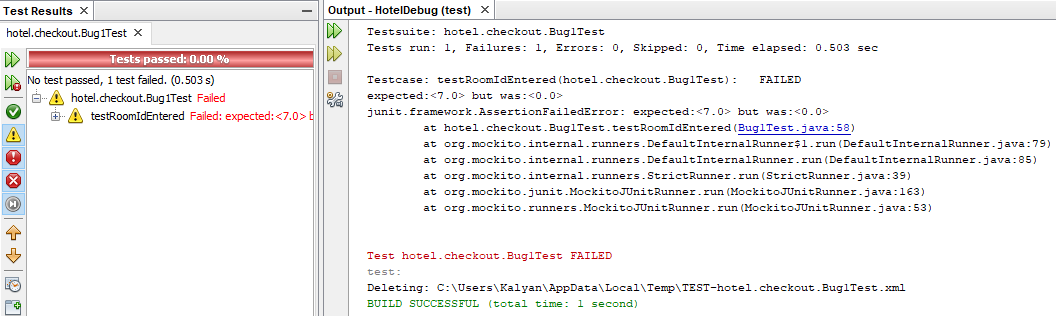
Bug Log

Bug 1: Service Charge $0.00 when Checking Out

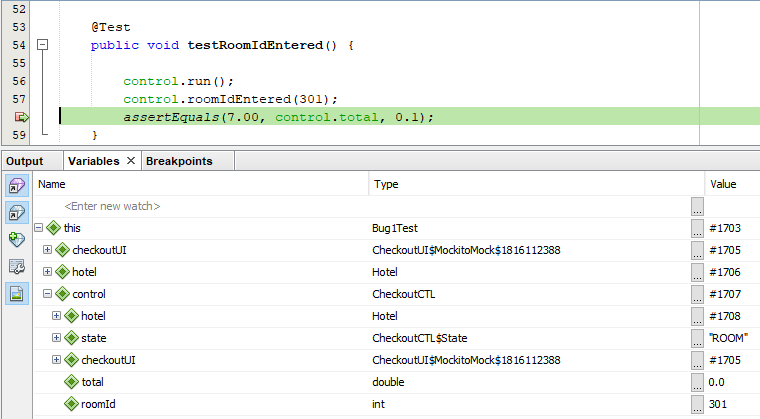
Since the bug occurs when checking out, after the user has entered the room ID, the bug should be able to be replicated by calling the roomIdEntered() method of the CheckoutCTL class.

Test method: testRoomIdEntered()

Output from Bug1Test.java demonstrating the bug:



Output from debugger showing control.total is an infected value (should be 7.00):



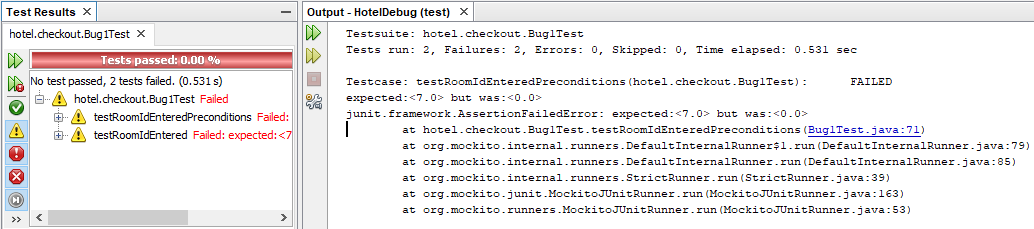
Hypothesis 1: The bug occurs before roomIdEntered() is called

Since the point of failure is in the roomIdEntered() method, the first test should demonstrate whether there is an infected value prior to calling this method.

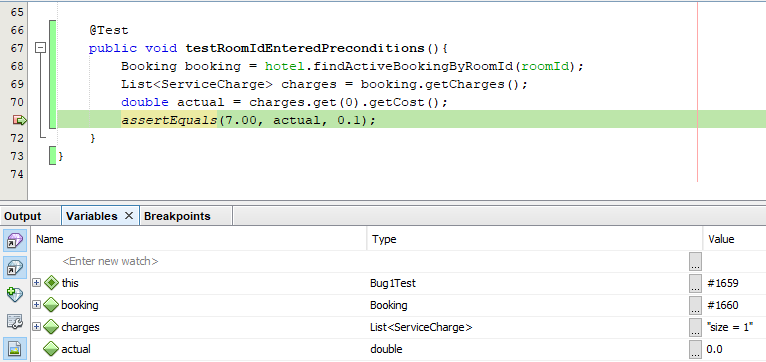
Test method: testRoomIdEnteredPreconditions()

Results: The hypothesis was proven correct, as the service charge was found to be an infected value before roomIdEntered() was called.

