# Basic Programming Practicum
# Final Exam Project


**Name**
Dicha Zelianivan Arkana

**NIM**
2241720002

**Class**
1i

**Department**
Information Technology

**Study Program**
D4 Informatics Engineering

# Contents

# 1 Preliminary

This document provides the documentation of how to use a CLI app that keep track of Student's violated rules. It includes a detailed steps on how to use each and every menu, a flowchart to understand the code flow, and the code itself.

# 2 Compiling and Running

The app itself is just a single `.java` file so it should be trivial to run it. It can be compiled using the `javac` command and then use the `java` command to run it.

Although, the app is provided in a form of Maven project, so it would be easier to just use an Integrated Development Environment (IDE) and import the project itself. After doing that, simply press `Ctrl + F5` or the play button in the IDE.

For a better experience, the app should be run on a terminal that supports ANSI escape code, such as the new Windows Terminal, because the program uses `\033[H\033[2J` to reset the screen state. It is done to simulate how a page navigation would work inside a GUI app.

# 3 Usage

These are the steps to use each and every part that is available on the app, including the logic behind it.

## 3.1 Top Level Menu

### 3.1.1 Logging In

Upon running the app, there should be a prompt asking the user to log in. Since there is no database integration, the credential is hardcoded. Insert **admin** as the username and **admin123** as the password.

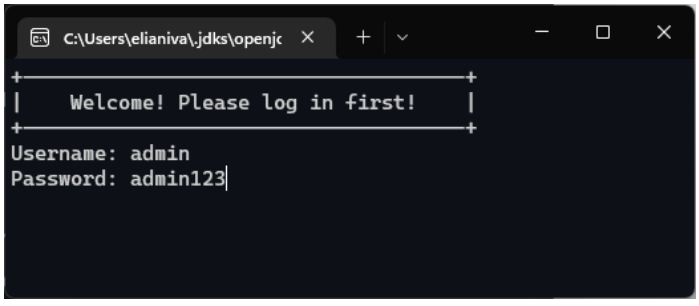


Figure 1: The app prompting a username and password

### 3.1.2 Main Menu

After the user logged in, there should be a main menu with a greeting message. The greeting message will only appear on initial login. It will not appear later on.

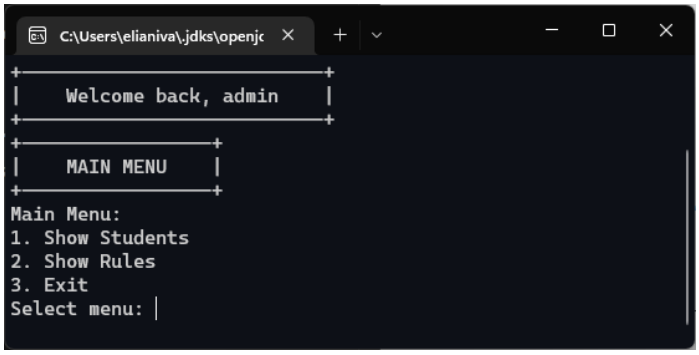


Figure 2: The app showing a main menu

## 3.2 Students Menu

On the main menu, choose the first menu to show a list of actions related to Student operations.

