Practices for Lesson 11: Working with Arrays, Loops, and Dates

Chapter 11

Practices for Lesson 11

Practices Overview

In these practices, you will use an ArrayList to iterate through data.

Practice 11-1: Iterating Through Data

Overview

In this practice, you will write code to allow teams of any size to be created from a comma-separated list of names stored in a `String`. You will find a new `PlayerDatabase` class in the `utility` package. At the moment, it contains only a `String` with a comma-separated list of names.

Tasks

1. Close any open code tabs, and open the **11-ArraysLoopsDates_Practice1** project.
2. In the `PlayerDatabase` class in the utility package, create an `ArrayList` and populate it with the names in the `String authorList`.
   a. Open the `PlayerDatabase` class and declare an `ArrayList` of type `Player`. Name it `players`.
      ```
      private ArrayList <Player> players;
      ```
   b. Click the red error icon in the margin. You will see that you need to import `java.util.ArrayList`. Do this, and then modify the `java.util.ArrayList` import so that it now imports `java.util.*` (all classes in `java.util`). The import statement will now look like this.
      ```
      import java.util.*;
      ```
   c. Add another import for `soccer.Player`.
      ```
      import soccer.Player;
      ```
   d. You need to find some way to iterate through the names and add each to the `ArrayList`. Look up `StringTokenizer` in the Javadocs. Notice that it is also in the `java.util` package.
   e. Create a no-argument constructor for the `PlayerDatabase` class.
      ```
      public PlayerDatabase() {
      }
      ```
   f. Within the constructor, create a `StringTokenizer authorTokens` that is built on the `authorList String`.
      ```
      StringTokenizer authorTokens =
         new StringTokenizer(authorList, ",");
      ```
   g. Instantiate the `ArrayList players`.
      ```
      players = new ArrayList();
      ```
   h. Create a ***while*** loop to iterate through the `StringTokenizer`. On each iteration, add a new `Player` to the `ArrayList`. Notice how easy it is to do this. With an array, you would have to find out the number of players in `authorList` and then add each player to consecutive elements of the array. Using an array is possible, but not as easy as using an `ArrayList`.
      ```
      while (authorTokens.hasMoreTokens()) {
         players.add(new Player(authorTokens.nextToken()));
      }
      ```
      Now you have an ArrayList of eligible players that you can use to populate teams.

3. Create a method to return an arbitrarily sized team.
   a. Create a method, `getTeam`, that takes an `int` (`numberOfPlayers`) and returns an array of `Players`.

```
public Player[] getTeam(int numberOfPlayers){
}
```

   b. Within the `getTeam` method, create a `Player` array named `teamPlayers`.

```
Player[] teamPlayers = new Player[numberOfPlayers];
```

   c. Now create a ***for*** loop to iterate through this array.

```
for (int i = 0; i < numberOfPlayers; i++) {
}
```

   d. On each iteration of the loop, randomly select a `Player` from the `players ArrayList` and add that player to the `teamPlayers` array.

```
int playerIndex = (int) (Math.random()*players.size());
teamPlayers[i] = players.get(playerIndex);
```

   e. Remove the player just selected from the `players ArrayList` (this is to ensure that the same player cannot play for more than one team). Notice how this is easy to do with an `ArrayList`; it would be much more difficult if using an array.

```
players.remove(playerIndex);
```

   f. Just after the ***for*** loop, return the `teamPlayers` array.

```
return teamPlayers;
```

4. Modify the `createTeams` method to use the `PlayerDatabase` class.
   a. Go to the `createTeams` method of `League` and remove all code that creates a `Player` object or a `Player` array. The following code should remain (do not worry that it has errors).

```
Team team1 = new Team("The Greens", thePlayers1);
Team team2 = new Team("The Reds", thePlayers2);
Team[] theTeams =  {team1, team2};
return theTeams;
```

   b. Instantiate a new `PlayerDatabase` object at the start of the `createTeams` method. You will need to import it also.

```
PlayerDatabase playerDB = new PlayerDatabase();
```

   c. Modify the lines that instantiate `team1` and `team2` so that they now use `playerDB` for the players. The lines will now look like this.

```
Team team1 = new Team("The Greens", playerDB.getTeam(3));
Team team2 = new Team("The Reds", playerDB.getTeam(3));
```

   d. Run the application a few times to test it. It should work as before, except now the players will be randomly assigned to each team.
5. Make the `createTeams` method more general-purpose by passing in team names and team sizes.
   a. Change the `createTeams` method signature to receive a `String` with the team names and an `int` for the number of players in each team.

```
public Team[] createTeams(String teamNames, int teamSize) {
```

