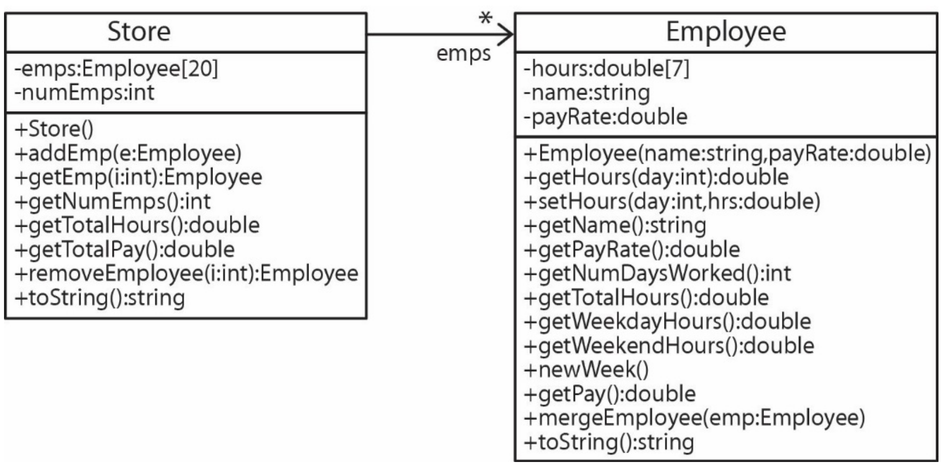
HW T2

You can work individually or in groups of 2.

Names: ______________________________ ______________________________

Problem

Consider the *Store* and *Employee* classes below. The *Employee* class was considered in a previous assignment and you can consider it to be fully tested. You will use the category-partition method to construct a test specification and JUnit tests for the two *Store* methods: *addEmp* and *removeEmployee.*



a. *emps* – An instance variable, which is an array that can hold up to 20 Employee objects. Employees are stored sequentially, with no gaps, starting with position 0.
b. *numEmps* – An instance variable that stores the total number of Employee objects in the *emps* array. Initially this value is 0.
c. *addEmp(e:Employee)* – Adds the Employee, *e* in the next available position. If there are already 20 employees and there is an attempt to add another then this method should do nothing, but should not crash.
d. *getEmp(i:int):Employee* – Returns the employee at position *i* if there is one, otherwise returns *null.*
e. *getNumEmps():int* – Returns the number of employees.
f. *getTotalHours():double* **–** Returns the total number of hours worked over all the employees.
g. *getTotalPay():double* **–** Returns the total pay worked over all the employees.
h. *removeEmployee(i:int):Employee* – Removes the employee at position *i* if there is one and returns it. All other employees to the right should be moved over one position to the left. If *i* is out of range then return null.
i. *toString():string* – Returns a message with this format: like the one shown below (for example, if there are 3 employees).

Steps

1. Use the category-partition method to develop a test specification for *addEmp.* Document the steps you took to arrive at this providing rationale as appropriate.

   - **The more I can see your thought process the better.**
   - **Type your results in a document in the format shown in Appendix A.**

2. Repeat Step 2 using *removeEmployee.* Note that *emps* being an array means there are quite a number of things to consider for testing this method.

3. Download the code for the *Store* class*.* Drag the *Store* class into an Eclipse project. Drag in the *Employee* class from HW_T1.

4. Develop JUnit tests for each method from the specification.

Deliverables

1. Code – zip the *prob1* package which includes your JUnit test class, *Store* class, *Employee* class, and Test Derivation document. Name the zip file: LastName1_LastName2.zip and submit on Blazeview in the dropbox named, *HW T2*.

Appendix A

Document template for submission. Name file: HW_T2_lastName1_lastName2_testDerivation.docx. **Delete this line and everything above it before submitting**

HW T2

Name(s) ______________________________ ______________________________

addEmp(e:Employee) Test Derivation

Note: Steps 1, 2, and 3 can be combined here if clearly explained. If so, just leave 1 and 2 empty and put your answer in 3.

1. Identify inputs/parameters for each feature. Supply justification and/or explanation as necessary.

2. Identify categories/characteristics for each input/parameter. Supply justification and/or explanation as necessary.

3. Partition categories into choices. Supply justification and/or explanation as necessary.

4. Identify constraints among choices and write your TSL input file. Show the contents of the file here. Supply justification and/or explanation as necessary.

5. Produce and evaluate test case specifications (frames) using TSL as needed. Show the test frames here. Supply justification and/or explanation as necessary.

6. Generate test cases from test case specifications. Show a table similar to the one shown below that shows each values you will use for each characteristic for each test. Supply justification and/or explanation as necessary.

	Values for Choices		
Test	**Name or Symbol for Characteristic 1**	**Name or Symbol for Characteristic 1**	**…**
1			
2			
…			

7. Identify what criteria must hold true for each test case to pass, *i.e.* what are you going to assert? Note that sometimes there are multiple side-effects so each one must be checked. Provide a numbered list. Supply justification and/or explanation as necessary.

    1. …
    2. …

removeEmployee(pos:int):Employee **Test Derivation**

Note: Steps 1, 2, and 3 can be combined here if clearly explained. If so, just leave 1 and 2 empty and put your answer in 3.

1. Identify inputs/parameters for each feature. Supply justification and/or explanation as necessary.

2. Identify categories/characteristics for each input/parameter. Supply justification and/or explanation as necessary.

3. Partition categories into choices. Supply justification and/or explanation as necessary.

4. Identify constraints among choices and write your TSL input file. Show the contents of the file here. Supply justification and/or explanation as necessary.

5. Produce and evaluate test case specifications (frames) using TSL as needed. Show the test frames here. Supply justification and/or explanation as necessary.

6. Generate test cases from test case specifications. Show a table similar to the one shown below that shows each values you will use for each characteristic for each test. Supply justification and/or explanation as necessary.

	Values for Choices		
Test	**Name or Symbol for Characteristic 1**	**Name or Symbol for Characteristic 1**	**…**
1			
2			
…			

7. Identify what criteria must hold true for each test case to pass, *i.e.* what are you going to assert? Note that sometimes there are multiple side-effects so each one must be checked. Provide a numbered list. Supply justification and/or explanation as necessary.

    1. …
    2. …