### 3.2.1 Adding a new Student

# 4 Code

```java
import java.util.Scanner;

public class Main {
    static String LINE_PLUS = "+";
    static String LINE_HORIZONTAL = "-";
    static String LINE_VERTICAL = "|";
    static String USERNAME = "admin";
    static String PASSWORD = "admin123";

    // [nim, fullName, classPlacement, violatedRuleIndices, currentPunishment, violationsCo
    static String[][] students = {
            {"1234560001", "Harimurti Suryono", "1A", "", "", "0"},
            {"1234560002", "Carla Andriani", "1B", "", "", "0"},
            {"1234560003", "Hana Astuti", "1C", "", "", "0"},
            {"1234560004", "Rini Padmasari", "1D", "", "", "0"},
            {"1234560005", "Karsana Nababan", "1E", "", "", "0"}
    };

    // [code, description, level]
    static String[][] rules = {
            {"R001", "Communicating in a disrespectful manner, whether written or written t
            {"R002", "Eating, or drinking in the lecture theatre/laboratory/workshop", "4"}
            {"R003", "Students sporting punk-style hair, painted other than black and/or sk
            {"R004", "Violating the rules / regulations that apply in Polinema both in the
            {"R005", "Not maintaining cleanliness in all areas of Polinema", "3"},
            {"R006", "Smoking outside the smoking area", "3"},
            {"R007", "Playing cards, online games in the campus area", "3"},
            {"R008", "Damaging facilities and infrastructure in the Polinema area", "2"},
            {"R009", "Accessing pornographic material in class or campus areas", "2"},
            {"R010", "Conducting practical political activities on campus", "2"},
            {"R012", "Using psychotropic substances and / or other addictive substances oth
    };

    static String[] punishments = {
            "Oral reprimand accompanied by a statement not to repeat the act, affixed with
            "A written reprimand accompanied by a statement not to repeat the act, affixed
            "a. Make a statement not to repeat the act, affixed with stamp duty, signed by
                    "\t\t  b. Perform special tasks, such as being responsible for repairin
            "a. Compensation for damages or replacement of similar objects/goods and/or\n"
                    "\t\t  b. Performing social service duties for a certain period of time
                    "\t\t  c. Given a grade of D in the relevant course when committing the
            "a. Disabled (Academic/Terminal Leave) for two semesters and/or\n" +
                    "\t\t  b. Dismissed as a student."
```

```
44      };
45
46      static Scanner scanner = new Scanner(System.in);
47
48      public static void main(String[] args) {
49          renderTitle("Welcome! Please log in first!");
50          login();
51          clearScreen();
52          renderTitle("Welcome back, " + USERNAME);
53          while (true) {
54              renderTitle("MAIN MENU");
55              int chosenMenu = pickMenu("Main Menu: ", new String[]{
56                      "Show Students",
57                      "Show Rules",
58                      "Exit"
59              });
60              boolean shouldContinue = routeMainMenu(chosenMenu);
61              if (shouldContinue) continue;
62              break;
63          }
64      }
65
66      static void login() {
67          while (true) {
68              String username = getNonEmptyString("Username: ", "Username can't be empty!");
69              String password = getNonEmptyString("Password: ", "Password can't be empty!");
70              if (username.equals(USERNAME) && password.equals(PASSWORD)) {
71                  break;
72              }
73              clearScreen();
74              System.out.println("Incorrect username and password!");
75          }
76      }
77
78      static boolean routeMainMenu(int chosenMenu) {
79          // this is a sad version of switch case...
80          // could've made it pretty, but it's just not allowed
81          return switch (chosenMenu) {
82              case 1 -> handleShowStudents();
83              case 2 -> handleShowRules();
84              case 3 -> exit();
85              default -> handleInvalidMenu();
86          };
87      }
88
```

```
89      static boolean routeStudentMenu(int chosenMenu) {
90          // this is a sad version of switch case...
91          // could've made it pretty, but it's just not allowed
92          return switch (chosenMenu) {
93              case 1 -> handleShowStudentDetail();
94              case 2 -> handleAddViolatedRuleToStudent();
95              case 3 -> handleAddStudent();
96              case 4 -> handleEditStudent();
97              case 5 -> handleRemoveStudent();
98              case 6 -> handleResetStudent();
99              case 7 -> back();