b. Because the team names will be passed in as a comma-separated list, you must (as before) use a `StringTokenizer` to set up a ***for*** loop to iterate through however many teams have been specified. Create the `StringTokenizer` now (you may need to click the red dot to import it). Put this line just below the line that instantiates the `PlayerDatabase` object.

```
StringTokenizer teamNameTokens = new
   StringTokenizer(teamNames,",");
```

c. Create a `Team` array called `theTeams`. It will have one element for each team name passed in.

```
Team[] theTeams = new Team[teamNameTokens.countTokens()];
```

d. Write a ***for*** loop that iterates through the array and creates a new `Team` for each element. You can use the `StringTokenizer` method `nextToken` to get the team name, and the `PlayerDatabase` method `getTeam` to get the array of type `Player`.

```
for (int i = 0; i < theTeams.length; i++){
   theTeams[i] = new Team(teamNameTokens.nextToken(),
      playerDB.getTeam(teamSize));
}
```

e. Remove the remainder of the method except for the return statement.

6. Modify the call to `getTeams` in the `main` method of `League` to pass in team names and team size.

a. Replace the current call to the createTeams method with the following:

```
Team[] theTeams = theLeague.createTeams("The Robins,The
Crows,The Swallows", 3);
```

b. Run the application a few times. It should work as before.

c. The method `createGames` is currently hard-coded; otherwise you could change the number of teams by changing the call to `createTeams`. However, you can change team size, so try making your league 5 per side.

Rewrite `createGames` to Generate All-Play-All Set of Games

7. Create a nested loop in `createGames` to return an array of Games that ensures that all teams play each of their competitors.

a. Delete everything in the `createGames` method except the return statement.

b. Instantiate an `ArrayList` to hold the games that you will create. (You may need to import `ArrayList`.)

```
ArrayList<Game> theGames = new ArrayList();
```

c. Create a ***for*** loop to iterate through all the teams in the `Team` array.

```
for (Team homeTeam: theTeams) {
}
```

d. For each `Team` you need to create a `Game` matching that `Team` against one of their competitors. Therefore, create another ***for*** loop within the one you just created. Use `awayTeam` as the local variable name this time.

```
for (Team awayTeam: theTeams) {
}
```

e. All you need to do now is to create a `Game` for each iteration of the inner loop. However, that means that "The Crows" could end up playing "The Crows." Therefore, write an ***if*** statement to exclude this possibility. The entire nested loop will look like this.

```
for (Team homeTeam: theTeams) {
   for (Team awayTeam: theTeams) {
      if (homeTeam!=awayTeam) {
         theGames.add(new Game(homeTeam,awayTeam));
      }
   }
}
```

f. Finally, you need to return an array, not an `ArrayList`. Therefore, you must use the `toArray` method of `ArrayList`. Just use the following code; it will be explained later:

```
return (Game[]) theGames.toArray( new Game[1] );
```

g. Test the application. It should work as before except that there are now more games than before (the Swallows get a chance!). The way it is set up now, teams play each other twice, once at home and once away.

This is the end of this practice. Shut down any NetBeans tabs that contain Java code.

Practice 11-2: Working with LocalDateTime

Overview

In this practice, you work with the `LocalDateTime` object so that games have a `LocalDateTime` attribute.

Tasks

1. Open the **11-ArraysLoopsDates_Practice2** project in NetBeans.
2. Add a new attribute to the `Game` object.
   a. Add a `LocalDateTime` attribute, `theDateTime`, just below the `goals` attribute.
   ```
   private LocalDateTime theDateTime;
   ```
   b. You will see an error because this class is not in `java.lang`. Add the `java.time.*` package as an import by clicking the red dot and selecting the first option Add import…
   c. You will see that the import is only for `LocalDateTime`. Therefore, replace `LocalDateTime` in the import statement with a `*` (now all classes in this package will be available to you). The import statement will now look like this:
   ```
   import java.time.*;
   ```
   d. Use the NetBeans refactor feature to create getter and setter methods for this attribute.
   e. Modify the constructor of the `Game` class to set this `LocalDateTime` attribute. The constructor will now look like this (new code is bolded):
   ```
   public Game(Team homeTeam, Team awayTeam,
      LocalDateTime theDateTime) {
      this.homeTeam = homeTeam;
      this.awayTeam = awayTeam;
      this.theDateTime = theDateTime;
   }
   ```
3. Modify the `getDescription` method of the `Game` class to work with this new attribute.
   a. Modify the `getDescription` method of `Game` so that it now returns the date and time of the game. The line that you need to modify currently is:
   ```
   returnString.append(this.getHomeTeam().getTeamName() + " vs. " +
      this.getAwayTeam().getTeamName() + "\n" );
   ```
   After you modify it, it will be (with new code in bold):
   ```
   returnString.append(this.getHomeTeam().getTeamName() + " vs. " +
      this.getAwayTeam().getTeamName() + "\n" +
      "Date " +
      this.theDateTime.format
      (DateTimeFormatter.ISO_LOCAL_DATE) + "\n");
   ```
   You may have to import DateTimeFormatter.
   b. In the `createGames` method of League, modify `theGames.add` method inside the ***if*** block to pass a new `LocalDateTime` object to the constructor of `Game`. Use `LocalDateTime.now()` to instantiate the `LocalDateTime` object (new code is bolded). You will have to import `LocalDateTime`.
   ```
   theGames.add(new Game(homeTeam, awayTeam, LocalDateTime.now()));
   ```

