Subject : *Interface&Collections&Exceptions*

Submission Date : *21.04.2016*

Due Date : *05.05.2016*

Programming Language : *Java SE*

EXPERIMENT

In this experiment, you are expected to develop a dedicated Agenda application, like Google Calendar or other popular calendar applications. Agenda system consist of users, professional meetings, personal or professional appointments, resting time activities such as attending events or courses.

You will use basic Java components such as interfaces, collections and exceptions within your design. If you design a system without interfaces, if you handle the errors without using exceptions properly or if you avoid using the subclasses of Collection, Collections class; your design will not be accepted.

There will be saving, arranging, attending, cancelling and listing commands in the Agenda system. Saving and arranging commands are similar, you should save given agenda components. But, command may cause an error. For example, if command tries to save a duplicated element (duplication can be detected by element id), you should catch this. Also, a meeting arranged for non-existing user should be detected by the system. You must control whether the id is an integer.

A user can attend to events, meetings, appointments or courses, but anniversaries and birthdays are only notifiers, not for attending.

Courses, anniversaries and birthdays are repeated events. Courses' period is a week differing from anniversaries and birthdays which are annual.

Meetings must be arranged within working hours –for weekdays, between 09:00 and 18:00-. Users can attend to an appointment any time, but he/she can only attend to an event or a course in his/her free times, excluding the working hours.

Weekends are resting time. So users can attend appointments, events or courses at all hours just like free times of weekdays between 18:00-09:00.

Listing is a vital part of a system to view robustness. You should use *Collections.sort* method of Java SE before printing the list to output. As, a list command expects to view the agenda in the order of starting time of the activity. You should also provide a *Collection* that consists of agenda elements implementing *Comparable* interface (and as a result compareTo method) so that you can use the *Collections.sort* method. If an agenda element is cancelled, you must remove it from users' agenda who previously decided to attend the activity.

You must use your own exception types for error detection. Any error reported(printed out) must be catched in a try block. Exceptions are *throwable* elements, try to keep an eye on good design oppurtunities to exploit them.

Definitions:

Meeting: A business meeting is a gathering arranged between at least two users. A starting time and duration is required for a meeting. It must start between 09:00 and 18:00 hours. A meeting cannot be scheduled for the resting period.

Appointment: An appoinment is a time period reserved to meet with someone or go somewhere. A starting time and duration is required. An appointment can start any time.

Event: Generally, it is a social or cultural meeting, something done for entertainment. Users join these kind of activities in their free time such as weekends or evenings. Proposed system should support **Concert, Theater and Sport** events. Event date and event name is required. Durations of concerts, theaters and sports are 3 hours, 2 hours and 4 hours respectively.

Anniversary: An anually recurring special date that is reminded or celebrated. It is defined as a date without a specific time, and not a busy time interval.

Birthday: The day on which someone was born. Defined as a date.

Course: It is another agenda item that makes a person busy same day and hours for each week.

INPUT/OUTPUT TYPES

Commands are described below. *Whitespace* character is basic separator of the command's parameters. Also there is a *ws* character after each ':' characters. There is not any *tab* character as separator. You will use capital letters for fixed words, but please do not capitalize any letter of a parameter given by user. Sample usages, outputs and errors are shown to make you understand syntax, examples are provided to demonstrate each command individually, consistency between examples is not guaranteed:

***SAVE/ARRANGE* COMMANDS**

SAVE USER <USER_ID> <FULLNAME>
Sample Usage:
SAVE USER 1001 Bruce Wayne
Sample Output:
Bruce Wayne SAVED

Errors:
DUPLICATED USER ID: 1001 ALREADY EXIST

Descriptions:
This command saves a user with a given id and full name. ID must be unique and an integer. *fullname* does not have to be unique.

