United International University

Group No : 03 (Xenon)

Title:Unit Testing-Parking Slot Booking With Intentional Defect.

Course Code: CSE 4495 **Section : B**

Course Name: Software Testing and Quality Testing Assurance

Submitted By

Md. Khademul Islam Nahin	**011221282**
Irfanuzzaman Montasir	**011221276**
Md. Abdullah Al Imran	**011221326**

Submitted To

Mobaswirul Islam

Lecturer Of CSE Dept.

United International University

Submission Date: 17 October, 2025

Wallert Class Test

Test ID	*Class.Method*	*Why this test*	*Verdict*	*Comment/Observations*
T1	initial_wallet_balance_check()	To verify after creating an object that whether the balance initializes as 0 or not.	**passed**	Everything is working fine. After creating object, balance initialized to 0 as expected
T2	adding_balance_then_check()	To verify that whether balance is equal or not after passing balance by parameter while creating an object.	**passed**	Balance correctly set after passing through the constructor. No issue found.
T3	addFundTest()	To verify that after adding balance by addfund method whether is accurate or not with my expected balance.	**passed**	Adding balance using addFund() worked perfectly and matched with expected result.
T4	deductFundsTest()	To verify that after deducting money from the main balance whether the main balance is equal to expected balance or not.	**passed**	The Deducting fund worked as expected. The remaining balance matched properly.
T5	deductFundsExceptionTest()	To verify whether the system throws the exception or not after deducting insufficient balance which is not present in the main balance.	**passed**	The system correctly throws exceptions when trying to deduct more than available.
T6	addFundsInvalidExceptionTest()	To verify whether the system throws the	**passed**	Negative and zero values not accepted. Exceptions handled properly.

		exception or not after adding invalid amount like negative balance or 0 balance which is not present in the main balance.		
T7	deductFundsNegativeAmountTest()	To verify whether the system throws the exception or not after deducting invalid amount like a negative balance.	**passed**	Negative deduction not allowed. Exception thrown as expected.
T8	deductZeroAmountTest()	To verify whether the system throws the exception or not after deducting invalid amount like Zero(0) balance.	**passed**	Deducting zero also throws exceptions correctly.
T9	TranferFundsTest1()	To verify whether the system throws the exception or not after transferring negative or Zero balance from a wallet to another wallet.	**passed**	For negative and zero transferring, exceptions worked perfectly.
T10	TranferFundsTest2()	To verify whether the system throws the exception or not after transferring insufficient balance on a current wallet to another wallet.	**passed**	The system handled insufficient balance and threw exceptions correctly.
T11	TranferFundsTest3()	Here verifying two things after transferring money from a current wallet to another wallet Whether the deducted money is equal to the	**passed**	Transfer worked correctly. Amount deducted and added as expected on both wallets.

		expected or not .Another test is after transferring money to another another ,in that account whether the balance is equal to the deducted balance of a current wallet or not.		
T12	negativeBalaceInitializationTest()	Here testing about a fact that after creating an object of wallet , if i put the negative amount on that object parameter and run the program whether it throws an exception or not.I fail it as intensional because its a defect.	***failed***	Objects created with negative amounts did not throw exceptions. Found as a defect.
T13	floatingPointTesting()	Here testing various floating type values like 0.55 + 0.4 and match with expected result to check whether it provides similar answers or not or there are tiny changes or not.	***failed***	Tiny decimal mismatch found because of double type variable.
T14	nullobjectTest()	To check whether transferring money from a current wallet to a null wallet object works as expected or not .	***failed***	Null wallet not handled. Throws runtime error during execution.
T15	maximumDoubleBalanceTest()	Here I'm testing if I reach the maximum double value and try to add some money. The system will throw exceptions or not.	***failed.***	The system failed to handle maximum double value and crashed.

Wallet Class Defects

Defect ID	*Class.Method*	*Description*	*Suggested Fix*
T1	negativeBalaceInitializationTest ()	It allows creating an object with negative balance.	There should be a checker in the constructor if the balance is greater than 0 or not .if it is less than zero it will throw an exception.
T2	floatingPointTesting()	Various floating values variation can cause tiny value changes because of taking variable declaration as double.	The BigDecimal can be used instead of double.Because in a monetary system tiny changes can hamper the system's transaction.
T3	nullobjectTest()	There is no checker for a null object ..so the system can fail anytime.	A null checker should be used in that case.
T4	maximumDoubleBalanceTest()	The code does not handle the maximum double value if i want to push the amount above the maximum value.	There should be an exception handler for this which will detect and handle this exception .