c. Run the application. You should see the time for the game before the description of the play. (At the moment this is a little strange because the `LocalDateTime` value for all the games is now!)

4. Modify the `createGames` method to increment the date that each game is scheduled to be played.
   a. In the `createGames` method of `League`, add a line at the start of the method to declare an `int` variable, `daysBetweenGames`, and initialize it to `0`.
      ```
      int daysBetweenGames = 0;
      ```
   b. At the start of the *if* block (inside the inner loop) add a line to increment the `daysBetweenGames` variable by `7`.
      ```
      daysBetweenGames += 7;
      ```
   c. Modify the call to the `Game` constructor so that each `Game` is now scheduled seven days later than the previous one. Look in the Javadocs for `LocalDateTime` to see what method to use (new code is bolded below).
      ```
      theGames.add(new Game(homeTeam, awayTeam,
      LocalDateTime.now()).plusDays(daysBetweenGames));
      ```
   d. Run the application again. You should now see that each game is now set for seven days later than the previous game. Of course, now it is a little strange because you also see the game result even though that is in the future! (But remember the random game generator is principally for testing and would not be used in the real world operation of the application).

5. Write a `getLeagueAnnouncement` method that calculates how long the League lasts.
   a. At the bottom of the `League` class, add a new method, `getLeagueAnnouncement`.
      ```
      public String getLeagueAnnouncement(Game[] theGames){


      }
      ```
   b. Because you will need the `Period` class, look it up in the Javadocs now. Can you see which method you will need to create a `Period` object? You will need the static method `between` that takes two `LocalDate` parameters and returns a `Period` object.
   c. In addition, you need to deal with the fact that the attribute you used on `Game` is `LocalDateTime` not `LocalDate`. How can you convert `LocalDateTime` to `LocalDate`? Look in the Javadocs for `LocalDateTime`. Examine the method `toLocalDate`.
   d. Add a line that creates a `Period` object based on the dates of the first and last games. You will need to use NetBeans to import the `java.time` package for `Period`.
      ```
      Period thePeriod =
        Period.between(theGames[0].getTheDateTime().toLocalDate(),
        theGames[theGames.length - 1].getTheDateTime().toLocalDate());
      ```
   e. Use `String` concatenation (or a `StringBuilder` and the `append` method) to return a `String` that describes the length of the `League` tournament. For example:
      ```
      return "The League is scheduled to run for " +
         thePeriod.getMonths() + " month(s), and " +
         thePeriod.getDays() + " day(s)\n";
      ```

6. Test the application.
   a. Just before the ***for*** loop in the `main` method of League, add a `System.out.println` to print the description of the league based on the `getGamesAnnouncement` method.
      `System.out.println(theLeague.getLeagueAnnouncement(theGames));`
   b. Run the application. Scroll up to the top of the output. You will see something like the following.

```
The League is scheduled to run for 1 month(s), and 4 day(s)

The Greens vs. The Reds
Goal scored after 17.0 mins by Rafael Sabatini of The Reds
Goal scored after 32.0 mins by Robert Service of The Reds
Goal scored after 35.0 mins by Geoffrey Chaucer of The Greens
The Reds win (1 - 2)

The Reds vs. The Greens
Goal scored after 21.0 mins by Rafael Sabatini of The Reds
Goal scored after 24.0 mins by George Eliot of The Greens

<-- Further output omitted -->
```

   c. Try changing the value of the `daysBetweenGames` variable to see whether the length of time required to run the league changes.

This is the end of this practice. Shut down any tabs containing Java code.