Rubric Detail

*A rubric lists grading criteria that coordinators use to evaluate student work. Your coordinator linked a rubric to this item and made it available to you. Select **Grid View** or **List View** to change the rubric's layout.*

Name: **Assignment 2 CRA**

Exit

Grid View List View

	Poor (zero marks)	**Fair (half marks)**	**Good (full marks)**
Source Control	**0** (0%) Git log is missing or does not demonstrate progressive development. Git commit messages do not convey useful information in appropriate levels of detail covering the changes made in the commit.	**1** (2.86%) Git log is present and demonstrates progressive development, but commits are not at the appropriate level of granularity (e.g. too many changes or different kinds of changes in one commit). Some git commit messages are useful, or all git commit messages are partially useful.	**2** (5.71%) Git log is present and demonstrates progressive development. Commits contain clearly defined features or changes and the commit messages are useful and contain appropriate levels of detail.
Collaboration	**0** (0%) Git log is missing or does not show evidence of collaboration between two people on this project. All or	**1** (2.86%) Git log is present and shows evidence of collaboration between two people on this project.	**2** (5.71%) Git log is present and shows evidence of collaboration between two people on this project. There is a

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	nearly all commits are from one person. Authorship of commits and authorship of code as described in the Javadoc comments do not match up.	Authorship of commits and authorship of code as described in the Javadoc comments often line up.	mix of contributions from both participants and the order of these contributions shows that Test-Driven Development was used on this project (e.g. tests before implementations). Authorship of commits and authorship of code as described in the Javadoc comments line up.
Program Architecture Report	**0** (0%) The program architecture report is either missing, unintelligible, or does not convey enough detail to assess the architectural design of the program.	**.75** (2.14%) The program architecture report describes the classes implemented and how they fit together, but without illustrating how the software engineering principles of abstraction and polymorphism are used.	**1.5** (4.29%) The report provides a clear and accurate description of your program's architecture, describing the classes that are used, how they interact, as well as referencing object-oriented design concepts such as polymorphism and abstraction. The language used in the report is precise, detail-oriented and uses appropriate technical report nomenclature.
GUI Test Report	**0** (0%)	**.75** (2.14%)	**1.5** (4.29%)

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	The GUI test report is either missing or does not provide anything close to a reasonable level of coverage of the application's required functionality.	The GUI test report is present and covers the required functionality, but does not provide a thorough coverage of exceptional behaviour. Screenshots are either not used or are used sparingly where including more would make the document more illustrative.	The GUI test report presents a thorough coverage of the application, testing all of the required functionality, with an adequate number of screenshots used to clearly illustrate program behaviour. Both expected and exceptional behaviour is comprehensively covered by this test report.
Javadoc	**0** (0%) The generated Javadoc was not submitted, or Javadoc-style comments are not present in the source code, or there are entire classes or a substantial number of public methods missing from the documentation.	**1** (2.86%) The code contains Javadoc-style comments and HTML documentation was generated from this and submitted with the assignment. Every class and every method is documented, with minor oversights tolerated.	**2** (5.71%) The code contains Javadoc-style comments and HTML documentation was generated from this and submitted with the assignment. Every class and every method is comprehensively documented and fully describes the class API such that the documentation could be used by other developers wishing to use these classes in their own program.
	0 (0%)	**1** (2.86%)	**2** (5.71%)

	Poor (zero marks)	Fair (half marks)	Good (full marks)
User interface functionality	The submitted program does not possess a graphical user interface, or does not use the Java Swing toolkit. If a Swing GUI is present, it has serious problems and is close to unusable, or is missing important functionality.	The submitted program uses the Java Swing toolkit to provide a modern graphical user interface. The interface allows the user to complete all of the tasks specified by the client in the assignment specification.	The submitted program uses the Java Swing toolkit to provide a modern graphical user interface. The interface allows the user to complete all of the tasks specified by the client in the assignment specification. Furthermore, the interface is able to gracefully handle exceptional behaviour, showing appropriate user-facing error messages in this case that will allow the user to respond to them and recover.
User interface design	**0** (0%) The submitted program does not possess a graphical user interface, or the user interface is so poorly designed it impacts usability.	**1** (2.86%) The submitted program uses the Java Swing toolkit to provide a modern graphical user interface that is sufficiently well designed to not interfere in the use of the program.	**2** (5.71%) The submitted program uses the Java Swing toolkit to provide a high quality and highly polished graphical user interface that meets client objectives, presents data in a clear and informative fashion and uses appropriate UI widgets to present

