

```
1 package control;
2
3 import adt.CircularDoublyLinkedList;
4 import adt.ListInterface;
5 import adt.exampleAdt.*;
6 import boundary.TutorManagementUI;
7 import dao.*;
8 import entity.Tutor;
9 import java.time.*;
10 import java.util.Comparator;
11 import java.util.Iterator;
12 import utility.*;
13
14 /**
15  *
16  * @author Yip Zi Yan
17  */
18 public class TutorManagement {
19
20     private final String fileName = "tutor.dat";
21     DAO tutorDAO = new DAO(fileName);
22     TutorManagementUI tutorUI = new TutorManagementUI();
23     ListInterface<Tutor> tutorList = new CircularDoublyLinkedList<>();
24
25     // seeder(generate fake data)
26     // TutorSeeder seeder = new TutorSeeder();
27     public TutorManagement() {
28         // tutorDAO.saveToFile(seeder.getTutorList());
29         tutorList = tutorDAO.retrieveFromFile();
30         Tutor.setTotalTutor(tutorList.getNumberOfEntries());
31     }
32
33     public void startUI() {
34
35         int choice;
36         do {
37             choice = tutorUI.getMenuChoice();
38             switch (choice) {
39                 case 1:
40                     displayTutorList();
41                     break;
```

```
42         case 2:
43             addNewTutor();
44             break;
45         case 3:
46             searchTutorUI();
47             break;
48         case 4:
49             updateTutorUI();
50             break;
51         case 5:
52             filterTutorUI();
53             break;
54         case 6:
55             generateReportUI();
56             break;
57         case 7:
58             return;
59         case 0:
60             MessageUI.displayExitMessage();
61             System.exit(0);
62     }
63     } while (choice != 0);
64
65 }
66
67 public static void main(String[] args) {
68     TutorManagement teachingControl = new TutorManagement();
69     teachingControl.startUI();
70 }
71
72 private void addNewTutor() {
73     if (!tutorUI.addTutorMenu()) {
74         return;
75     }
76     int currentId;
77
78     try {
79         currentId = Integer.parseInt(tutorList.getLastEntry()
80             .getTutorId().substring(1)) + 1;
81     } catch (Exception e) {
82         System.out.println("Something went wrong.");
```

```
83         GeneralUtil.systemPause();
84         return;
85     }
86
87     do {
88         Tutor newTutor = tutorUI.addTutor(currentId);
89
90         if (newTutor != null) {
91             tutorList.add(newTutor);
92             tutorDAO.saveToFile(tutorList);
93         }
94
95     } while (tutorUI.contAction("Anymore To Add? [Y|N] > "));
96
97 }
98
99 private void displayTutorList() {
100     tutorUI.displayAllTutor("Tutor List", getAllTutor(),
101         true, 78);
102 }
103
104 private String getAllTutor() {
105     String outputStr = "";
106     int number = 0;
107
108     Iterator<Tutor> it = tutorList.getIterator();
109     while (it.hasNext()) {
110         outputStr += String.format("%2d.  ", ++number)
111             + it.next() + "\n";
112     }
113     return outputStr;
114 }
115
116
117 private void searchTutorUI() {
118     int choice;
119     do {
120         choice = tutorUI.findTutorMenu();
121         switch (choice) {
122             case 1:
123                 searchTutorBy("name",
```

```
124         "Search Tutor Name > ");
125         break;
126     case 2:
127         searchTutorBy("email",
128             "Search Tutor Email > ");
129         break;
130     case 3:
131         searchTutorBy("id",
132             "Search Tutor ID > ");
133         break;
134     }
135     } while (choice != 0);
136 }
137
138 private void searchTutorBy(String attribute, String queryQuestion) {
139     String query = tutorUI.getTutorQuery(queryQuestion, "find")
140         .toLowerCase();
141     String outputStr = "";
142     Iterator<Tutor> it = tutorList.getIterator();
143     int number = 0;
144
145     while (it.hasNext()) {
146         Tutor matchTutor = it.next();
147         switch (attribute) {
148             case "name":
149                 if
150 (matchTutor.getTutorName().toLowerCase().startsWith(query)) {
151                     outputStr += String.format("%2d.  ", ++number)
152                         + matchTutor + "\n";
153                 }
154                 break;
155             case "email":
156                 if