Output from Bug1Test.java:



Output from debugger showing that booking.charges contains an infected value (should be 7.00):



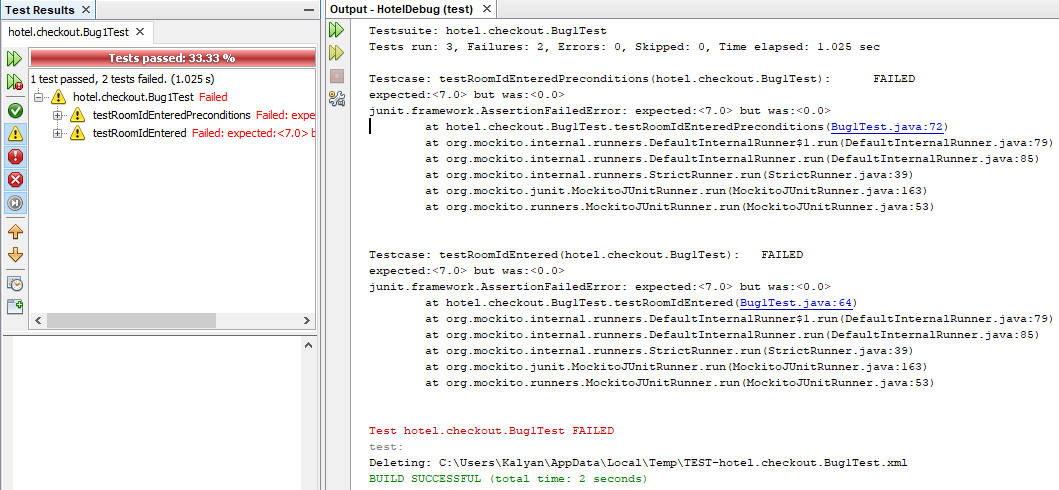
Hypothesis 2: The bug occurs in the constructor of the ServiceCharge class

Since the infected value is the cost field of a ServiceCharge object, the first dependence is the ServiceCharge() constructor which sets that value.

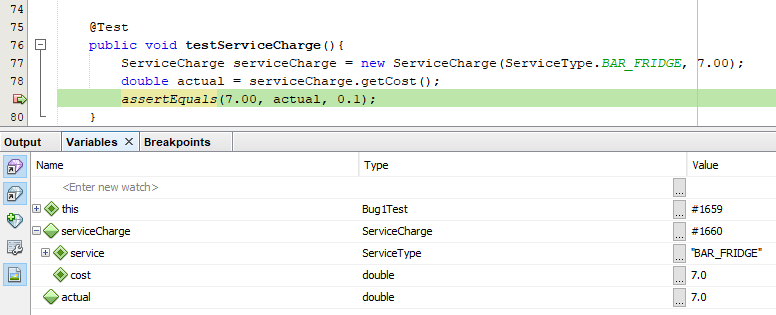
Test method: testServiceCharge()

Results: The test succeeded, showing that the bug is not in the constructor of the ServiceCharge class, thus disproving the hypothesis.

Output from Bug1Test.java:



Output from debugger showing no infected values:



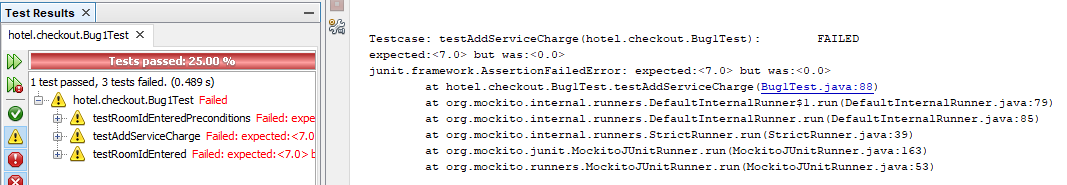
Hypothesis 3: The bug occurs in the addServiceCharge() method of the Booking class

This method, which creates the ServiceCharge using the constructor, is next in the chain of dependences.