	Poor (zero marks)	Fair (half marks)	Good (full marks)
			functionality in a clear, aesthetically pleasing fashion.
Unit Testing - Item	**0** (0%) Unit tests for the Item class are either not present or woefully inadequate, failing to cover all expected behaviour or failing to cover any exceptional behaviour. There is insufficient test coverage for this class to correctly implement this class within the Test-Driven Development paradigm.	**.25** (.71%) Unit tests for the Item class are present and cover all expected behaviour. There is some coverage of exceptional behaviour. Enough functionality is covered by these tests to enable implementing this class within the Test-Driven Development paradigm.	**.5** (1.43%) Unit tests for the Item class thoroughly cover all behaviour, both expected and unexpected, wherever such behaviour is amenable to unit testing. The unit testing is thorough enough that it effectively acts as a secondary API for the class, allowing that class to be completely implemented within the Test-Driven Development paradigm.
Unit Testing - Stock	**0** (0%) Unit tests for the Stock class are either not present or woefully inadequate, failing to cover all expected behaviour or failing to cover any exceptional behaviour. There is insufficient test coverage for this class to correctly	**.5** (1.43%) Unit tests for the Stock class are present and cover all expected behaviour, wherever such behaviour is amenable to unit testing. There is some coverage of exceptional behaviour. Enough functionality is covered by these	**1** (2.86%) Unit tests for the Stock class thoroughly cover all behaviour, both expected and unexpected, wherever such behaviour is amenable to unit testing. The unit testing is thorough enough that it effectively acts as a secondary API for the class, allowing

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	implement this class within the Test-Driven Development paradigm.	tests to enable implementing this class within the Test-Driven Development paradigm.	that class to be completely implemented within the Test-Driven Development paradigm.
Unit Testing - Truck	**0** (0%) Unit tests for the Truck, Ordinary Truck and Refrigerated Truck classes are either not present or woefully inadequate, failing to cover all expected behaviour or failing to cover any exceptional behaviour. There is insufficient test coverage for these classes to correctly implement these classes within the Test-Driven Development paradigm.	**.75** (2.14%) Unit tests for the Truck, Ordinary Truck and Refrigerated Truck classes are present and cover all expected behaviour, wherever such behaviour is amenable to unit testing. There is some coverage of exceptional behaviour. Enough functionality is covered by these tests to enable implementing these classes within the Test-Driven Development paradigm.	**1.5** (4.29%) Unit tests for the Truck, Ordinary Truck and Refrigerated Truck classes thoroughly cover all behaviour, both expected and unexpected, wherever such behaviour is amenable to unit testing. The unit testing is thorough enough that it effectively acts as a secondary API for the classes, allowing those classes to be completely implemented within the Test-Driven Development paradigm.
Unit Testing - Manifest	**0** (0%) Unit tests for the Manifest class are either not present or woefully inadequate, failing to cover all expected behaviour or	**1** (2.86%) Unit tests for the Manifest class are present and cover all expected behaviour, wherever such behaviour is amenable to unit	**2** (5.71%) Unit tests for the Manifest class thoroughly cover all behaviour, both expected and unexpected, wherever such behaviour is amenable to unit

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	failing to cover any exceptional behaviour. There is insufficient test coverage for this class to correctly implement this class within the Test-Driven Development paradigm.	testing. There is some coverage of exceptional behaviour. Enough functionality is covered by these tests to enable implementing this class within the Test-Driven Development paradigm.	testing. The unit testing is thorough enough that it effectively acts as a secondary API for the class, allowing that class to be completely implemented within the Test-Driven Development paradigm.
Unit Testing - Store	**0** (0%) Unit tests for the Store class are either not present or woefully inadequate, failing to cover all expected behaviour or failing to cover any exceptional behaviour. There is insufficient test coverage for this class to correctly implement this class within the Test-Driven Development paradigm.	**.5** (1.43%) Unit tests for the Store class are present and cover all expected behaviour, wherever such behaviour is amenable to unit testing. There is some coverage of exceptional behaviour. Enough functionality is covered by these tests to enable implementing this class within the Test-Driven Development paradigm.	**1** (2.86%) Unit tests for the Store class thoroughly cover all behaviour, both expected and unexpected, wherever such behaviour is amenable to unit testing. The unit testing is thorough enough that it effectively acts as a secondary API for the class, allowing that class to be completely implemented within the Test-Driven Development paradigm.
Unit Testing Separation of Concerns	**0** (0%) There is no evidence that the team behind this submission went to the trouble of making sure that	**.5** (1.43%) There is evidence that the team behind this submission went to the effort of making sure a	**1** (2.86%) There is evidence that the team behind this submission went to the effort of making sure a

