This is an example of my Selenium WebDriver with Java automation skills. I was given this take-home assignment a couple of years ago as part of a QA Automation job interview. I have recently updated it to reflect the significant changes to the Orbitz.com site since then (travel related web sites typically change a great deal over time). This was the testing challenge:

1. Go to orbitz.com
2. Select the Flights tab and prepare for a Multi-city trip with 3 legs
3. On the Search page input flight details such that it completes a round trip
    a. Only one adult passenger, no children
    b. Trip is to start 35 days from the current date
    c. Airport 1 to Airport 2 to Airport 3 then back to Airport 1
    d. Gap between each leg is 7 days
4. Search for this trip and on the Search Results page perform the following checks
    a. If the search results are not sorted by lowest price then sort them that way
    b. Design a strategy to confirm that search results are really sorted by lowest price
    c. On the flight summary pane at the top of this page, check that all the Flight 1, Flight 2 and Flight 3 information, including travel dates, is identical to the data entered on the Search page
5. Select the flight option with the lowest price such that all three legs of the trip are operated by the same airline, excluding United Airlines, and no leg has more than one stop over.
    a. If United Airlines happens to be the lowest priced option then skip it and select the next lowest priced airline that still fulfills all the other restrictions.
6. On the ”Review your trip” page perform following checks
    a. Verify that all the airports are the same as those on the Search Results page in Step #4
    b. Verify that the price is the same as displayed on the Search Results page for the flight selected in step #5
    c. Verify that the flight numbers are the same as those displayed on Search Results pages for the flight selected in step #5
    d. If any flights are not non-stop then perform flight number checks for all connecting flights in the trip
7. Click the “Continue Booking” button and on the resulting Payment page perform the same checks as step #6

    Optional for Senior level [This will test advance skills and overall understanding/approach of automation framework design] :

8. Script should be able to launch either Firefox or Chrome without any code change, but only by passing an appropriate parameter value from the command line or through a properties file.
9. Any performance improvement, scalability and code reusability notes will be a plus.
10. Demonstrate solution based on standard design patterns is a plus.
11. Handle the following conditions:

a. If United Airlines is the only airline that fulfills all the conditions in step #5 then go ahead and use it
b. If step #5 does not fetch any search results then randomly select 3 other airports and try again. Keep trying until search results are returned. Do this without code changes, recompiling the code or performing file I/O.

Additional provided instructions: (with my comments in red)

1. Use Selenium WebDriver and Java to automate this scenario
2. Incorporate use of any standard logging API to log useful information at various execution levels like info, warn, error, debug, etc…. I used log4j2 as my logging tool. Note: I will demonstrate the use of the TestNG testing framework and asserts in another project I'll post on GitHub.
3. Incorporate the use of appropriate collection types and iteration strategy to maximize performance. I made frequent use of ArrayList and List, especially when calling findElements(). Set is used when dealing with window handles.
4. Demonstrate the appropriate use of exception handling in your script. My "lower level" methods usually throw their exceptions which are eventually caught and processed in the "upper level" startHere() methods that each page has. There is no really good way to recover from most lower level exceptions in this program (such as failing to find a web element) so I log the problem and try to continue.
5. Demonstrate segregation of code and data. The initial 3 airports and the requested browser name are read in from a file
6. The conditions regarding travel dates and gaps should hold true for any day the script is executed. I'm using java.util.Calendar to help with this. See SearchPage.java, enterDepartureDates()
7. Script can be executed for any arbitrary airports without code change or recompiling the code. The initial three airport codes are read in from a file
8. Use an approach to determine whether the page transition is complete or loaded, or any intermediate interstitials disappeared, such as to reduce false failures. Implicit and explicit waits are used to achieve this. Implicit waits affect all elements while explicit waits are customized to specific elements.

My notes:

- I completed all 11 assignment steps and will refer to them in my code as "Tasks" when I implement that particular functionality. For example, Task 4a, Task 6d, etc...
- While locating elements, if an ID is available I use it because that is the faster way and should be unique (although I have seen poorly written HTML where it is not). If an ID does not exist then I usually use ClassName, CSS Selector or build an Xpath. I hardly ever use the Tag name to locate elements because they are rarely unique.
- My program is capable of running successive test cases while switching between different browsers in any order.
- The program's automation testing framework uses a Page Object Model which is a design pattern that creates an Object repository for web UI elements. By

"Object repository" I mean each web page of the application gets its own page class. Each page class finds the elements and contains the methods that perform actions on those elements for that specific web page. This design pattern improves code maintenance, promotes cleaner code, reduces redundancy, increases re-usability and helps with organization. I like it! The startHere methods in each page class allow you to visualize each step of the scenario. In this case there are 5 web pages of the Orbitz web app that will be tested.