class name: Date Method Signature: public boolean isValid() //check if given date is a valid calendar date			
Test Case #	Requirement	Test Description and Input Data	Expected Result/Output
1	In a non-leap-year, February has a maximum of 28 days, but other months have a maximum of 30 days. The method should return false for any day in February after 28 for a non-leap-year.	• testDaysInFeb_NonLeap() • Create Date instance with invalid day, month, and year, and run isValid() on that • Test Data: “2/30/2023”	false
2	In a leap year, February has 29 days. This method should return true for the 29th of Feb of 2024, a leap year.	• testDaysInFeb_Leap() • Create Date instance with valid day, month, and year, and run isValid() on that • Test Data: “2/29/2024”	true
3	When entering the date, the user must follow the “mm/dd/year” format where year is the whole year. This method should return false for an incorrect date entered.	• testDateFormatting() • Create Date instance with incorrect year, but proper month and day, and run isValid() on that • Test Data: “05/08/24”	false
4	Some months have 30 days, some have 31 days. October has a maximum of 31 days. This method should return false for a date that exceeds the maximum amount of days, 31, for October.	• testMaxDaysInOct() • Create Date instance of October 32nd, 2023 to see if isValid() catches days that exceed a month’s limit • Test Data: “10/32/2023”	false
5	Leading zeros should not have an effect on the date itself, as long as there are still visible months, days, and years. This method should return true for a Date that has leading zeros in all fields.	• testLeadingZeros() • Create Date instance for a date with three 0’s in front of each parameter, and run isValid() to check its validity • Test Data: “00012/00010/0002023”	true

class name: Date Method Signature: public int checkIfWithinBounds() //check if given date does not exceed 6-month bound			
Test Case #	Requirement	Test Description and Input Data	Expected Result/Output
6	A Date cannot be valid if it exceeds the 6-month bound limit. When a user types a date, it must be between now and 6 months from now. This method returns an integer, 1, if the date entered is not within the bounds.	● testFutureDatesOutOfRange() ● Create Date instance with a date that is out of the 6-month range, and run checkIfWithinBounds() on that date; a return value of 1 signifies that the date is past the 6-month limit ● Test Data: “5/20/2024”	1

class name: Date Method Signature: public int checkIfInPast() //check if given date does not occur in the past			
Test Case #	Requirement	Test Description and Input Data	Expected Result/Output
7	A Date cannot be valid if it is before the current day. A user must type a date that is now or in the future. This method returns an integer 1 to denote that the date entered is in the past.	● testPastDateOutOfRange() ● Create Date instance with a date that is in the past; run the method checkIfInPast() on it; a return value of 1 signifies that the date is in the past ● Test Data: “1/16/2023”	1

class name: Event Method Signature: public String toString() //returns all event data in a single formatted line			
Test Case #	Requirement	Test Description and Input Data	Expected Result/Output
1	When taking in the duration of an event for the morning timeslot, the time itself will change from AM to PM. This	● testTimeChangeFromAMtoPM() ● Create Event instance by creating instances of its parameters like Date and Contact. The enums can go directly into the event. We run the	“succeeded”

class name: Event Method Signature: public String toString() //returns all event data in a single formatted line			
	method ensures that after adding a certain duration to the start time, the time will change from AM to PM. It will return a String saying "succeeded", meaning the actualOutput matches the expectedOutput and AM correctly switched to PM.	toString() on this method to see its output and whether the time changed correctly ● The expected output is a hard-coded line matching the output given in the directions, if this output matches the result of the toString(), we pass the test case ● Test Data: Date - "1/30/2024", Contact - (Department.CS, "cs@rutgers.edu"), Timeslot.MORNING, Location.ARC103, duration 120	

class name: Event Method Signature: public int compareTo(Event event) //compares the startTimes of events to decide which is earlier			
Test Case #	Requirement	Test Description and Input Data	Expected Result/Output
2	There may be a moment when two events are added with the same date, but different timeslots. In this case we must compare the Timeslots to ensure that the events are still ordered properly. This method returns -1 and prints "succeeded" if the two events created have the same integer output of -1, meaning the Timeslots are compared properly and event1 comes before event2.	● testCompareToForEqualDates() ● Create two Event instances with filled parameters of same Dates, but different Timeslots; run the compareTo() function with event1 being compared to event2, and our expected output is -1, meaning event1 comes before event2 ● Test Data: ○ Event 1: Date - "3/12/2024", Contact - Department.BAIT, "bait@rutgers.edu", Timeslot.MORNING, Location.ARC103, duration 60 ○ Event2: Date - "3/12/2024", Contact - Department.ITI, "iti@rutgers.edu", Timeslot.AFTERNOON, Location.HLL114, duration 30	-1 / "succeeded"

class name: Event Method Signature: public boolean equals() //checks if two given events are equal			
Test Case #	Requirement	Test Description and Input Data	Expected Result/Output
3	The equals method ensures if two created events share the same qualities, and therefore are equal. This method will return true if the two events are the exact same with no differences, and print “succeeded”.	● testTwoEventsEqual() ● Create two Events with the exact same parameters, and run the equals() method on them; if the expectedOutput, which is set to true, equals the actualOutput, then we know the two events are the same ● Test Data: ○ Event1/Event2: Date - “12/25/2023”, Contact - Department.CS/”cs@rutgers.edu”, Timeslot.EVENING, Location.BE_AUD, duration 60	true
4	The equals method ensures if two created events share the same qualities, and therefore are equal. This method will return false if the two events we are comparing are not equal to eachother, and do not share the same parameters.	● testTwoEventsNotEqual() ● Create two Events with different parameters, and run the equals() method on them; if the expectedOutput, which is set to false, equals the actualOutput, then we know the two events are different ● Test Data: ○ Event1: Date - “11/18/2023”, Contact - Department.ITI/”iti@rutgers.edu”, Timeslot.MORNING, Location.HLL114, duration 60 ○ Event2: Date - “10/22/2023”, Contact - Department.CS/”cs@rutgers.edu”, Timeslot.EVENING, Location.BE_AUD, duration 120	false