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	the person who wrote the code for a class and the person who wrote the unit tests for that same class aren't the same person. Alternatively, the documentation and/or Git logs provide insufficient information for the marker to determine who wrote classes and unit tests.	different person worked on the unit tests and the implementation of each class.	different person worked on the unit tests and the implementation of each class, and managed this along with an equitable distribution of workload to ensure that one programmer wasn't doing all of the unit testing and another programmer wasn't doing all of the implementations.
OOP and Design Patterns	**0** (0%) The authors of this submission have shown limited understanding of object-oriented principles when designing classes and the way those classes interact with each other. Design patterns are either not used at all or not used correctly. The submitted code shows examples of poor object-oriented programming practice, such as classes exposing implementation details in their	**1** (2.86%) The authors of this submission have shown some understanding of object-oriented principles when designing classes and the way those classes interact with each other. Design patterns are used appropriately. The submitted code generally shows evidence of acceptable object-oriented programming practice, although it could be improved, further encapsulated	**2** (5.71%) The authors of this submission have shown a strong understanding of the principles of object-oriented design and development when designing the classes and class interactions used in this assignment. Good object-oriented programming practice (such as abstraction and encapsulation) is used wherever appropriate and the classes' external interfaces does not require

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	interfaces or omitting important refactorings in the classes' external interfaces.	and refactored etc.	further refactoring.
Code Quality - Item	**0** (0%) The Item class was either not implemented or implemented very poorly, with badly-chosen variable, field and method names, poorly structured code with either too much or too little use of whitespace and commenting, large amounts of code duplication and/or magic numbers. If the comments are present, they may even be outright incorrect or deceptive. The code is cryptic and difficult to read and debug. Method names are not self-describing and understanding their functionality often requires reading the code itself.	**.25** (.71%) The code used to implement the Item class is in a reasonable state, with mostly acceptable-to-good use of variable, field and method names. The code is fairly well structured with a mostly appropriate level of whitespace and commenting. Comments are accurate and up-to-date. Code duplication and other antipatterns in need of source-level refactoring are kept to a minimum. The code is not too difficult to read, and method names and variables are mostly self-describing.	**.5** (1.43%) The Item class was implemented very competently, with well-chosen identifier names and excellently structured code that is self-documenting, includes comments and whitespace in appropriate quantities to make the code even clearer. Comments are accurate, up-to-date and describe the logic behind the code, not merely what the code is doing. Source-level refactorings are not needed to improve the code.

	Poor (zero marks)	Fair (half marks)	Good (full marks)
Code Quality - Stock	**0** (0%) The Stock class was either not implemented or implemented very poorly, with badly-chosen variable, field and method names, poorly structured code with either too much or too little use of whitespace and commenting, large amounts of code duplication and/or magic numbers. If the comments are present, they may even be outright incorrect or deceptive. The code is cryptic and difficult to read and debug. Method names are not self-describing and understanding their functionality often requires reading the code itself.	**.5** (1.43%) The code used to implement the Stock class is in a reasonable state, with mostly acceptable-to-good use of variable, field and method names. The code is fairly well structured with a mostly appropriate level of whitespace and commenting. Comments are accurate and up-to-date. Code duplication and other antipatterns in need of source-level refactoring are kept to a minimum. The code is not too difficult to read, and method names and variables are mostly self-describing.	**1** (2.86%) The Stock class was implemented very competently, with well-chosen identifier names and excellently structured code that is self-documenting, includes comments and whitespace in appropriate quantities to make the code even clearer. Comments are accurate, up-to-date and describe the logic behind the code, not merely what the code is doing. Source-level refactorings are not needed to improve the code.
Code Quality - Truck	**0** (0%) The Truck, Ordinary Truck and Refrigerated Truck classes were either not implemented or implemented	**.5** (1.43%) The code used to implement the Truck, Ordinary Truck and Refrigerated Truck classes is in a reasonable	**1** (2.86%) The Truck, Ordinary Truck and Refrigerated Truck classes were implemented very competently, with