100             case 8 -> exit();
101             default -> handleInvalidMenu();
102         };
103     }
104
105     static boolean routeRuleMenu(int chosenMenu) {
106         // this is a sad version of switch case...
107         // could've made it pretty, but it's just not allowed
108         return switch (chosenMenu) {
109             case 1 -> handleAddRule();
110             case 2 -> handleEditRule();
111             case 3 -> handleRemoveRule();
112             case 4 -> back();
113             case 5 -> exit();
114             default -> handleInvalidMenu();
115         };
116     }
117
118     static boolean exit() {
119         clearScreen();
120         renderTitle("Exiting...");
121         System.exit(0);
122         return false;
123     }
124
125     static boolean back() {
126         clearScreen();
127         return false;
128     }
129
130     static boolean handleInvalidMenu() {
131         System.out.println("Invalid menu!");
132         clearScreen();
133         return true;
```

```
134     }
135
136     static boolean handleShowStudents() {
137         clearScreen();
138         while (true) {
139             renderStudentsTable("Showing All Students", students);
140             int chosenMenu = pickMenu("Students Table Menu: ", new String[]{
141                     "Show Student Detail",
142                     "Add Student Violated Rule",
143                     "Add Student",
144                     "Edit Student",
145                     "Remove Student",
146                     "Reset Student",
147                     "Back",
148                     "Exit",
149             });
150             boolean shouldContinue = routeStudentMenu(chosenMenu);
151             if (shouldContinue) continue;
152             break;
153         }
154         return true;
155     }
156
157     static boolean handleShowStudentDetail() {
158         clearScreen();
159         String nim;
160
161         while (true) {
162             renderTitle("Select Student");
163             renderStudentsTable("Showing All Students", students);
164             nim = getNonEmptyString("NIM: ", "NIM can't be empty!");
165
166             if (has(students, nim, 0)) break;
167
168             clearScreen();
169             System.out.println("Student with the NIM of " + nim + " doesn't exist!");
170         }
171
172         clearScreen();
173         renderTitle("Showing Details for Student " + nim);
174
175         int studentIndex = -1;
176         for (int i = 0; i < students.length; i++) {
177             if (students[i][0].equals(nim)) {
178                 studentIndex = i;
```

```
179                 break;
180             }
181         }
182         if (studentIndex == -1) {
183             clearScreen();
184             System.out.println("Failed to find the student with a nim of " + nim);
185             return true;
186         }
187 
188         String[] student = students[studentIndex];
189         System.out.println("NIM\t\t: " + student[0]);
190         System.out.println("Name\t\t: " + student[1]);
191         System.out.println("Class\t\t: " + student[2]);
192         System.out.println("Punishment\t: " + (student[4].isEmpty() ? "Student has no punis
193 
194         if (student[3].length() > 0) {
195             renderRulesTable("Rules that have been violated", filterRulesByIndices(rules, s
196         }
197 
198         getString("Press enter to continue...");
199 
200         clearScreen();
201         return true;
202     }
203 
204     static boolean handleAddViolatedRuleToStudent() {
205         clearScreen();
206         String nim, code;
207 
208         int studentIndex = -1;
209         while (true) {
210             renderTitle("Add Violated Rule to Student");
211             renderStudentsTable("Showing All Students", students);
212             nim = getNonEmptyStringWithLimit("Student's NIM: ", "NIM can't be empty!", 10,
213 
214             if (!has(students, nim, 0)) {
215                 clearScreen();
216                 System.out.println("Student with the NIM of " + nim + " doesn't exist!");
217                 continue;
218             }
219 
220             for (int i = 0; i < students.length; i++) {
221                 if (students[i][0].equals(nim)) {
222                     studentIndex = i;
223                     break;
```

```
224                 }
225             }
226 
227             if (students[studentIndex][3].length() == 3) {
228                 System.out.println("This student has maxed out the violated rule limit. Ple
229                 getString("Press enter to continue...");
230                 return true;
231             }
232 
233             break;
234         }
235 
236         clearScreen();
237 
238         int ruleIndex = -1;
239         while (true) {
