argument, it obtains a file containing a list of action to be performed with the text
  - actions are performed in the given order
  - an action ~ a class implementing TextProcessor interface

```
interface TextProcessor {
  String process(String text);
}
```

  - the file contains full names of classes implementing the interface

File

```
cz.cuni.mff.ajava.testprocessor.ToUpperProcessor
cz.cuni.mff.ajava.testprocessor.JustifyLeft
```

Exercise 3

- Create an extensible shell (i.e. an interactive program performing commands)
  - help
    - builtin command (the only one)
    - prints out a list of all available commands
  - other commands – classes implementing a prescribed interface
    - design a suitable interface
      it has to contain at least methods for
      - name of the command
      - help (for the help command)
      - command execution
  - the list of commands (i.e., classes) is passed in a file
    - as in the previous example

Exercise 3

- a possible interface for commands

```
interface Command {
   String getCommand();
   String getHelp();
   String execute(String... args);
}
```