Vehicle Class Testing

Test ID	*Class.Method*	*Why this test*	*Verdict*	*Comment/Observations*

T1	getVehicleIdTest()	To verify that whether The vehicle id is equal to the expected result or not.	***passed***	Vehicle id matched correctly with expected value.
T2	getVehicleTypeTest()	To verify that whether the vehicle type is equal to the expected result or not.	***passed***	The vehicle type returned properly without issue.
T3	getWalletTest()	To verify whether the wallet object which is created under the vehicle object is equal to the expected object or not.	***passed***	Wallet object linked properly inside vehicle object.
T4	vehicleWalletBalanceTest()	To verify that vehicle wallet balance is equal to the expected balance or not.	***passed***	Wallet balance initialized and matched with expected result.
T5	tosrtingTest()	Verifying that to_string information matches with the expected information or not.	***passed***	Output matched correctly with expected format.
T6	VehicleNullWalletWithToStringTEst()	Verifying after creating a vehicle object and initializing with a null wallet and calling the to_string method will handle the exception or not.	***failed***	Null wallet not handled properly in toString(). The program fails in this case.
T7	nullVehicleTypeWithTosrtingTest()	Verifying after creating a vehicle object and initializing with a null vehicleType and calling the to_string method will handle the exception or not.	***failed***	Null vehicle type not handled, shows error during printing.
T8	NegativeBalanceTest()	Verifying after creating a vehicle object and initializing with a negative balance ,does the system handle the exception or not.	***failed***	Negative balance accepted during creation. It should have thrown an exception but it didn't.
T9	nullVehicleTypeTest2()	Verifying after creating a vehicle object and initializing with a null Vehicle Type,does the system handle the exception or not.	***failed***	Null vehicle type accepted while creating object, which is defect.

T10	NullVehicleWallet Test()	Verifying, after creating a vehicle object and initializing with a null Vehicle wallet,does the system handle the exception or not.	***failed***	Vehicle wallet null not handled. The system crashed at runtime.
T11	NullVehicleWallet GetterTest	After creating a vehicle object and initializing with a null Vehicle wallet,then call the getwallet method to see whether it has handled the exception or not.Also checking if it is null or not by using assertnull.	***failed***	Calling getWallet() on a null wallet object throws an error.
T12	nullVehicleTypeG etterTest	After creating a vehicle object and initializing with a null Vehicle type,then call the getvechicle type method to see whether it has handled the exception or not.Also checking its null or not using assert null.	***Failed***	Calling getVehicleType() when null is not handled, the system fails.

Wallet Class Defects

Defect ID	***Class.Method***	***Description***	***Suggested Fix***
T1	VehicleNullWalletWith ToStringTEst()	There is no checker in to_string method to detect an exception of a null value.	There should be an exception handler.
T2	nullVehicleTypeWithT osrtingTest()	There is no checker in to_string method to detect an exception of a null value.	There should be an exception handler.
T3	NegativeBalanceTest ()	There is no checker in the constructor if the vehicle object is	There should be an exception handler on the constructor for this case..

		initialized with negative balance.	
T4	nullVehicleTypeTest2( )	There is no checker in the constructor if the vehicle object is initialized with null vehicle type..	There should be an exception handler on the constructor for this case..
T5	NullVehicleWalletTest ()	There is no checker in the constructor if the vehicle object is initialized with null vehicle wallet..	There should be an exception handler on the constructor for this case..
T6	NullVehicleWalletGetterTest	When calling getWallet method, it can not handle the null exception properly.	There should be an exception handler on the constructor for this case..
T7	nullVehicleTypeGetterTest	When calling the vehicle type method, it can not handle the null exception properly.	There should be an exception handler on the constructor for this case..

Booking Class Testing

Test ID	*Class.Method*	*Why this test*	*Verdict*	*Comment/Observations*
T1	testBookIdTest()	To check whether the book id is equal to expected result or not.	***passed***	Booking ID matched correctly with expected.
T2	getVehicleTest()	To check after creating a vehicle object under the Booking object is null or not and also testing whether the vehicle object is equal to the expected object or not.	***passed***	Vehicle object inside booking not null and matched correctly.