240             renderTitle("Add Violated Rule to Student");
241             renderRulesTable("Showing All Rules", rules);
242             code = getNonEmptyStringWithLimit("Rule's Code: ", "Code can't be empty!", 4, 4
243 
244             if (!has(rules, code, 0)) {
245                 clearScreen();
246                 System.out.println("Rule with the code of " + code + " doesn't exist!");
247                 continue;
248             }
249 
250             for (int i = 0; i < rules.length; i++) {
251                 if (rules[i][0].equals(code)) {
252                     ruleIndex = i;
253                     break;
254                 }
255             }
256 
257             break;
258         }
259 
260         String[] currentStudent = students[studentIndex];
261 
262         boolean isUpgraded = shouldUpgrade(currentStudent, rules[ruleIndex]);
263         String ruleIndices = currentStudent[3];
264         boolean changedToDifferentLevel = currentStudent[3].length() > 0 && !toString(ruleI
265         if (isUpgraded && changedToDifferentLevel) {
266             currentStudent[3] = toString(ruleIndex);
267         } else {
268             currentStudent[3] += toString(ruleIndex);
```

```
269         }
270         currentStudent[4] = resolvePunishmentIndex(currentStudent[3], isUpgraded);
271         currentStudent[5] = incrementString(currentStudent[5], Integer.MAX_VALUE);
272 
273         clearScreen();
274         System.out.println("Rule have been added to the student successfully");
275 
276         return true;
277     }
278 
279     static boolean handleAddStudent() {
280         clearScreen();
281         String nim, fullName, classPlacement;
282 
283         while (true) {
284             renderTitle("Add New Student");
285             nim = getNonEmptyStringWithLimit("NIM: ", "NIM can't be empty!", 10, 10, false)
286             fullName = getNonEmptyStringWithLimit("Full Name: ", "Full Name can't be empty!
287             classPlacement = getNonEmptyStringWithLimit("Class: ", "Class can't be empty!",
288 
289             if (!has(students, nim, 0)) break;
290 
291             clearScreen();
292             System.out.println("Student with the NIM of " + nim + " already exists!");
293         }
294 
295         String[][] newStudents = new String[students.length + 1][2];
296         for (int i = 0; i < students.length; i++) {
297             newStudents[i] = students[i];
298         }
299         newStudents[students.length - 1] = new String[]{nim, fullName, classPlacement, toSt
300         students = newStudents;
301 
302         clearScreen();
303         System.out.println("Students have been succesfully added!");
304         return true;
305     }
306 
307     static boolean handleEditStudent() {
308         clearScreen();
309         String oldNim, nim, fullName, classPlacement;
310         int studentIndex = -1;
311 
312         while (true) {
313             renderTitle("Edit Student");
```

```
314             renderStudentsTable("Showing Students to Edit", students);
315             oldNim = getNonEmptyStringWithLimit("NIM: ", "NIM can't be empty!", 10, 10, fal
316 
317             if (has(students, oldNim, 0)) break;
318 
319             clearScreen();
320             System.out.println("The student with the NIM of " + oldNim + " doesn't exists")
321         }
322 
323         for (int i = 0; i < students.length; i++) {
324             if (students[i][0].equals(oldNim)) {
325                 studentIndex = i;
326                 break;
327             }
328         }
329 
330         if (studentIndex == -1) {
331             clearScreen();
332             System.out.println("Failed to find student to edit");
333             return true;
334         }
335 
336         String[] student = students[studentIndex];
337 
338         clearScreen();
339         renderTitle("New Student Data");
340         nim = getNonEmptyStringWithLimit("NIM (old: " + student[0] + "): ", "NIM can't be e
341         fullName = getNonEmptyStringWithLimit("Full Name (old: " + student[1] + "): ", "Ful
342         classPlacement = getNonEmptyStringWithLimit("Class (old: " + student[2] + "): ", "C
343 
344         students[studentIndex][0] = nim.isEmpty() ? student[0] : nim;
345         students[studentIndex][1] = fullName.isEmpty() ? student[1] : fullName;
346         students[studentIndex][2] = classPlacement.isEmpty() ? student[2] : classPlacement;
347 
348         clearScreen();
349         System.out.println("Students have been succesfully added!");
350         return true;
351     }
352 
353     static boolean handleRemoveStudent() {
354         clearScreen();
355         String nim;
356 
357         while (true) {
358             renderTitle("Remove Student");
```

```
359             renderStudentsTable("Showing Students to Remove", students);
