# **CS160 Outcome Assessment Exam**

CS160 contributes to three of the [learning outcomes for the BSCS degree](http://www.sjsu.edu/cs/accreditation/bscs/outcomes/index.html):

       Outcome H: Recognition of the need for and an ability to engage in continuing professional development

       Outcome I: An ability to use current techniques, skills, and tools necessary for computing practice

       Outcome K: An ability to apply design and development principles in the construction of software systems of varying complexity

To maintain the program's accreditation, these outcomes must be periodically assessed to see how well CS160 is doing. So really, we are testing CS160, not the students. That's the idea behind this test. Please try to answer the following questions to the best of your ability. You are allowed to use any online resources, but please don't get help from an actual person. Try to have fun with the test. See how well you stack up to the program outcomes. Maybe note any deficiencies that you should correct.

**BetaTester**

Similar to [JUnit](https://en.wikipedia.org/wiki/JUnit), BetaTester is a tool that allows programmers to create and run tests of their Java applications.

There are two types of tests in BetaTester: unit tests and test suites. A test suite is a collection of tests. Running a test suite runs each of its members and returns the total number of errors discovered.

A unit test has a name, a unary method to be tested, and a hash map associating method inputs to their expected outputs. When run, a unit test calls its method with each of the inputs in the hash map, comparing the output with the expected output from the hash map. It returns the number of errors. (What's a unary method? A unary method is a method that expects a single explicit input. Java 8 provides the interface Function<Domain, Range> to represent unary methods as objects. A hash map has the type Map<Domain, Range>.)

For example, suppose a Calculator class has methods for squaring and cubing numbers.

class Calculator {  
 public Double square(Double x) { return x \* x; }  
 public Double cube(Double x) { return x \* x \* x; }  
 // etc.  
}

Here's how a unit test could be set up for testing the calculator's square method:

Calculator calc = new Calculator();  
UnitTest<Double, Double> squareTest   
 = new UnitTest<Double, Double>("square", calc::square); // calc::square is a method-object  
// add some rows to the hash map:  
squareTest.put(3.0, 9.0);  
squareTest.put(4.0, 16.0);  
squareTest.put(5.0, 25.0);  
squareTest.put(6.0, 36.0);

Assume a similar unit test was created for the cube method. We could create a test suite for testing all of the calculator's methods as follows:

TestSuite calcTests = new TestSuite();  
calcTests.add(squareTest);  
calcTests.add(cubeTest);

Of course we could also add test suites to calcTests.

Finally, we could run calcTests and print the number of errors detected:

System.out.println("# of errors = " + calcTests.run());// this shoud print 0

**Problem 1: Modeling BetaTester Use Cases (Outcome K)**

Based on the description given above, draw a UML use case diagram showing the use cases that will be needed. Don't forget to include any important actors and arrows. You may add use cases inferred from the description, but don't add speculative use cases or actors. For example, a use case for sounding alarms when bugs are detected might be nice, but there's nothing in the description to suggest that it is needed.

Submit an image of your diagram. Either use a UML diagramming tool (free and trial ones like [StarUML](http://staruml.io/) are available) and export your diagram as an image, or draw a picture by hand and take a photo of it.

Don't know what a use case diagram is? Then declare a Java interface:

interface BetaTester<Domain, Range> {???}

Include in this interface declarations of the top-level functions BetaTester should make available to programs and programmers. The names of the functions should reflect what they do, but feel free to add a comment to each function to clarify what it does. Try to include any necessary parameters. Don't worry too much if your interface contains errors. This would be expected because the types of the parameters and return values haven't been declared yet. This is more of a requirements specification than an interface that will actually get implemented. Turn your declaration in as BetaTester.java. You may add as a comment any explanation you see fit about why you were not able to draw a use case diagram.

**Problem 2: Designing BetaTester (Outcome K)**

Based on the description given above, draw a UML class diagram showing how you would design BetaTester. Include all important details—classes, attributes, operations, types, arrows, multiplicities, etc. As with problem 1, you may add classes, interfaces, and arrows inferred from the description, but don't add speculative classes like LaserCanon or WarpDrive.

Did you use a design pattern? If so, which one?

Submit an image of your diagram. Either use a diagramming tool and export your diagram as an image, or draw a picture by hand and take a photo of it.

Don't know what a class diagram is? Create a file called BetaTesterApp.java. In this file put all of the class and interface declarations you think BetaTester will need. Don't include constructors, getters, setters, delegators, and other "administrative" methods. Also, don't include implementations. All of the methods should be abstract, for now. You may add as a comment any explanation you see fit about why you are not able to draw a class diagram.

**Problem 3: Using BetaTester (Outcome I)**

How comfortable are you with your IDE? Power user? Just the basics? Intimidated by it? Let's see.

The file [calc.zip](http://www.cs.sjsu.edu/faculty/pearce/assessment/cs160/betaTester/calc.zip) contains two files:

Calc.java (a calculator that calculates a few simple math functions).

CalculatorTest.java (contains a main method that creates and runs some calculator tests).

The BetaTester application described above is contained in [bt.jar](http://www.cs.sjsu.edu/faculty/pearce/assessment/cs160/betaTester/bt.jar).

Use your IDE to create a Java project called calculator. Download bt.jar and add it to your project's build path. Download and unzip [calc.zip](http://www.cs.sjsu.edu/faculty/pearce/assessment/cs160/betaTester/calc.zip) and add it to your project as a linked resource.

       Open your project's property sheet so that the project's build path can be seen. The build path should contain bt.jar. Position the property sheet window so that the project's source files (i.e., Calculator.java and CalculatorTest.java) can be seen in the project's list of sources. (E.g. as seen in the package explorer frame of Eclipse.) Take a screen shot and save it to an image file. Upload the image file to Canvas.

       Run CalculatorTest.main. What is the exact output produced? (This can be submitted as a text file.)

Unable to run Calculator.Test? Here is the source code for BetaTester: [bt.zip](http://www.cs.sjsu.edu/faculty/pearce/assessment/cs160/betaTester/bt.zip). Unzip it and calc.zip, and copy the files into your project. Still no? Then submit a brief message explaining how far you got, what went wrong, etc.

Unable to take screenshots? That's part of the challenge. Google it.

**Problem 4: Debugging Calc.java (Outcome I)**

Apparently Calculator.isPrime contains a few bugs. Set a breakpoint inside of this method. Step through it. Submit a screen shot of the debugger window showing the local variables of isPrime at the point of error.

Describe the bug? How would you fix it?

Couldn't work the debugger? Why?

**Problem 5: And now for a brief survey (Outcome H)**

Give short answers (< 50 words) to the following questions.

H1. Describe an important technology that has emerged in the past five years.

H2. Describe one technical area where you lack needed expertise.

H3. Describe a technical skill that you recently acquired and applied to a project.

H4. In your words, why do computer professionals need to continuously learn new skills and technologies?