Code Exercise: Writing Unit Tests

Mobile Development Frameworks III

For this assignment, you will be given a simple application that has some utility classes that need tested.

You will be writing both local and instrumented unit tests that ensure all code functions as required.

You will need to reference the test cases listed in the associated activity to complete this assignment.

Code Structure and Efficiency	10%	Excellent (100%)	Good (80%)	Fair (45%)	Poor (0%)
This is a measure of how well your code is structured and how efficiently it runs. It is not expected that you will get a 100% in this category at all times. If you read your feedback thoroughly and apply it after each submission, your grade in this category will improve over time.		App meets all "Good" requirements.	Meets all positive "Fair" requirements.	Does not suffer from any negatives listed in the "Poor" column.	Application is missing required elements.
		Application contains proper separation of fragment and activity code and uses interfaces to communicate between the fragment and activity.	Does not suffer from any negatives listed in the "Fair" or "Poor" columns.	Application does not use fragments for all screens.	Application is non-functional.
		Application contains no efficiency issues.	Application uses fragments for all screens.	Application contains more than minor efficiency issues.	

PersonConversionUtil	10%	Excellent (100%)	Good (80%)	Fair (45%)	Poor (0%)
The PersonConversionUtil class should be exercised with unit tests to ensure all proper functionality. Test inputs should be created based on the provided testing spec. All tests should pass. These tests will require a dependency that contains JSON so that your tests can utilize the JSON classes. Any dependency that contains the JSON classes may be used for this.		App meets all "Good" requirements.	App meets all positive "Fair" requirements.	Does not suffer from any negatives listed in the "Poor" column.	Code is not unit tested.
		Unit tests are contained in their own class that is properly named for the class that is being tested.	App does not suffer from any negatives listed in the Fair or Poor columns.	Unit test methods are present for all PersonConversionUtil methods.	
		Unit tests are named to be specific as to what they're testing.	All unit tests pass when run.	One or more unit tests fail or do not account for the given test cases.	
			All unit tests exercise the given test cases in the Activity document..		
			All unit tests exist in the proper project used for local testing.		

PersonFormatUtil	20%	Excellent (100%)	Good (80%)	Fair (45%)	Poor (0%)
The PersonFormatUtil class should be exercised with unit tests to ensure all proper functionality. Test inputs are provided in the associated activity as are the expected outputs. Several methods in the PersonFormatUtil class need to have their code filled out to match the expected inputs and outputs. All unit tests should pass when run.		App meets all "Good" requirements.	App meets all positive "Fair" requirements.	Does not suffer from any negatives listed in the "Poor" column.	Code is not filled out.
		Unit tests are contained in their own class that is properly named for the class that is being tested.	App does not suffer from any negatives listed in the Fair or Poor columns.	PersonFormatUtil methods are filled out based on the labeled TODOs.	
		Unit tests are named to be specific as to what they're testing.	All unit tests pass when run.	PersonFormatUtil methods perform the functions described in the TODOs.	
			All unit tests exercise the given test cases in the Activity document..	Code is not unit tested.	
			All unit tests exist in the proper project used for local testing.	One or more unit tests fail or do not account for the given test cases.	

PersonStorageUtil	25%	Excellent (100%)	Good (80%)	Fair (45%)	Poor (0%)
The PersonStorageUtil class should be exercised with instrumented unit/integration tests to ensure proper functionality. You will need to mock the Context class to pull a mocked file location as to not interfere with the application's production data.		App meets all "Good" requirements.	App meets all positive "Fair" requirements.	Does not suffer from any negatives listed in the "Poor" column.	Code is not unit tested.
		Unit tests are contained in their own class that is properly named for the class that is being tested.	App does not suffer from any negatives listed in the Fair or Poor columns.	Unit tests are present in the proper project used for instrumented tests.	
		Unit tests are named to be specific as to what they're testing.	All classes are properly mocked so that no changes are made to the deployed application.	One or more unit tests fail or do not account for the given test cases.	
			All unit tests pass when run.	Objects are not properly mocked resulting in changes to the deployed application.	
			All unit tests exercise the given test cases in the Activity document.		

PreferenceUtil	35%	Excellent (100%)	Good (80%)	Fair (45%)	Poor (0%)
The PreferenceUtil class should be exercised with instrumented unit tests to ensure proper functionality. You will need to mock the Context and SharedPreferences classes to complete these tests. The mocked Context should return a mocked SharedPreferences. The mocked SharedPreferences		App meets all "Good" requirements.	App meets all positive "Fair" requirements.	Does not suffer from any negatives listed in the "Poor" column.	Code is not unit tested.
		Unit tests are contained in their own class that is properly named for the class that is being tested.	App does not suffer from any negatives listed in the Fair or Poor columns.	Unit tests are present in the proper project used for instrumented tests.	

SharedPreferences. The mocked SharedPreferences should return the proper values based on the test cases listed to ensure the proper outputs.		Unit tests are named to be specific as to what they're testing.	All classes are properly mocked so that no changes are made to the deployed application.	One or more unit tests fail or do not account for the given test cases.	
			All unit tests pass when run.	Objects are not properly mocked resulting in changes to the deployed application.	
			All unit tests exercise the given test cases in the Activity document..		
	100%				