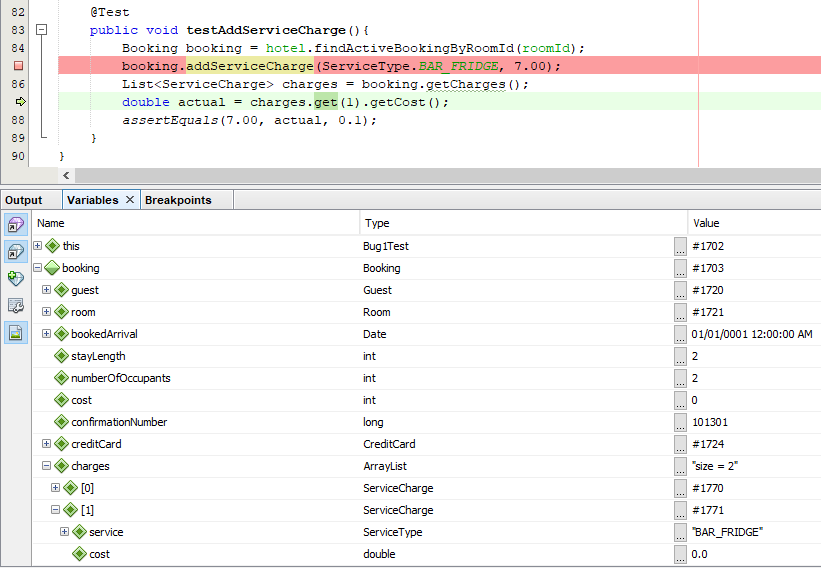
Test method: testAddServiceCharge()

Results: The value “cost”, which was provided directly in the test, became infected after calling Booking.addServiceCharge(), thereby proving the hypothesis correct.

Output from Bug1Test.java:



Output from debugger showing that booking.charges[1].cost is an infected value (should be 7.00):

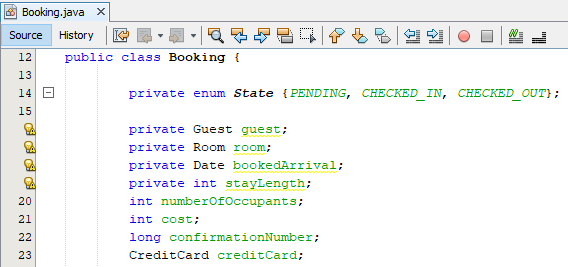


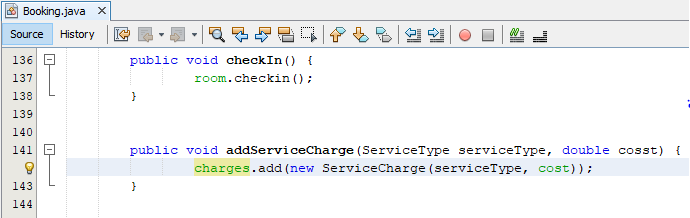
The Bug:

There was a typo in the parameters for Booking.addserviceCharge(): “cosst” instead of “cost”. Additionally, the Booking class had a data field “cost”, which was never initialized (and thus always had the default value of 0.0) and was being injected as a field of the ServiceCharge’s constructor.

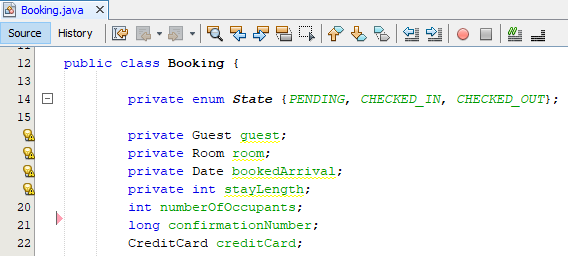
In order to fix this bug, I fixed the typo in Booking.addServiceCharge() and removed the data field “cost” from the Booking class.

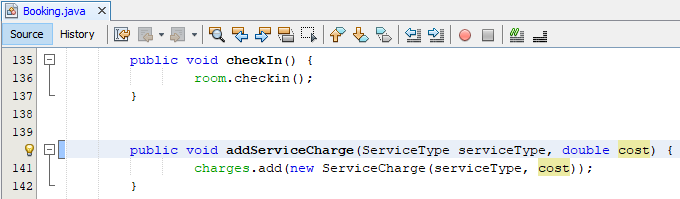
Before the fix:



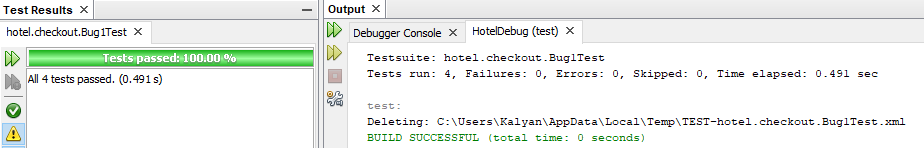


After the fix:





Output from Bug1Test.java showing all tests passed:

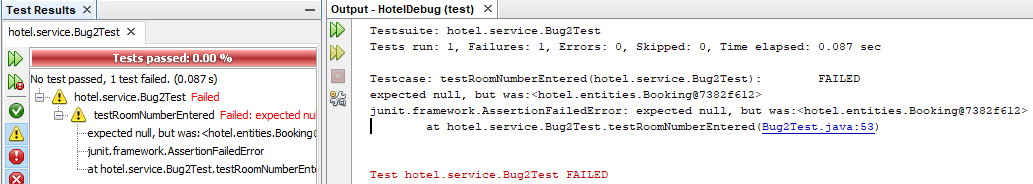


Bug 2: It is possible to charge a room for service after the guest has checked out

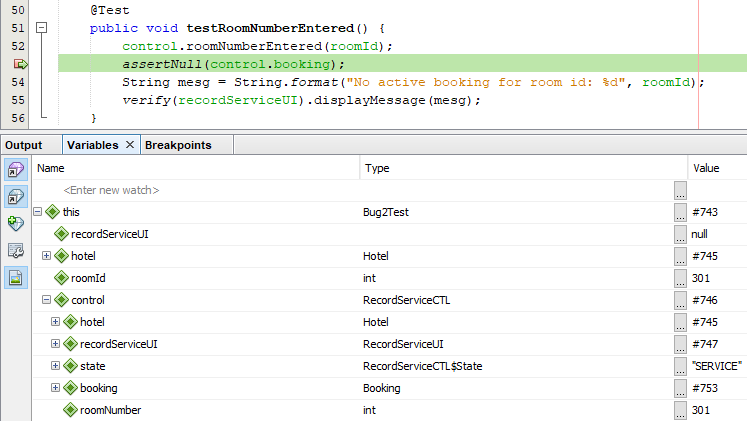
Since the bug occurs when recording a service, after the user has entered the room ID, the bug should be able to be replicated by calling the roomNumberEntered() method of the RecordServiceCTL class.

Test method: testRoomNumberEntered()

Output from Bug2Test.java demonstrating the bug:



Output from debugger showing that control.booking is an infected value (should be null):



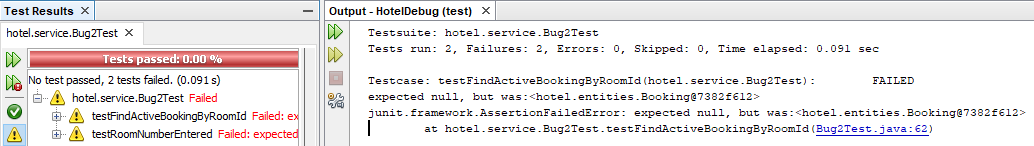
Hypothesis 1: The bug occurs before roomNumberEntered() is called

Since the point of failure is in the roomNumberEntered() method, the first test should demonstrate whether there is an infected value prior to calling this method.

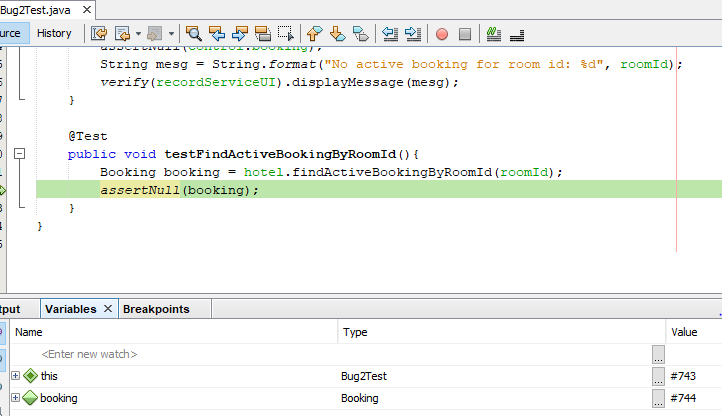
Test method: testFindActiveBookingByRoomId()

Results: The hypothesis was proven correct, since Hotel.findActiveBookingsByRoomId() returned an infected value

Output from Bug2Test.java:



Output from debugger showing that booking is an infected value (should be null):



Hypothesis 2: The bug occurs before findActiveBookingByRoomId() is called

Since this method returns an infected value, we should check whether an infected value is present before it is called.

Test method: testActiveBookingsByRoomId()