ARRANGE APPOINTMENT <DATE> <DURATION> <USER_ID>,<USER_ID> <APPOINTMENT_ID>
Sample Usage:
ARRANGE APPOINTMENT 01/05/2016 17:00 90 1001,1050 2001
Sample Output:
AN APPOINTMENT ARRANGED FOR 1001,1050 AT 01/05/2016 17:00
Errors:
DUPLICATED ID: 2001 ALREADY EXIST
USER NOT FOUND: 1050
INCOMPATIBLE USER: 1001'S AGENDA IS NOT COMPATIBLE

Descriptions:
This command arranges an appointment for two users given by id at given date. ID must be unique and an integer. Users should exist and available (not busy) at the appointment time. For example, if a user has another appointment or meeting etc., you should give an error. Date format will be dd/M/yyyy HH:mm.,

---

ARRANGE MEETING <DATE> <DURATION> <USER_ID>,<USER_ID>,…,<USER_ID> <MEETING_ID>
Sample Usage:
ARRANGE MEETING 01/05/2016 17:00 90 1001,1002,1050 3001
Sample Output
A MEETING ARRANGED FOR 1001,1050 AT 01/05/2016 17:00
Errors:
DUPLICATED ID: 3001 ALREADY EXIST
USER NOT FOUND: 1050
INCOMPATIBLE USER: 1001'S AGENDA IS NOT COMPATIBLE

Descriptions:
This command arranges a meeting between at least two users given by id at given date. ID must be unique and an integer. Users should exist and compatible for arrange an appointment. For example, if a user has another appointment or meeting etc., you should give an error. Meeting will be arranged if and only if all users exist and are available. Date format will be dd/M/yyyy HH:mm. A meeting has to be STARTED during legal working-time(09.00-18.00), but end of a meeting does not have to be in this range.

ARRANGE CONCERT <QUOTA> <DATE> <EVENT_ID> <EVENT_NAME>
Sample Usage:
ARRANGE CONCERT 10 01/05/2016 21:00 4001 SEZENLI YILLAR
Sample Output:
SEZENLI YILLAR ARRANGED AT 01/05/2016 21:00
Errors:
DUPLICATED ID: 4001 ALREADY EXIST

Description:
This command arranges a concert at a given date. ID must be unique and an integer. Date format will be dd/M/yyyy HH:mm. Default duration of a concert is 3 hours.

ARRANGE THEATER <QUOTA> <DATE> <EVENT_ID> <EVENT_NAME>
Sample Usage:
ARRANGE THEATER 20 01/05/2016 21:00 4002 MAKBET
Sample Output
MAKBET ARRANGED AT 01/05/2016 21:00

Errors
DUPLICATED ID: 4002 ALREADY EXIST

Description:
This command arrange a theater at given date. ID must be unique and an integer. Date format will be dd/M/yyyy HH:mm. Default duration of a theater is 2 hours.

ARRANGE SPORT <QUOTA> <DATE> <EVENT_ID> <EVENT_NAME>
Sample Usage:
ARRANGE SPORT 5 01/05/2016 21:00 4002 THY EUROLEAGUE MATCH
Sample Output:
THY EUROLEAGUE MATCH ARRANGED AT 01/05/2016 21:00
Errors:
DUPLICATED EVENT ID: 4002 ALREADY EXIST

Description:
This command arrange a sport activity at given date. ID must be unique and an integer. Date format will be dd/M/yyyy HH:mm. Default duration of a match is 4 hours.

SAVE ANNIVERSARY <DATE> <USER_ID> <ANNIVERSARY_ID> <DESCRIPTION>
Sample Usage:
SAVE ANNIVERSARY 01/05/2016 1001 7001 WEEDING ANNIVERSARY
Sample Output:
WEEDING ANNIVERSARY ADDED TO Bruce Wayne'S AGENDA
Errors:
DUPLICATED ID: 7002 *ALREADY EXIST*
USER NOT FOUND: 1050

Description:
This command saves an anniversary for given date of each year. ID must be unique and an integer. Date format will be dd/M/yyyy. Users should exist. An anniversary is just an information, does not restrain users' other agenda activities.

SAVE BIRTHDAY <DATE> <USER_ID> <BIRTHDAY_ID> <DESCRIPTION>
Sample Usage:
SAVE BIRTHDAY 01/05/2016 1001 7002 ALFRED'S BIRTHDAY
Sample Output
ALFRED'S BIRTHDAY ADDED TO Bruce Wayne'S AGENDA
Errors
DUPLICATED ID: 7002 *ALREADY EXIST*
USER NOT FOUND: 1050

Description
This command saves a birthday for given date of each year. ID must be unique and an integer. Date format will be dd/M/yyyy. User should exist. A birthday record is just an information, does not restrain users' other agenda activities. The user is the agenda's owner, not the birthday boy!

