```
Campspot Developer Programming Challenge Notes
Author: Kenneth C. LaMarca
Date:   03/31/2017 - 04/02/2017

/////////////////////////////////////////////////////////////////////////
Quick Start:
Command Line operations to build executable jar
From the top project folder

BUILD:
javac -classpath ./jars/*.jar ./src/com/campspot/*.java -d .
jar cvfm ReservationSystem.jar manifest.txt com jars

EXECUTE:
java -jar ReservationSystem.jar (jsonfile) (testcases) ("-v" optional)
testcases : AllTests , ReservationTests, CampsiteTests

I did use IntelliJ IDEA to code / build / run and test this code. I will
include the built JAR from the IntelliJ project as CampSpot.jar
/////////////////////////////////////////////////////////////////////////


Implement a camping reservation system that utilizes the gap rule.

The gap rule states that a reservation cannot be made if it will leave a specified gap.

Requirements:

Gap size must be configurable
Output each availible campground following the gap rule
Must include executable test cases
Must include a readme file explaining (this will be the readme)
    -   how to build and run your program and tests
    -   high level description of the approach
    -   any assumptions of considerations made


High Level Approach:
    Use of GSON library to create java objects out of json

    Classes:
        Campsite
            -   id -int
            -   name -string
            -   List<Reservation> sorted
            -   validReservation(search,List<GapRule> rules)
        Reservation
            -   startDate (LocalDate object)
            -   endDate (LocalDate object)
            *   compareTo
            *   equals
        Search
            -   startDate
            -   endDate
        GapRule
            -   gapsize - int
        Input (gson placeholder class)
            -   search -Search
            -   List<Campsite> campsites
            -   List<GapRule> gapRules
            -   List<Reservation> reservations
            *   initializeData()    , campsites should contain list of reservations gson
            doesn't handle this without a custom parser
            *   toString    ,   print out the data in all the objects for sanity
```

Campsite.validReservation(search,List<GapRule> rules);

- This function is the meat of the functionality. It takes the search object and creates a new Reservation object from the search parameters and the campsite's id. The new reservation is inserted into the reservations list if it .contains() returns false

- List.contains() uses the objects .equals function to compare the object with those in the list Reservation.equals() is overwritten to include startDate , endDate comparisons and to filter out overlapping reservation dates. Reservations that overlap or are contained within an existing reservation should not be added to the list.

- The reservation list is sorted after the new reservation is added. The Collections.sort() function utilizes the objects compareTo operation to decide on how it should be sorted. Reservation has implemented the comparable<Reservation> interface. Reservations are compared by start date to end date and vice versa.

- After being sorted the new reservation index is found and that reservation is compared with it's immediate neighbors via the checkRules function.

Campsite.checkRules(Reservation compareRes, Reservation newRes, List<GapRule> rules)

- This function verifies that the day gap between the newRes and the compareRes does not violate any of the gap rules. It utilizes the ChronoUnit.Days.between function from the java.time library. The day gap is one less than the return from Days.between as day gap is entirely exclusive.

- After all checks are finished the reservation is removed from the list of reservations for this campsite

- The return from validReservation is just a boolean which is true if none of the checks for validity have failed.

Assumptions:

- The first argument in args[] is always the json file.
- Existing reservations do not have to follow the gap rules.
  - In the example input there are reservations that do not follow the gap rules therefore I decided that existing reservations would not be verified.

    This set violates the gapRule of 2.
    ```
    {"campsiteId": 7, "startDate": "2016-06-03", "endDate": "2016-06-04"},
    {"campsiteId": 7, "startDate": "2016-06-07", "endDate": "2016-06-09"}
    ```

- Gap rule would be exclusive. The gap between 06-03 and 06-05 would only be 1 day.

- All dates are in the format YYYY-MM-DD

- A reservations contained within eachother would be equivalent by the compareTo method or equals method I decided that it was outside the scope of this project to determine which reservation should be removed

Testing:

- I could have used a testing framework but in the interest of time I just wrote a simple testing functions that can be called via command line arguments

- For the sake of time testing focused mainly on the reservation.compareTo, reservation.equals, List<Reservation> Collections.sorting and campsite.validReservation functions.

- Testing is grouped into 2 sets Reservation object tests and Campsite tests.

- Campsite.validReservation tests utilize a variety of gapRule lists and reservation dates that test boundary cases as well as adding checking a reservation at either end of the sorted list.

- Reservation CompareTo and equals tests are designed to test every possible result. Since the equals and compare functions are really just a series of if statements I wanted to purposefully hit every if statement.

- Running the tests just involves an extra command line argument or multiple.

```
    ex. java -jar ReservationSystem.jar test-case.json AllTests
        java -jar ReservationSystem.jar test-case.json ReservationTests
        java -jar ReservationSystem.jar test-case.json ReservationTests CampsiteTests
        java -jar ReservationSystem.jar test-case.json ReservationTests -v
```

- A -v as the last command argument will activate verbose mode which gives more details with each test