T3	getParkingSlotsTest()	To check after creating a parking slot object under the Booking object is null or not and also testing whether the parkingslot object is equal to the expected object or not.	***passed***	The parking slot object matched perfectly.
T4	getStartTimeTest()	To verify that ,whether the actual time is equal to the expected result or not.	***passed***	Start time matched properly with expected value.
T5	getEndTime()	To verify that ,whether the actual time is equal to the expected result or not.	***passed***	End time matched properly.
T6	getAmountTest()	To verify whether the booking amount is expected or not.	***passed***	Booking amount returned correctly.
T7	getBookingStatusTest()	Checking actual booking status is equal to the expected or not.	***passed***	Booking status “ACTIVE” verified successfully.
T8	completeBookingTest()	After clicking the complete book method to test whether it works as expected or not.	***passed***	After completing booking, status changed to COMPLETED successfully.
T9	mixtesting()	Here testing ,1st of all trying to cancel the book method after that, trying complete booking methods to see whether the status change or not ,but the changes should not occur in this case a defect ..	***passed***	System allowed to complete a booking after cancellation which is wrong, defect found.
T10	toStringTesting()	To verify to_string works as expected or not.	***passed***	toString() output matched fine with expected string.
T11	nullTestingOfParkingSlotObject()	After passing an object of the parking slot in the booking object to see whether it handled the exception or not.	***failed.***	The system failed to handle null parking slot objects. Throws an exception.
T12	nullVehicleObjectTest	After passing an object of the Vehicle in the booking object to see	***Failed***	System failed to handle null vehicle object.

		whether it handled the exception or not.		
T13	negativeAmountTest()	After passing a negative amount on the booking object to see whether it handled the exception or not.	***failed***	A negative amount accepted should not be allowed.
T14	TimingTest()	To verify whether selecting a time range like end time is before the start time or not.	***Passed***	For reversed time range, system not validating properly.
T15	nullTimingTest()	After passing null time value in the object to see whether it handles the exception or not.	***failed***	Null timing values caused exceptions, not handled correctly.
T16	zeroAmountTesting()	Testing whether initializing the booking object with 0 amount throws an exception or not.	***Failed.***	Zero amount accepted, not properly validated.
T17	TimingTest2()	To verify whether the same time conflict or not but it does not conflict.	***Passed***	The same start and end time didn't conflict, but should be checked strictly.

Booking Class Defects

Defect ID	***Class.Method***	***Description***	***Suggested Fix***
T1	nullTestingOfParkingSlotObject()	The null object of the parking slot can be passed through the Booking object while initialization of the object,which is not handled by the code.	There should be a checker of exceptions in the constructor of the Booking class ,so that it can detect null objects.
T2	nullVehicleObjectTest	The null object of the vehicle can be passed through the Booking object while initialization of the	There should be a checker of exceptions in the constructor of the Booking class ,so that it can detect null objects.

		object.which is not handled by the code.	
T3	negativeAmountTest()	The negative value can be passed through the Booking object while initialization of the object,which is not handled by the code.	There should be a checker of negative value in the constructor of the Booking class ,so that it can detect null objects.
T4	nullTimingTest()	The null time value can be passed through the Booking object while initialization of the object,which is not handled by the code.	There should be a checker of exception in the constructor of the Booking class ,so that it can detect null timing value.
T5	zeroAmountTesting()	The zero value can be passed through the Booking object while initialization of the object,which is not handled by the code.	There should be a checker of exception in the constructor of the Booking class ,so that it can detect null timing value.
T6	mixtesting()	It is a big defect case. After running the cancel booking method ,if i run again the complete booking method it should trigger an error because after canceling a booking i cant complete that booking .Which is the big defect in this class..	For fixing this there should be an exception thrown in the complete booking method so that when it will check the status of the booking whether it will cancel or not if it “cancel” it can throw an exception immediately.
T7	TimingTest2()	Another big defect.Same time can be selected here,there is no	While creating a booking object it will check whether the end time is greater than start

		conflict between the start time and end time.	time or not.After that i will pass the timing value to this object.