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	very poorly, with badly-chosen variable, field and method names, poorly structured code with either too much or too little use of whitespace and commenting, large amounts of code duplication and/or magic numbers. If the comments are present, they may even be outright incorrect or deceptive. The code is cryptic and difficult to read and debug. Method names are not self-describing and understanding their functionality often requires reading the code itself.	state, with mostly acceptable-to-good use of variable, field and method names. The code is fairly well structured with a mostly appropriate level of whitespace and commenting. Comments are accurate and up-to-date. Code duplication and other antipatterns in need of source-level refactoring are kept to a minimum. The code is not too difficult to read, and method names and variables are mostly self-describing.	well-chosen identifier names and excellently structured code that is self-documenting, includes comments and whitespace in appropriate quantities to make the code even clearer. Comments are accurate, up-to-date and describe the logic behind the code, not merely what the code is doing. Source-level refactorings are not needed to improve the code.
Code Quality - Manifest	**0** (0%) The Manifest class was either not implemented or implemented very poorly, with badly-chosen variable, field and method names, poorly structured code with either too much or too little	**.75** (2.14%) The code used to implement the Manifest class is in a reasonable state, with mostly acceptable-to-good use of variable, field and method names. The code is fairly well	**1.5** (4.29%) The Manifest class was implemented very competently, with well-chosen identifier names and excellently structured code that is self-documenting, includes comments and

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	use of whitespace and commenting, large amounts of code duplication and/or magic numbers. If the comments are present, they may even be outright incorrect or deceptive. The code is cryptic and difficult to read and debug. Method names are not self-describing and understanding their functionality often requires reading the code itself.	structured with a mostly appropriate level of whitespace and commenting. Comments are accurate and up-to-date. Code duplication and other antipatterns in need of source-level refactoring are kept to a minimum. The code is not too difficult to read, and method names and variables are mostly self-describing.	whitespace in appropriate quantities to make the code even clearer. Comments are accurate, up-to-date and describe the logic behind the code, not merely what the code is doing. Source-level refactorings are not needed to improve the code.
Code Quality - Store	**0** (0%) The Store class was either not implemented or implemented very poorly, with badly-chosen variable, field and method names, poorly structured code with either too much or too little use of whitespace and commenting, large amounts of code duplication and/or magic numbers. If the comments are	**.5** (1.43%) The code used to implement the Store class is in a reasonable state, with mostly acceptable-to-good use of variable, field and method names. The code is fairly well structured with a mostly appropriate level of whitespace and commenting. Comments are accurate and up-to-date. Code	**1** (2.86%) The Store class was implemented very competently, with well-chosen identifier names and excellently structured code that is self-documenting, includes comments and whitespace in appropriate quantities to make the code even clearer. Comments are accurate, up-to-date and describe the logic behind

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	present, they may even be outright incorrect or deceptive. The code is cryptic and difficult to read and debug. Method names are not self-describing and understanding their functionality often requires reading the code itself.	duplication and other antipatterns in need of source-level refactoring are kept to a minimum. The code is not too difficult to read, and method names and variables are mostly self-describing.	the code, not merely what the code is doing. Source-level refactorings are not needed to improve the code.
Exception handling	**0** (0%) The submitted code does not implement the three custom Exception classes required by the assignment specification, or implements them so poorly that they cannot be used as proper exceptions, or alternatively implements them but doesn't use them. In cases where exceptional behaviour can occur within classes, exceptions are not thrown, or alternatively legitimate	**.5** (1.43%) The submitted code implements the three custom Exception classes correctly and mostly throws them where it is appropriate to do so throughout the code. For the most part these exceptions then make it back to the user in the form of error messages delivered through the GUI and the messages shown are generally appropriate and useful to the user.	**1** (2.86%) The submitted code implements the three custom Exception classes correctly and throws them wherever appropriate. These exceptions are then gracefully handled at the level of the GUI and displayed to the user in the form of useful error messages without throwing confusing things like stack traces or messages that presume knowledge of the internal workings of the program in their faces.

