Chapter 13

Objects of Arrays

13.1 Exercises

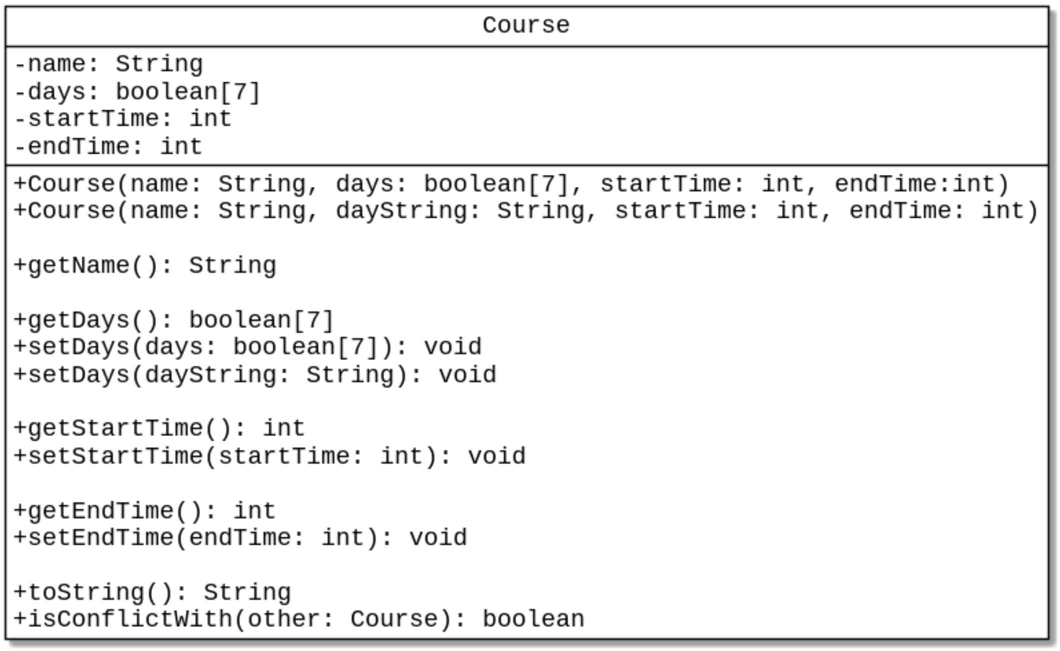
Exercise 13.1 The previous chapter's exercise that implemented a `Course` class lacks one important feature: most courses meet more than one day a week. Figure 13.1 is a revised UML diagram for an implementation of `Course` that solves that problem:

Instead of having one day of the week, `Course` objects will now contain an array of seven `boolean` values, where `true` indicates that the course meets on that day. Array entry 0 corresponds to Monday, 1 to Tuesday, and so on until entry 6 for Sunday.

Notice that there are two constructors. The first one has you provide an array of booleans; the second has you provide a `String` that represents the days on which a course meets. This string will have the format of one or more day abbreviations separated by slashes. The abbreviations are `"M"`, `"T"`, `"W"`, `"Th"`, `"F"`, `"Sa"`, and `"Su"`. Your constructor will translate this string into the appropriate array of `boolean` for the `days` attribute. This gives you two ways to construct a course:

```java
Course course1 = new Course("ECON 101",
   new boolean[]{true, false, true, false, true, false, false},
   915, 1045);

Course course2 = new Course("ECON101", "M/W/F", 915, 1045);
```



```
Course
-name: String
-days: boolean[7]
-startTime: int
-endTime: int
+Course(name: String, days: boolean[7], startTime: int, endTime:int)
+Course(name: String, dayString: String, startTime: int, endTime: int)

+getName(): String

+getDays(): boolean[7]
+setDays(days: boolean[7]): void
+setDays(dayString: String): void

+getStartTime(): int
+setStartTime(startTime: int): void

+getEndTime(): int
+setEndTime(endTime: int): void

+toString(): String
+isConflictWith(other: Course): boolean
```

Figure 13.1: UML Diagram for a Course class

Similarly, the two `setDays` methods allow you to set the days on which a course meets by passing in array of seven `boolean` values or a `String` as described. These are equivalent:

```
// change to Tuesday-Thursday
course1.setDays(new boolean[]{false, true, false, true,
   false, false, false});

course2.setDays("T/Th");
```

Instead of the `compareTo` method from the previous exercise, you will write an `isConflictWith` method that returns `true` if the days and times of the two courses overlap, `false` otherwise. *Hint*: first, check to see if both courses have a day in common. For example, a Monday/Wednesday course and a Tuesday/Wednesday course might have a conflict. A Monday/Wednesday course cannot have a conflict with a Tuesday/Thursday course since the two courses are on different days.

If there is a day conflict, then check the times to see if they overlap. Two courses overlap if the course with the earlier start time ends after the course with the later start time begins. Thus, a course from 0915 to 1045 overlaps with a class from 1015 to 1230. If the end of the earlier course is the same time as the start of the later one, that is *not* an overlap: a class from 0915 to 1045 does not overlap a class from 1045 to 1215. (We presume that you can teleport instantly from one classroom to the other.)

