Project 1 Test Design Document - Jackson Lee, Serena Zeng

| Java class/method name being tested: Date class, isValid() method | | | |
|---|---|---|---|
| Test case # | Requirement | Test Description and input data | Expected result/output |
| 1 | January, March, May, July, August, October and December, each has 31 days | ● Create dates in those months with <= 31 dates<br>● Test data:<br>○ 1/1/2023<br>○ 3/5/2023<br>○ 5/10/2023<br>○ 7/15/2023<br>○ 8/22/2023<br>○ 10/29/2023<br>○ 12/30/2023 | true |
| 2 | January, March, May, July, August, October and December, each has 31 days | ● Create date in those months with value > 31.<br>● Test data:<br>○ 1/32/2003<br>○ 3/32/2003<br>○ 5/32/2003<br>○ 7/32/2003<br>○ 8/32/2003<br>○ 10/32/2003<br>○ 12/32/2003 | false |
| 3 | April, June, September and November, each has 30 days | ● Create date in those months with date <= 30<br>● Test data:<br>○ 4/1/2023<br>○ 6/18/2023<br>○ 9/27/2023<br>○ 11/30/2023 | true |
| 4 | April, June, September and November, each has 30 days | ● Create date in those months with value > 30<br>● Test data:<br>○ 4/31/2003<br>○ 6/31/2003<br>○ 9/31/2003 | false |

| | | | |
|---|---|---|---|
| | | ○ 11/31/2003 | |
| 5 | Dates cannot have negative values | ● Dates cannot have negative or 0 values<br>● Test data:<br>○ -1/31/2003<br>○ 0/31/2003<br>○ 1/-31/2003<br>○ 1/0/2003<br>○ 1/31/-2003<br>○ 1/31/0 | false |
| 6 | Months have to be between 1 and 12. | ● Create date with month > 12<br>● Test data: 13/31/2003 | false |
| 7 | Years not divisible by 4 are not leap years | ● Create February 29 date matching description<br>● Test data: 2/29/2003 | false |
| 8 | Years divisible by 4 but not divisible by 100 are leap years | ● Create February 29 date matching description<br>● Test: 2/29/2008 | true |
| 9 | Years divisible by 100 but not divisible by 400 are leap years | ● Create February 29 date matching description<br>● Test: 2/29/1900 | false |
| 10 | Years divisible by 400 are leap years | ● Create February 29 date matching description<br>● Test: 2/29/2000 | true |

| Java class/method name being tested: Date class, compareTo() method | | | |
|---|---|---|---|
| Test case # | Requirement | Test Description and input data | Expected result/output |
| 1 | Member with alphabetically prior last name is placed in front. | ● Create member 1 with alphabetically prior last name to member2. Call member 1’s compareTo().<br>● “John Doe”, “Mary Lindsey” | negative |
| 2 | Member with alphabetically later last name is | ● Create member 1 with alphabetically later last | positive |

| | | | |
|---|---|---|---|
| | placed behind | name to member2. Call member 1’s compareTo().<br>● “Duke Ellington”, “Roy Brooks” | |
| 3 | Member with same last name and alphabetically prior first name is placed in front | ● Create member 1 and 2 with same last names, but member 1 with alphabetically prior first name. Call member 1’s compareTo()<br>● “Jane Doe”, “John Doe” | negative |
| 4 | Member with same last name and alphabetically later first name is placed behind | ● Create member 1 and 2 with same last names, but member 1 with alphabetically later first name. Call member 1’s compareTo()<br>● “John Doe”, “Jane Doe” | positive |
| 5 | Members with same first and last names can go in either order. | ● Create member 1 and 2 with same first and last names. Call member 1’s compareTo()<br>● “John Doe”, “John Doe” | zero |
| 6 | Cases do not matter in ordering | ● Create member 1 and 2 with same first and last name,, but different cases. Call member 1’s compareTo()<br>● “John Doe”, “john doe” | zero |
| 7 | Member1 with last name that is prefix of Member2's last name comes is placed behind | ● Create member 1 and 2 where member 2 has a last name that uses the last name of member 1 as a prefix. Call member 1’s compareTo()<br>● “John Doe”, “John Doee” | negative |