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	exceptions are thrown but are handled in the wrong place, stopping the errors from getting back to the user through the GUI.		
Functionality - View store capital	**0** (0%) The user is unable to use the submitted application to view the store's current capital. Either the program fails to start at all, or this particular item of functionality is missing / bugged, or the user interface is so unintuitive a reasonable user would not be able to find it and use it.	**.25** (.71%) The user is able to use the submitted application to view the store's current capital, but the functionality is not quite spot-on - perhaps the value shown is not in dollars, or is formatted in the wrong way, or shows an inappropriate number of decimal places. Alternatively, the user experiences bugs when attempting to view the store's current capital, even though these bugs do not ultimately prevent the user from completing this activity.	**.5** (1.43%) The user is able to use the submitted application to view the store's current capital without any problems or inaccuracies.
Functionality - View inventory	**0** (0%) The user is unable to use the submitted application to	**.5** (1.43%) The user is able to use the submitted application to	**1** (2.86%) The user is able to use the submitted application to

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	view the store's inventory. Either the program fails to start at all, or this particular item of functionality is missing / bugged, or the user interface is so unintuitive a reasonable user would not be able to find it and use it.	view the store's inventory, but the functionality is not quite spot-on - for example, not every Item property is shown, or some of the values are off or formatted incorrectly. Alternatively, the user experiences bugs when attempting to view the store's inventory, even though these bugs do not ultimately prevent the user from completing this activity.	view the store's inventory without any problems or inaccuracies.
Functionality - Load item properties	**0** (0%) The user is unable to use the submitted application to load an item properties file. Either the program fails to start at all, or this particular item of functionality is missing / bugged, or the user interface is so unintuitive a reasonable user would not be able to find it and use it.	**.5** (1.43%) The user is able to use the submitted application to load an item properties file, but the functionality is not quite spot-on - for example, not all the items are loaded or some of the properties are not loaded correctly. Alternatively, the user experiences bugs when attempting to load an item properties file,	**1** (2.86%) The user is able to use the submitted application to load an item properties file without any problems or inaccuracies.

	Poor (zero marks)	Fair (half marks)	Good (full marks)
		even though these bugs do not ultimately prevent the user from completing this activity.	
Functionality - Export shipping manifests	**0** (0%) The user is unable to use the submitted application to export a shipping manifest based on the current inventory. Either the program fails to start at all, or this particular item of functionality is missing / bugged, or the user interface is so unintuitive a reasonable user would not be able to find it and use it.	**1.25** (3.57%) The user is able to use the submitted application to export a shipping manifest based on the current inventory, but the functionality is not quite spot-on - for example, the shipping manifest may order all the items required, but suboptimally fill trucks resulting in the store losing more money than it should, or may order too many or too few items. Alternatively, the user experiences bugs when attempting to export a shipping manifest, even though these bugs do not ultimately prevent the user from completing this activity.	**2.5** (7.14%) The user is able to use the submitted application to export a shipping manifest based on the current inventory without any problems or inaccuracies.

	Poor (zero marks)	Fair (half marks)	Good (full marks)
Functionality - Load shipping manifests	**0** (0%) The user is unable to use the submitted application to load a shipping manifest and update the store's inventory and capital accordingly. Either the program fails to start at all, or this particular item of functionality is missing / bugged, or the user interface is so unintuitive a reasonable user would not be able to find it and use it.	**.5** (1.43%) The user is able to use the submitted application to load a shipping manifest and update the store's inventory and capital accordingly, but the functionality is not quite spot-on - for example, it may calculate the shipping cost incorrectly, resulting in the wrong amount of capital being subtracted. Alternatively, the user experiences bugs when attempting to load a shipping manifest, even though these bugs do not ultimately prevent the user from completing this activity.	**1** (2.86%) The user is able to use the submitted application to load a shipping manifest and update the store's inventory and capital accordingly without any problems or inaccuracies.
Functionality - Load sales records	**0** (0%) The user is unable to use the submitted application to load a sales record and update the store's capital and inventory accordingly. Either the program fails to	**.5** (1.43%) The user is able to use the submitted application to load a sales record and update the store's capital and inventory accordingly, but the functionality is not quite spot-	**1** (2.86%) The user is able to use the submitted application to load a sales record and update the store's capital and inventory accordingly without any

	Poor (zero marks)	Fair (half marks)	Good (full marks)
	start at all, or this particular item of functionality is missing / bugged, or the user interface is so unintuitive a reasonable user would not be able to find it and use it.	on - for example, it may not update the store's capital correctly based on the items sold and their price, or it may not remove the correct number of items from inventory. Alternatively, the user experiences bugs when attempting to load a sales record, even though these bugs do not ultimately prevent the user from completing this activity.	problems or inaccuracies.

Name:**Assignment 2 CRA**

Exit