Once you have the `Course` class written, write a test program whose `main` method sets up this array of courses:

Name	Days	Start Time	End Time
ECON 101	M/W	0915	1135
PHYS 002	Tu/Th	1025	1300
COMSC 079C	M	0745	0930
ACCTG 0063A	W/F	1015	1235
CHEM 001	Tu/Th	1700	1925
MATH 017	M/Tu/W/Th	1305	1545
ART 099	Tu/Th	1205	1335

Once you have set up this array, your code will look at all the pairs of courses and display the ones that have a time conflict.

Here is what the output might look like:

```
Course conflicts:
ECON 101 M/W (0915-1135) and COMSC 079C M (0745-0930)
ECON 101 M/W (0915-1135) and ACCTG 063A W/F (1015-1235)
PHYS 002 T/Th (1025-1300) and ART 099 T/Th (1205-1635)
MATH 017 M/T/W/Th (1305-1545) and ART 099 T/Th (1205-1635)
```

Do not duplicate the conflict information. For example, ECON 101 conflicts with COMSC 079C, so there is no need to output that COMSC 079C conflicts with ECON 101.

Your code must work for any set of courses. If there are no conflicts, then your program should output a message to that effect, for example:

```
Course conflicts:
None found
```

Exercise 13.2 Create a `Competitor` class. Make a UML diagram *before* you start implementing the class, according to the following specifications.

A `Competitor` object has these private attributes:

- A `name`, which is a `String`.
- A `team`, which is a `String` that can have the value `"Green"` or `"Orange"`.
- An array of `scores`, which is an array of three `int` values that gives the competitor's score in each of three games. A value of -1 means that the competitor did not take part in the game.

Write a constructor and getters and setters for each of the attributes. Write a `toString` method that displays a competitor's information. (I have deliberately left this open-ended. Use your judgment as to what the return value should look like.)

Now that you have the `Competitor` class, write a program whose `main` method sets up the following array of competitors:

Name	Team	First Game	Second Game	Third Game
Tom	Orange	5	17	22
Joe	Green	3	14	22
Maria	Green	6	18	21
Fred	Orange	15	-1	23
Carlos	Orange	17	15	24
Phuong	Green	7	19	21
Enrique	Green	3	16	20
Nancy	Orange	9	12	24

Then compute and print:

- The average score for every person in the array
- The average score for each game

- The average score for each team

Exercise 13.3 This exercise will familiarize you with `ArrayList`s. You will use the `Person` class that has been written for you and is available in the code repository that accompanies this book. Figure 13.2 shows the UML diagram for the class.

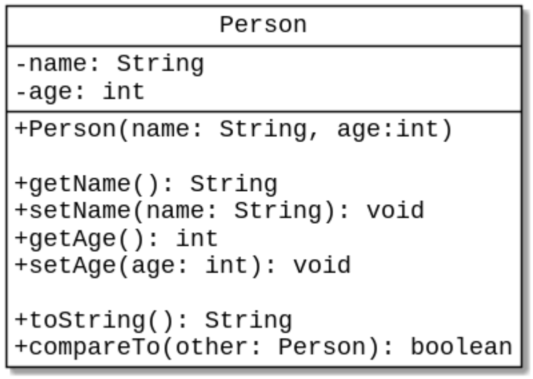


Figure 13.2: UML Diagram for the Person class

The `compareTo` method compares the names; if they are equal, it compares the ages. In this exercise, we won't allow duplicate names, but the code is written to be general enough to work in other applications.

Write a program that creates an `ArrayList<Person>` and repeatedly asks users to either:

- Add a name and age to the list if it isn't already there.
- Remove a name (and its age) from the list if it is in the list.
- Change the age for a person if the person is in the list.
- Finish

When the user finishes, the program sorts the `ArrayList` *by age*, which means you will *not* use the `compareTo` method, which would sort them by name. It then prints the sorted list of users:

```
A)dd, R)emove, C)hange age, F)inish: A
Enter name: Joe
Enter age: 2r
Please use digits for your answer.
Enter age: 24
A)dd, R)emove, C)hange age, F)inish: A
Enter name: Marta
Enter age: 47
A)dd, R)emove, C)hange age, F)inish: a
Enter name: Douglas
Enter age: 33
A)dd, R)emove, C)hange age, F)inish: A
Enter name: Vinh
Enter age: 18
A)dd, R)emove, C)hange age, F)inish: a
Enter name: Vinh
Vinh is already in the list.
A)dd, R)emove, C)hange age, F)inish: r
Enter name: Josephine
Josephine is not in the list.
A)dd, R)emove, C)hange age, F)inish: R
Enter name: Douglas
Removed Douglas.
A)dd, R)emove, C)hange age, F)inish: C
Enter name: Vinh
Enter age: 19
Vinh is now 19 years old.
A)dd, R)emove, C)hange age, F)inish: F

People sorted by age:
Vinh (19)
Joe (24)
Marta (47)
```

Your program should allow the commands to be input in either upper or lower case. As in the sample output, it should produce appropriate error messages if the user tries to add a duplicate name, remove a non-existent name, or give an invalid command.