Parking Slot Testing

Test ID	*Class.Method*	*Why this test*	*Verdict*	*Comment/Observations*
T1	slotIDTest()	To check whether the slot id is equal to expected result or not	***passed***	Slot ID returned as expected.
T2	slotTypeTest()	To check whether the slot type is equal to expected result or not	***passed***	Slot type matched with expected one.
T3	isActiveTest()	To check whether the isActive variable is activated or not after creating an object.	***passed***	Slot active after creation, works fine.
T4	getWalletTest()	To check whether the wallet is not null after creating an object or not.	***passed***	Wallet object not null, successfully created.
T5	getBookingsTest( )	To check whether the booking is created or not after creating an object of parkingslot..	***passed***	Booking list not null, initialized properly.
T6	isCompatible_IFis ActiveTest()	To verify that isCompatible() method correctly works when the slot is active.	***passed***	When slot active, isCompatible() returned true correctly.

T7	deactivateTest()	To verify that after deactivating the slot, isCompatible() returns false even for valid types.	***passed***	After deactivation, the slot is no longer compatible as expected.
T8	vehicleCompatibilityTypeTest() **Note:**In this case we only test 1 or 2 cases.1 or 2 cases would be sufficient for this.	To verify compatibility of various vehicle types with different slot types according to parking rules.	***passed***	Vehicle and slot compatibility rules worked fine for valid types.
T9	isCompatibleTesting_foroverlap_forNotOverlap()	To verify isAvailable() method properly detects overlapping and non-overlapping booking times.	***passed***	Overlap detection worked fine. Non-overlapping times passed successfully.
T10	invalidParkingSlotID()	To check whether a parking slot is valid or not.	***failed***	Invalid slot ID accepted without error. Defect found.
T11	startAndEndTimeNUllTesting()	To test whether null start and end time will work with multiple valid booking times or not.	***failed.***	Null start and end times not handled. The system throws exceptions.
T12	incorrectTimeRangeTest()	Here testing whether the same start and end time like 18 and 18 and start time is greater than end time(start > end)return false or not.	***passed***	Same or reverse times correctly returned false.
T13	nullVehicleTypeCompatibilityTest()	If we pass vehicle type as null to check whether it will throw an exception or not.	***passed***	Null vehicle type handled properly by exception.
T14	timesNUllTestWithoutAnyBookingTimes()	To check using start and end null time will successfully complete the book or not. **Note:**There is no booking exist in the system right now .we	***passed.***	Null timing values accepted when no booking exists. Should throw an exception but didn't.

		are only testing it as an initial run.		
T15	parkingslotTypeNUllTest()	To verify whether it detects the null parking slot and throws an exception or not.	***failed***	Null slot type accepted, no exception thrown. Defect.
T16	BookingFrom_curentday_to_another_day_test()	To check whether I can book from today to the next day or not.	***passed.***	Cross-day booking worked perfectly.

Parking Slot Defects

Defect ID	***Class.Method***	***Description***	***Suggested Fix***
T1	startAndEndTimeNUllTesting()	If we test this considering start and end time as null with other existing booking valid times it will give us an exception by the system but the code did not handle it by throwing an exception .It could hamper the run time of the code.	Implementing exception handling will be beneficial in this case and will prevent run time errors.
T2	invalidParkingSlotID()	Invalid parking slot id can be initialized. It dont check whether it is integer or not.	In the parking slot constructor there should be a checker which will check where the value is valid or not. If not it

			will immediately throw an exception.
T3	timesNUllTestWithoutAnyBookingTimes()	If i book a slot with null start and end time where there is no existing booking left on the system it will work perfectly but it should not work because it contains null timing value.	In isAvailable() there should be a checker which will check where the value is valid or null or not. If the Timing values are null it immediately throws an exception.
T4	parkingslotTypeNUllTest()	If can't handle the null slot type value while initializing the constructor.	There should be a checker on the constructor which will check whether it is null or not and immediately throws an exception.

Parking System Testing

Test ID	*Class.Method*	*Why this test*	*Verdict*	*Comment/Observations*
T1	testGetInstanceSingleton()	To confirm the singleton pattern correctly provides the same instance.	***passed***	The method confirms the singleton design pattern.
T2	testSystemBalance_StartsAtZero()	To verify that the singleton instance reset in the test setup works correctly. This ensures each test starts with a fresh system wallet.	***passed***	Confirms test isolation is working; the system balance is 0.0 at the start of every test