360             nim = getNonEmptyString("NIM: ", "NIM can't be empty!");
361 
362             if (has(students, nim, 0)) break;
363 
364             clearScreen();
365             System.out.println("Student with the NIM of " + nim + " doesn't exist!");
366         }
367 
368         String[][] filteredStudents = new String[students.length - 1][4];
369         int count = 0;
370         for (int i = 0; i < students.length; i++) {
371             if (students[i][0].equals(nim)) continue;
372             filteredStudents[count] = students[i];
373             count++;
374         }
375         students = filteredStudents;
376 
377         clearScreen();
378         System.out.println("Students have been successfully removed!");
379         return true;
380     }
381 
382     static boolean handleResetStudent() {
383         clearScreen();
384         String nim;
385 
386 
387         int studentIndex = -1;
388         while (true) {
389             renderTitle("Reset Student");
390             renderStudentsTable("Showing Students to Reset", students);
391             nim = getNonEmptyStringWithLimit("Student's NIM: ", "NIM can't be empty!", 10,
392 
393             if (!has(students, nim, 0)) {
394                 clearScreen();
395                 System.out.println("Student with the NIM of " + nim + " doesn't exist!");
396                 continue;
397             }
398 
399             for (int i = 0; i < students.length; i++) {
400                 if (students[i][0].equals(nim)) {
401                     studentIndex = i;
402                     break;
403                 }
```

```
404             }
405
406             break;
407         }
408
409         students[studentIndex][3] = "";
410         students[studentIndex][4] = "";
411
412         clearScreen();
413         System.out.println("Students have been successfully reset!");
414         return true;
415     }
416
417     static boolean handleShowRules() {
418         clearScreen();
419         while (true) {
420             renderRulesTable("Showing All Rules", rules);
421             int chosenMenu = pickMenu("Rules Table Menu: ", new String[]{
422                     "Add Rule",
423                     "Edit Rule",
424                     "Remove Rule",
425                     "Back",
426                     "Exit",
427             });
428             boolean shouldContinue = routeRuleMenu(chosenMenu);
429             if (shouldContinue) continue;
430             break;
431         }
432         return true;
433     }
434
435     static boolean handleAddRule() {
436         clearScreen();
437         String code, description;
438         int level;
439
440         while (true) {
441             renderTitle("Add New Rule");
442             code = getNonEmptyStringWithLimit("Code: ", "Code can't be empty!", 4, 4, false
443             description = getNonEmptyStringWithLimit("Description: ", "Description can't be
444             level = getIntegerWithRange("Level: ", 1, 5, false);
445
446             if (!has(rules, code, 0)) break;
447
448             clearScreen();
```

```
449             System.out.println("Rule with the code of " + code + " already exists!");
450         }
451
452         String[][] newRules = new String[rules.length + 1][2];
453         for (int i = 0; i < rules.length; i++) {
454             newRules[i] = rules[i];
455         }
456         newRules[rules.length - 1] = new String[]{code, description, toString(level)};
457         rules = newRules;
458
459         clearScreen();
460         System.out.println("New rule have been successfully added!");
461         return true;
462     }
463
464     static boolean handleEditRule() {
465         clearScreen();
466         String oldCode, code, description;
467         int level;
468         int ruleIndex = -1;
469
470         while (true) {
471             renderTitle("Edit Rule");
472             renderRulesTable("Showing Rules to Edit", rules);
473             oldCode = getNonEmptyStringWithLimit("Code: ", "Code can't be empty!", 4, 4, fa
474
475             if (has(rules, oldCode, 0)) break;
476
477             clearScreen();
478             System.out.println("The rule with the code of " + oldCode + " doesn't exists");
479         }
480
481         for (int i = 0; i < rules.length; i++) {
482             if (rules[i][0].equals(oldCode)) {
483                 ruleIndex = i;
484                 break;
485             }
486         }
487
488         if (ruleIndex == -1) {
489             clearScreen();
490             System.out.println("Failed to find rule to edit");
491             return true;
492         }
493
```

```
494         String[] rule = rules[ruleIndex];
495 
496         clearScreen();
497         renderTitle("New Rule Detail");