157 (matchTutor.getEmail().toLowerCase().contains(query)) {
158                     outputStr += String.format("%2d.  ", ++number)
159                         + matchTutor + "\n";
160                 }
161                 break;
162             case "id":
163                 if
164 (matchTutor.getTutorId().toLowerCase().equals(query)) {
```

```
162             outputStr += String.format("%2d.  ", ++number)
163                 + matchTutor + "\n";
164         }
165         break;
166     }
167
168     }
169
170     tutorUI.displayFindResult(outputStr);
171
172 }
173
174 private void updateTutorUI() {
175     int choice;
176     do {
177         tutorUI.displayAllTutor("Tutor List", getAllTutor(),
178             false, 78);
179         choice = tutorUI.updateTutorMenu();
180         switch (choice) {
181             case 1:
182                 updateTutor("num", "Enter No. > ");
183                 break;
184             case 2:
185                 updateTutor("id", "Enter Tutor ID > ");
186                 break;
187
188         }
189     } while (choice != 0);
190 }
191
192 private void updateTutor(String searchBy, String queryQuestion) {
193     String query = tutorUI.getTutorQuery(queryQuestion, "update")
194         .toLowerCase();
195     Tutor updateTutor = null;
196     Iterator<Tutor> it = tutorList.getIterator();
197
198     switch (searchBy) {
199         case "id":
200             while (it.hasNext()) {
201                 Tutor matchTutor = it.next();
202
```

```
203             if
(matchTutor.getTutorId().toLowerCase().equals(query)) {
204                 updateTutor = matchTutor;
205             }
206
207         }
208         break;
209     case "num":
210         try {
211             updateTutor = tutorList
212                 .getEntry(Integer.parseInt(query) - 1);
213         } catch (Exception e) {
214             }
215         break;
216     }
217
218     updateSelectedField(updateTutor);
219
220 }
221
222 private void updateSelectedField(Tutor updateTutor) {
223     int choice;
224     boolean isRemove = false;
225     do {
226         tutorUI.displaySelectedTutor(updateTutor);
227         if (updateTutor == null) {
228             return;
229         }
230
231         choice = tutorUI.selectAttributeToUpdate(updateTutor);
232         switch (choice) {
233             case 1:
234                 updateTutorName(updateTutor);
235                 break;
236             case 2:
237                 updateTutorGender(updateTutor);
238                 break;
239             case 3:
240                 updateTutorStatus(updateTutor);
241                 break;
242             case 4:
```

```
243         updateSalary(updateTutor);
244         break;
245     case 5:
246         isRemove = removeTutor(updateTutor);
247         break;
248     }
249     if (isRemove) {
250         tutorUI.removedClosure();
251     }
252     } while (choice != 0 && !isRemove);
253 }
254
255 private void updateTutorName(Tutor updateTutor) {
256     String newName = tutorUI.updateTutorName();
257     if (newName == null) {
258         return;
259     }
260
261     updateTutor.setTutorName(newName);
262     tutorDAO.saveToFile(tutorList);
263 }
264
265 private void updateTutorGender(Tutor updateTutor) {
266     boolean confirmToChange = tutorUI
267         .updateTutorGender(updateTutor.isFemale());
268     if (!confirmToChange) {
269         return;
270     }
271
272     if (updateTutor.isFemale()) {
273         updateTutor.setGender('M');
274     } else {
275         updateTutor.setGender('F');
276     }
277     tutorDAO.saveToFile(tutorList);
278
279 }
280
281 private void updateSalary(Tutor updateTutor) {
282     double newSalary = tutorUI.updateTutorSalary();
283     if (newSalary == -1) {
```

```
284         return;
285     }
286
287     updateTutor.setSalary(newSalary);
288     tutorDAO.saveToFile(tutorList);
289
290 }
291
292 private void updateTutorStatus(Tutor updateTutor) {
293     String newStatus = tutorUI
294         .updateTutorStatus(updateTutor.getStatus());
295     if (newStatus == null) {
296         return;
297     }
298
299     updateTutor.setStatus(newStatus);
300     tutorDAO.saveToFile(tutorList);
301
302 }
303
304 private boolean removeTutor(Tutor updateTutor) {
305     boolean isDeleted = false;
306
307     if (tutorUI.removeTutor()) {