T3	testGetAvailableParkingSlots_ForCar()	To validate that the system correctly identifies all compatible and available slots for a vehicle type (CAR).	***passed***	The method correctly filters slots based on compatibility rules.
T4	testGetAvailableSlots_ExcludesBookedSlots()	To ensure that a slot that has been booked is correctly excluded from the list of available slots for an overlapping time.	***passed***	The availability check correctly identifies time conflicts.
T5	testBook_Successful()	A valid booking should succeed, as we change status to ACTIVE, and transfer funds correctly.	***passed***	The core booking logic and fund transfer from vehicle to system work as expected.
T6	testBook_FailsWithInsufficientFunds()	To test the insufficient funds validation, ensuring a booking is rejected if the vehicle's wallet cannot cover the cost.	***passed***	The system correctly throws an InsufficientFundsException.
T7	testBook_FailsForOverlappingTime()	To confirm that the system prevents double-booking a slot by rejecting a request for an overlapping time window.	***passed***	The system correctly throws an IllegalArgumentException for time conflicts.
T8	testBook_ZeroDurationThrows()	To test that a booking with a zero-hour duration (start time equals end time) is rejected.	***passed***	The system correctly throws an IllegalBookingTimeException.
T9	testBook_FailsIfSlotIsInactive()	To verify that the system prevents booking of a parking slot that has been marked as inactive.	***passed***	A booking attempt on a deactivated slot correctly throws an IllegalArgumentException.
T10	testPricing_FractionalHours()	To test that the pricing logic correctly truncates fractional hours (1.5 hours is billed as 1 hour).	***passed***	The pricing calculation correctly uses Duration.toHours().

T11	testPricing_UnderOneHourIsFree()	To confirm a booking of less than 1 hour (e.g., 59 minutes) results in a cost of zero.	***failed***	The test fails because a zero-cost booking causes an InvalidAmountException in the Wallet.
T12	testPricing_DefectForMicrocar()	To check if a MICROCAR is priced correctly. It is expected to fail.	***failed***	The test fails because the system applies the default CAR rate (1.0) instead of a specific rate for MICROCAR.
T13	testCompleteBooking_Successful()	the booking status changes to COMPLETED and 80% of the fee is paid to the slot owner.	***passed***	The fund settlement from system to slot works as specified.
T14	testCompleteBooking_NoDoublePayment()	To ensure that completing the same booking twice does not result in a double payment to the slot owner.	***failed***	The test fails with an InsufficientFundsException, revealing a deeper issue with state handling on second completion.
T15	testCancelBooking_Successful()	The booking status becomes CANCELLED and a 90% refund is issued to the vehicle. It is the standard cancellation process	***pass***	The refund logic works as specified.
T16	testCompleteBooking_DefectWhenAlreadyCancelled()	To check system behavior when trying to complete a booking that is already CANCELLED. It is expected to fail.	***failed***	The test fails with an InsufficientFundsException because the system incorrectly attempts a payout.
T17	testCancelBooking_DefectWhenAlreadyCompleted()	To check system behavior when trying to cancel a booking that is already COMPLETED. It is expected to fail.	***failed***	The test fails with an InsufficientFundsException because the system incorrectly attempts a refund.

Parking System Defects

Defect ID	*Class.Method*	*Description*	*Suggested Fix*
T1	ParkingSystem.calculatePrice()	The switch statement that determines the vehicle's rate multiplier is missing a case for MICROCAR. This causes it to use the default rate of 1.0 (same as a CAR), leading to incorrect pricing.	Add a new case MICROCAR: block to the switch statement in calculatePrice() with the appropriate rate multiplier.
T2	ParkingSystem.completeBooking()	The method does not check the booking's current status. It allows a CANCELLED booking to be completed, which incorrectly attempts an 80% payout to the slot owner for a service that was never provided.	Add a validation check at the beginning of the method, such as if (booking.getBookingStatus() != BookingStatus.ACTIVE), and throw an exception if the booking is not active.
T3	ParkingSystem.cancelBooking()	The method does not check the booking's current status. It allows a COMPLETED booking to be cancelled, which incorrectly attempts a 90% refund to the vehicle owner even after the service has been used and paid for.	Add a validation check at the beginning of the method, such as if (booking.getBookingStatus() != BookingStatus.ACTIVE), and throw an exception if the booking is not active.
T4	ParkingSystem.book()	The system does not handle zero-cost bookings. When a booking costs 0.0 (e.g., duration < 1 hour), it attempts to transfer 0.0, which the Wallet class rejects by throwing an InvalidAmountException.	In the book method, add a condition to bypass the vehicle.getWallet().transferFunds() call if the calculated amount is zero.