498         while (true) {
499             code = getNonEmptyStringWithLimit("Code (old: " + rule[0] + "): ", "Code can't 
500             if (!has(rules, code, 0)) break;
501             System.out.println("There's a rule with the same code already! Please try anoth
502         }
503         description = getNonEmptyStringWithLimit("Description (old: " + rule[1] + "): ", "D
504         level = getIntegerWithRange("Level (old: " + rule[2] + "): ", 1, 5, true);
505 
506         rules[ruleIndex][0] = code.isEmpty() ? rule[0] : code;
507         rules[ruleIndex][1] = description.isEmpty() ? rule[1] : description;
508         rules[ruleIndex][2] = level == -1 ? rule[2] : String.format("%s", level);
509 
510         clearScreen();
511         System.out.println("Rule have been succesfully added!");
512         return true;
513     }
514 
515     static boolean handleRemoveRule() {
516         clearScreen();
517         String code;
518 
519         while (true) {
520             renderTitle("Remove Rule");
521             renderRulesTable("Showing Rules to Remove", rules);
522             code = getNonEmptyString("Code: ", "Code can't be empty!");
523 
524             if (has(rules, code, 0)) break;
525 
526             clearScreen();
527             System.out.println("Rule with the code of " + code + " doesn't exist!");
528         }
529 
530         String[][] filteredRules = new String[rules.length - 1][3];
531         int count = 0;
532         for (int i = 0; i < rules.length; i++) {
533             if (rules[i][0].equals(code)) continue;
534             filteredRules[count] = rules[i];
535             count++;
536         }
537         rules = filteredRules;
538 
```

```
539             clearScreen();
540             System.out.println("Rule have been successfully removed!");
541             return true;
542         }
543 
544         static void renderTitle(String title) {
545             int paddingSize = 4;
546             int titleLength = title.length();
547 
548             String horizontalBorder = LINE_PLUS + LINE_HORIZONTAL.repeat(titleLength + paddingS
549 
550             System.out.println(horizontalBorder);
551             System.out.println(LINE_VERTICAL + " ".repeat(paddingSize) + title + " ".repeat(pad
552             System.out.println(horizontalBorder);
553         }
554 
555         static void renderStudentsTable(String title, String[][] students) {
556             renderTitle(title);
557             final String tableLine = String.format(
558                     "%s%s%s%s%s%s%s%s%s%s%s",
559                     LINE_PLUS, LINE_HORIZONTAL.repeat(6), LINE_PLUS, LINE_HORIZONTAL.repeat(14)
560                     LINE_PLUS, LINE_HORIZONTAL.repeat(24), LINE_PLUS, LINE_HORIZONTAL.repeat(9)
561                     LINE_PLUS, LINE_HORIZONTAL.repeat(14), LINE_PLUS
562             );
563             System.out.println(tableLine);
564             System.out.printf("%s  No. %s     NIM      %s       Full Name        %s  Class  %s
565             System.out.println(tableLine);
566             for (int i = 0; i < students.length; i++) {
567                 String[] student = students[i];
568                 System.out.printf(
569                         "%s  %-2s  %s  %-10s  %s  %-20s  %s  %5s  %s   %8s   %s\n",
570                         LINE_VERTICAL, (i + 1) + ".", LINE_VERTICAL, student[0], LINE_VERTICAL,
571                         LINE_VERTICAL, student[2], LINE_VERTICAL, student[5], LINE_VERTICAL);
572             }
573             System.out.println(tableLine);
574         }
575 
576         static void renderRulesTable(String title, String[][] rules) {
577             renderTitle(title);
578             final String tableLine = String.format(
579                     "%s%s%s%s%s%s%s%s%s%s",
580                     LINE_PLUS, LINE_HORIZONTAL.repeat(7), LINE_PLUS, LINE_HORIZONTAL.repeat(10)
581                     LINE_PLUS, LINE_HORIZONTAL.repeat(124), LINE_PLUS, LINE_HORIZONTAL.repeat(9
582             System.out.println(tableLine);
583             System.out.printf(
```

```
584                 "%s  No.  %s    ID    %s  %sDescription%s   %s  Level  %s\n",
585                 LINE_VERTICAL, LINE_VERTICAL, LINE_VERTICAL, " ".repeat(54), " ".repeat(54)
586         System.out.println(tableLine);
587         for (int i = 0; i < rules.length; i++) {
588             String[] rule = rules[i];
589             System.out.printf(
590                     "%s  %3s  %s   %-5s  %s  %-120s  %s  %5s  %s\n",
591                     LINE_VERTICAL, (i + 1) + ".", LINE_VERTICAL, rule[0], LINE_VERTICAL, ru
592                     LINE_VERTICAL, rule[2], LINE_VERTICAL);
593         }
594         System.out.println(tableLine);
595     }
596
597     static int pickMenu(String menuTitle, String[] menus) {