308         isDeleted = tutorList.remove(updateTutor);
309         tutorDAO.saveToFile(tutorList);
310     }
311     return isDeleted;
312 }
313
314 private void filterTutorUI() {
315     int choice;
316     StackInterface<String> filter = new ArrayStack<>();
317
318     do {
319
320         choice = tutorUI.filterTutorMenu(filter.toString());
321         switch (choice) {
322             case 1:
323                 filter.push("FT");
324                 break;
```



```
325         case 2:
326             filter.push("PT");
327             break;
328         case 3:
329             filter.push("RS");
330             break;
331         case 4:
332             filter.push("RT");
333             break;
334         case 5:
335             filter.pop();
336             break;
337         case 6:
338             filterTutor(filter);
339             break;
340     }
341
342     } while (choice != 0);
343 }
344
345 private void filterTutor(StackInterface<String> selectedFilter) {
346     adt.exampleAdt.ListInterface<String> filter = new ArrayList<>();
347     int number = 0;
348     String output = "";
349
350     while (!selectedFilter.isEmpty()) {
351         filter.add(selectedFilter.pop());
352     }
353
354     for (int i = filter.getNumberOfEntries(); i > 0; i--) {
355         Iterator<Tutor> it = tutorList.getIterator();
356         while (it.hasNext()) {
357             Tutor matchTutor = it.next();
358             if (matchTutor.getStatus().equals(filter.getEntry(i))) {
359                 output += String.format("%2d.  ", ++number)
360                     + matchTutor + "\n";
361             }
362         }
363     }
364
365     tutorUI.displayFilteredTutor(output);
```

```
366
367     }
368
369     private void generateReportUI() {
370         int choice;
371         int year, month;
372         do {
373             choice = tutorUI.generateTutorReportMenu();
374
375             switch (choice) {
376                 case 1:
377                     generateSalaryReport();
378                     break;
379                 case 2:
380                     generateRecruitmentReport();
381                     break;
382             }
383         } while (choice != 0);
384     }
385
386     private void generateSalaryReport() {
387         int year = tutorUI.getReportYear();
388         int month = tutorUI.getReportMonth();
389         String reportHeader = String.format("%s %d/%02d",
390             "Monthly Tutor Salary Report", year, month);
391         double totalSalary = 0.0;
392
393         Iterator<Tutor> it = tutorList.getIterator();
394         YearMonth yearMonth = YearMonth.of(year, month);
395         LocalDateTime selectedDate = yearMonth
396             .atEndOfMonth().atTime(23, 59, 59);
397
398         ListInterface<Tutor> validTutor = new
399         CircularDoublyLinkedList<>();
400
401         //get tutor in date range and calculate salary
402         while (it.hasNext()) {
403             Tutor next = it.next();
404             if (isValidTutor(next, selectedDate)) {
405                 validTutor.add(next);
406                 totalSalary += next.getSalary();
407             }
408         }
409     }
410 }
```

```
406         }
407     }
408
409     if (isEmptyTutorReport(validTutor)) {
410         return;
411     }
412     sortTutorList(validTutor, reportHeader, totalSalary);
413
414 }
415
416 private boolean isEmptyTutorReport(ListInterface<Tutor> validTutor) {
417     if (validTutor.isEmpty()) {
418         System.err.println("No Tutor In Selected Date.");
419         GeneralUtil.systemPause();
420         return true;
421     }
422     return false;
423 }
424
425 private void generateRecruitmentReport() {
426     int year = tutorUI.getReportYear();
427     int month = tutorUI.getReportMonth();
428     String reportHeader = String.format("%s %d/%02d",
429         "Monthly Tutor Recruitment Report", year, month);
430
431     Iterator<Tutor> it = tutorList.getIterator();
432
433     ListInterface<Tutor> validTutor = new
434 CircularDoublyLinkedList<>();
435
436     //get tutor in selected date
437     while (it.hasNext()) {
438         Tutor next = it.next();
439         if (isValidTutor(next, year, month)) {
440             validTutor.add(next);
441         }
442     }
443
444     if (isEmptyTutorReport(validTutor)) {
445         return;