ARRANGE COURSE <QUOTA> <DATE> <COURSE_ID> <COURSE_NAME>
Sample Usage:
ARRANGE COURSE 2 01/05/2016 20:00 5001 TENNIS COURSE
Sample Output:
TENNIS COURSE ARRANGED ON TUESDAY 20:00
Errors:
DUPLICATED ID: 5001 ALREADY EXIST

Description:
This command saves a course for given week of day and hour of each week. ID must be unique and an integer. Date format will be dd/M/yyyy HH:mm. Default duration of a course is 2 hours.

ATTEND COMMANDS

```
ATTEND EVENT <EVENT_ID> <USER_ID>,<USER_ID>,,…,<USER_ID>
```
Sample Usage:
```
ATTEND EVENT 4001 1001,1002,1003,1004,1050,1051
```
Sample Output:
```
2 USERS ADDED TO ATTENDENCE LIST OF SEZENLI YILLAR
```
Errors:
USER NOT FOUND: 1003
USER NOT FOUND: 1051
UNAVAILABLE USER: 1002
EVENT NOT FOUND: 4001
QUOTA FULL: 1004
QUOTA FULL: 1050

Description:
This command registers the given users as they will attend the event. This event may be a concert, theater or a sport game. Event and users must exist. If some of given user ids does not exist, omit these users, save existing users and give an error for the non-existing users as given Errors section of the command above. You should also check the quota of the event and only accept first *n* existing users. Also give an error for unregistered users not in the first n.

```
ATTEND COURSE < COURSE _ID> <USER_ID>,<USER_ID>,,…,<USER_ID>
```
Sample Usage:
```
ATTEND COURSE 5001 1001,1002,1003,1004,1050,1051
```
Sample Output:
```
2 USERS ADDED TO ATTENDENCE LIST OF TENNIS COURSE
```
Errors:
USER NOT FOUND: 1003
USER NOT FOUND: 1051
UNAVAILABLE USER: 1002
COURSE NOT FOUND: 4001
QUOTA FULL: 1004
QUOTA FULL: 1050

Description:
This command registers given users as attending the course. Course and users must exist. If some of given user ids does not exist, omit these users, save existing users and give an error for non-existing users as given Errors section above. You should also check the quota of the course and only accept first *n* existing users. Also give an error for this rejection as given above.

CANCEL COMMANDS

```
CANCEL APPOINTMENT <APPOINTMENT_ID>
```
Sample Usage:
```
CANCEL APPOINTMENT 2001
```
Sample Output:
```
APPOINTMENT CANCELLED: 2001
```
Errors:
APPOINTMENT NOT FOUND: 2001

Description:
This command cancels an appointment if it exists. If an appointment is cancelled, it should be removed from users' agenda also.

CANCEL MEETING < MEETING_ID>
Sample Usage:
CANCEL MEETING 3001
Sample Output:
MEETING CANCELLED: 3001
Errors:
MEETING *NOT FOUND: 3001*

Description:
This command cancels a meeting if exists. If a meeting is cancelled, it should be removed from users' agenda also.

CANCEL EVENT < EVENT _ID>
Sample Usage:
CANCEL EVENT 4001
Sample Output:
EVENT CANCELLED: 4001
Errors:
EVENT *NOT FOUND: 4001*

Description:
This command cancels an event if exists. If an event is cancelled, it should be removed from users' agenda also.

CANCEL COURSE < COURSE _ID>
Sample Usage:
CANCEL COURSE 5001
Sample Output:
COURSE CANCELLED: 5001
Errors:
COURSE *NOT FOUND: 5001*

Description:
This command cancels a course if exists. If a course is cancelled, it should be removed from users' agenda also.

REPORT COMMANDS

LIST DAILY <USER_ID> <DATE>
Sample Usage:
LIST DAILY 1002 11/06/2016
Sample Output:
```
ANNIVERSARY: WEDDING ANNIVERSARY 11/06/2016
BIRTHDAY: ALFRED'S BIRTHDAY 11/06/2016
APPOINTMENT WITH Bruce Wayne AT 11/06/2016 15:00
MEETING WITH Bruce Wayne,Clark Kent AT 11/06/2016 17:00
SEZENLI YILLAR AT 11/06/2016 21:00
TENNIS COURSE AT 11/06/2016 22:00
```

Errors:
USER NOT FOUND: 1050

Descriptions:
This command lists a Daily agenda of given user. User must exist and list of agenda must be in sorted by date. Anniversaries and birthdays come first. While printing appointments and meetings, do not print listed user (given as listing parameter -userid-), only print other attenders.

