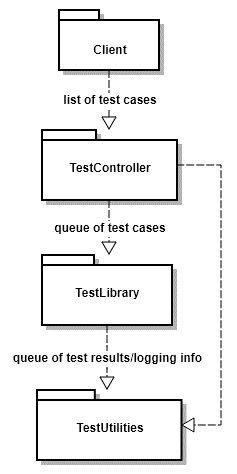
**Test Harness**

**Phase #1 – Architecture and Design**

**Group:** Santhosh Srinivasan, Jiawen Zhen, Alifa Stith

**1. Architecture**

The high-level architecture for Test Harness can be reasonably divided into three distinct components: *TestController*, *TestLibrary*, and *TestUtilities*. The *Client* component is included to represent an easily extensible input source (console, GUI, etc.).

**1.1. TestController Package**

The *TestController* package is the main entry point to the Test Harness application and will be responsible for the following functionality:

* Launching the Test Harness application and receiving any input provided by the user;
* Identifying and collecting all *TEST* functions within the given source code;
* Initializing and maintaining queues; and
* Storing and maintaining a summary of test results.

**1.2. TestLibrary Package**

The *TestLibrary* package can be thought of as providing the main functionality of the application. This is the package responsible for running all the tests and catching all exceptions. The *TestLibrary* package also provides the test result logging messages as well as any exception messages.

**1.3. TestUtilities Package**

The job of the *TestUtilities* package is to collect and log the output data throughout the execution of Test Harness. It calculates time information for each test as well as for the summary of test results. This package is crucially responsible for writing the logging information to permanent storage within the file system.

Figure 1. Block diagram of major components of Test Harness, with arrows indicating the flow of program execution.

**1.4. Queue Interface**

The packages communicate primarily through a set of two queues: one for the input test cases (called Q1), and another for the test results and log messages (called Q2). This interface was chosen because it is elegantly extensible to support a future multithreaded design. The *TestController* package can enqueue tests upon Q1 while the *TestLibrary* package dequeues test cases and runs them through their tests. Similarly, the *TestLibrary* package can enqueue test result messages upon Q2 while the *TestUtilities* package dequeues and logs them.

**2. Design**

The design of Test Harness breaks down the architecture from the previous section into individual classes. The *TestController* package is comprised of three classes: *TestHarness*, *TestIdentifier*, and *TestResultCounter*; the *TestLibrary* package contains an additional three classes: *TestRunner*, *TestAssertion*, and *TestExceptionHandler*; and two classes make up the *TestUtilities* package: *TestLogger* and *TestTimer*.

Diagram

Description automatically generated

Figure . Class diagram (UML) for Test Harness. Classes are grouped by package membership, and arrows show USING relationships between classes.

**2.1. Classes in TestController Package:**

**TestHarness, TestIdentifier, TestResultCounter**

The *TestHarness* class is the entry point into the Test Harness application. Its first task is to obtain the *TEST* functions to test, which it does by prompting the *TestIdentifier* class to provide tests from the source code. *TestIdentifier* then scans the source code, collecting pointers to all *TEST* functions and enqueuing them onto Q1. *TestHarness* also initiates the *TestLibrary* package’s execution of tests on the functions in Q1. Finally, *TestHarness* provides a communication interface between the *TestResultCounter* class, which stores and maintains a summary of test results, and the classes of the *TestUtilities* package.

**2.2. Classes in TestLibrary Package:**

**TestRunner, TestAssertion, TestExceptionHandler**

**2.3. Classes in TestUtilities Package:**

**TestTimer, TestLogger**