446     }
```

```
446
447     sortTutorList(validTutor, reportHeader, -1);
448 }
449
450 private void sortTutorList(ListInterface<Tutor> validTutor, String
reportHeader, double totalSalary) {
451     int pageSize;
452     int choice;
453     do {
454         choice = tutorUI.sortSelection();
455         switch (choice) {
456             case 1:
457 validTutor.sortBy(Comparator.comparing(Tutor::getTutorName),
458                 true);
459                 break;
460             case 2:
461 validTutor.sortBy(Comparator.comparing(Tutor::getTutorName),
462                 false);
463                 break;
464             case 3:
465 validTutor.sortBy(Comparator.comparing(Tutor::getTutorId),
466                 true);
467                 break;
468             case 4:
469 validTutor.sortBy(Comparator.comparing(Tutor::getSalary),
470                 true);
471                 break;
472             case 5:
473 validTutor.sortBy(Comparator.comparing(Tutor::getSalary),
474                 false);
475                 break;
476         }
477
478         if (choice != 0) {
479             pageSize = tutorUI
480                 .getPageSize(validTutor.getNumberOfEntries());
```

```
481         reportPreview(validTutor, reportHeader, pageSize,
totalSalary);
482     }
483
484     } while (choice != 0);
485 }
486
487 private boolean isValidTutor(Tutor tutor, LocalDateTime selectedDate)
{
488     return !tutor.getCreated_at().isAfter(selectedDate) &&
(tutor.isWorking());
489 }
490
491 private boolean isValidTutor(Tutor tutor, int year, int month) {
492     return year == tutor.getCreated_at().getYear()
493         && month == tutor.getCreated_at().getMonthValue()
494         && (tutor.isWorking());
495 }
496
497 private void reportPreview(ListInterface<Tutor> report, String
reportHeader,
498     int pageSize, double totalSalary) {
499     String choice;
500     Paginator page = new Paginator(report, pageSize);
501     String currentPage = getPageContent(page.jumpTo(0), report);
502     do {
503
504         if (currentPage == "") {
505             currentPage =
getPageContent(page.jumpTo(page.currentPage),
506                 report);
507         }
508
509         tutorUI.displayAllTutor(reportHeader,
510             currentPage, false, 90);
511
512         System.out.printf("Page No: %-130d < 1 .. %d >\n",
513             page.currentPage + 1, page.pageNumber);
514         if (totalSalary > 0) {
515             System.out.printf("Total Salary To Pay: RM %, .2f\n\n",
totalSalary);
```

```
516         }
517
518         if (page.isEndOfPage()) {
519             MessageUI.displayInfoMessage(String.format("%88s", "END OF
PAGES"));
520         }
521
522         choice = tutorUI.pageController().toLowerCase();
523         switch (choice) {
524             case ">":
525                 currentPage = getPageContent(page.nextPage(), report);
526                 break;
527             case ">|":
528                 currentPage = getPageContent(page.toEnd(), report);
529                 break;
530             case "<":
531                 currentPage = getPageContent(page.prevPage(), report);
532                 break;
533             case "|<":
534                 currentPage = getPageContent(page.toStart(), report);
535                 break;
536             default:
537                 if (choice.matches("[0-9]+")) { // is integer
538                     currentPage =
getPageContent(page.jumpTo(Integer.parseInt(choice) - 1), report);
539                 } else if (!choice.equals("exit")) {
540                     System.err.println("Invalid command.");
541                     GeneralUtil.systemPause();
542                 }
543                 break;
544         }
545
546     } while (!choice.equals("exit"));
547 }
548
549 private String getPageContent(ListInterface<Tutor> list,
ListInterface<Tutor> original) {
550     String outputStr = "";
551     try {
552         int number = original.indexOf(list.getFirstEntry());
553         Iterator<Tutor> it = list.getIterator();
```

```
554         while (it.hasNext()) {
555             outputStr += String.format("%2d.  ", ++number)
556                 + it.next() + "\n";
557
558         }
559     } catch (Exception e) {
560         GeneralUtil.systemPause();
561     }
562
563     return outputStr;
564 }
565
566 }
```