598         System.out.println(menuTitle);
599         for (int i = 0; i < menus.length; i++) {
600             System.out.printf("%d. %s\n", i + 1, menus[i]);
601         }
602         return getIntegerWithRange("Select menu: ", 1, menus.length, false);
603     }
604
605     static String getString(String prompt) {
606         System.out.print(prompt);
607         String userInput = scanner.nextLine().trim();
608         return userInput;
609     }
610
611     static String getNonEmptyString(String prompt, String warning) {
612         while (true) {
613             System.out.print(prompt);
614             String userInput = scanner.nextLine().trim();
615             if (!userInput.isEmpty()) {
616                 return userInput;
617             }
618             System.out.println(warning);
619         }
620     }
621
622     static String getNonEmptyStringWithLimit(String prompt, String warning, int min, int ma
623         while (true) {
624             String userInput = allowEmpty ? getString(prompt) : getNonEmptyString(prompt, w
625             if (allowEmpty && userInput.isEmpty()) return userInput;
626             if (userInput.length() >= min && userInput.length() <= max) return userInput;
627             System.out.println("The input can't be shorter than " + min + " or longer than
628         }
```

```
629     }
630
631     static int getIntegerWithRange(String prompt, int min, int max, boolean allowEmpty) {
632         while (true) {
633             System.out.print(prompt);
634             String userInputStr = scanner.nextLine();
635             if (userInputStr.isEmpty() && allowEmpty) return -1;
636
637             int userInput = Integer.parseInt(userInputStr);
638             if (userInput >= min && userInput <= max) return userInput;
639
640             System.out.println("The input can't be lower than " + min + " or greater than "
641         }
642     }
643
644     static void clearScreen() {
645         System.out.print("\033[H\033[2J");
646         System.out.flush();
647     }
648
649     static boolean has(String[][] items, String needle, int fieldIndex) {
650         for (String[] item : items) {
651             if (item[fieldIndex].equals(needle)) return true;
652         }
653         return false;
654     }
655
656     static String toString(int number) {
657         return String.format("%d", number);
658     }
659
660     static String toString(char character) {
661         return String.format("%c", character);
662     }
663
664     static boolean shouldUpgrade(String[] student, String[] nextRule) {
665         String ruleIndices = student[3];
666         if (ruleIndices.isEmpty()) return false;
667
668         boolean isLevelOne = nextRule[2].equals("1");
669         int currentRuleIndex = Integer.parseInt(toString(ruleIndices.charAt(ruleIndices.len
670         boolean isHigher = Integer.parseInt(rules[currentRuleIndex][2]) > Integer.parseInt(
671
672         if (isLevelOne || isHigher) return true;
673
```

```
674         String previous = "";
675         for (int i = 0; i < ruleIndices.length(); i++) {
676             String currentIndex = toString(ruleIndices.charAt(i));
677             if (previous.isEmpty()) {
678                 previous = currentIndex;
679             }
680 
681             if (!previous.equals(currentIndex)) return true;
682         }
683 
684         return previous.length() == 2;
685     }
686 
687     static String incrementString(String previous, int limit) {
688         int prev = Integer.parseInt(previous);
689         int now = prev + 1;
690         return now < limit ? toString(now) : previous;
691     }
692 
693     static String resolvePunishmentIndex(String currentLevel, boolean isUpgraded) {
694         String lastRuleIndexStr = toString(currentLevel.charAt(currentLevel.length() - 1));
695         int lastRuleIndex = Integer.parseInt(lastRuleIndexStr);
696         int lastLevel = Integer.parseInt(rules[lastRuleIndex][2]);
697 
698         if (!isUpgraded) return toString(punishments.length - lastLevel);
699 
700         int level = Integer.parseInt(incrementString(toString(lastLevel), 5));
701         return toString(punishments.length - level + 1);
702     }
703 
704     static String resolvePunishment(String currentLevel) {
705         int currentIndex = Integer.parseInt(currentLevel);
706         return punishments[currentIndex];
707     }
708 
709     static String[][] filterRulesByIndices(String[][] rules, String indices) {
710         String[][] filteredRules = new String[indices.length()][3];
711         for (int i = 0; i < indices.length(); i++) {
712             int index = Integer.parseInt(String.format("%c", indices.charAt(i)));
713             filteredRules[i] = rules[index];
714         }
715         return filteredRules;
716     }
717 }
```