LIST WEEKLY <USER_ID> <DATE>
Sample Usage:
LIST WEEKLY 1002 11/06/2016
Sample Output:
```
APPOINTMENT WITH Bruce Wayne AT 11/06/2016 15:00
MEETING WITH Bruce Wayne,Clark Kent AT 13/06/2016 17:00
ANNIVERSARY: WEDDING ANNIVERSARY 15/06/2016
SEZENLI YILLAR AT 15/06/2016 21:00
TENNIS COURSE AT 17/06/2016 20:00
```
Errors:
USER NOT FOUND: 1050

Descriptions:
This command lists a Weekly agenda of given user. User must exist and list of agenda must be in sorted by date. Anniversaries and birthdays come first within same date,

---

LIST MONTHY <USER_ID> <DATE>
Sample Usage:
LIST MONTHY 1002 11/06/2016
Sample Output:
```
APPOINTMENT WITH Bruce Wayne AT 11/06/2016 15:00
TENNIS COURSE AT 16/06/2016 20:00
MEETING WITH Bruce Wayne,Clark Kent AT 23/06/2016 17:00
TENNIS COURSE AT 23/06/2016 20:00
SEZENLI YILLAR AT 25/06/2016 21:00
TENNIS COURSE AT 30/06/2016 20:00
TENNIS COURSE AT 07/07/2016 20:00
```
Errors:
USER NOT FOUND: 1050

Descriptions:
This command lists a Monthly agenda of given user. User must exist and list of agenda must be in sorted by date. Anniversaries and birthdays come first within same date, You should be aware that a recurring event might be listed multiple times in the same list. While printing appointments and meetings, do not print listed user (given as listing parameter -userid-), only print other attenders.

LIST ATTENDANCE <COURSE|APPOINTMENT|MEETING|EVENT> <ID>
Sample Usage:
LIST ATTENDANCE EVENT 4001
Sample Output:
```
Bruce Wayne,Clark Kent,Muhtar Kent
```
Errors:
<COURSE|APPOINTMENT|MEETING|EVENT> *NOT FOUND: 4001*

Descriptions:
This command lists attendances of a course, an appointment, a meeting or an event. Listed activity should exist. List of users’ fullname must be reported.

REPORT

You should follow lab report format document on ftp://ftp.cs.hacettepe.edu.tr/pub/dersler/genel/FormatForLabReports.doc. Show your class diagram to make your design more readable. You should also explain how to add new abilities to your design such as:

- Assume you need to handle Holidays. It can be legal/religious holidays or annual leave. A user can not attend a meeting during holidays of course. Think about that.

RULES and HINTS

- Arrays are forbidden except String operations. Use subclasses of Collection class.
- Design as modular as possible. Try more each time.
- Use OOP concepts such as Inheritance, Encapsulation, Polymorphism
- Consider interfaces for better solutions
- Do not implement your sorting algorithm. Use collections.sort instead.
- Avoid using instanceof
- Avoid using static variables. It is not forbidden, but don't use if not really necessary.
- String.split or StringTokenizer is sufficent to solve your parsing problems
- You can use Calendar class some of the date operations. Think about Calendar.getInstance() (we don't use new keyword to construct a calendar object)
- SimpleDateFormat is another class you may need to utilise.
- You won't submit javadoc in this experiment. But you MUST obey javadoc rules, only do not generate javadoc.
- You will be graded not only for the outputs, but also readibility, modularity, OOP design, comment lines and report(30 points).
- Understand output clearly, ask if you don't. Your experiment has to give an output exactly same as the instructor's output (with white spaces, commas, dots etc.)

NOTES

- Use Eclipse for development
- Use UNDERSTANDABLE names for your variables, functions and classes.
- Write READABLE SOURCE CODE blocks.
- Use EXPLANATORY COMMENTS in your source codes.
- Don't miss the deadline.
- Save all your work until the assignment is graded.

```
 |--src
        -- Main.java
        -- *.java
|--report
        -- report.pdf
```

REFERENCES

[1] http://www.mkyong.com/java/java-object-sorting-example-comparable-and-comparator/

[2] https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html

[3] http://www.mkyong.com/java/java-date-and-calendar-examples/