Testing

A *bug* is an error in a program. Testing is the process of analyzing and executing a program to determine whether it has bugs. Our main tool in testing is the *test case*: a set of inputs together with the expected output.

Debugging is a separate process. Having detected that the program has a bug, we need to find out what caused it and remove it. This process is called *debugging*.

Principles for creating test cases

In this lecture, we concentrate on test cases —how to develop them, when to develop them, and why. Writing a bunch of test cases, as we ask you to do in assignments, may feel like wasting time. But if you do this task properly, you will find that your productivity will increase.

Developing good test cases is important. Some companies have quality assurance (QA) teams that do nothing but test software —other people, not the original programmers, do testing once the programmers believe their product is finished.

When both writing and testing a method, understand exactly what the method is supposed to do. This means that *the method specification should be written before the method body.* Some people advocate also writing a set of test cases before writing the method body, for this helps us understand what the method is supposed to do. This can be useful when writing a method from a spec that someone else gave us.

Test early and often. The sooner you test a method, the sooner you will find and fix bugs. In fact, as soon as you have written a method specification, you should write and example of a call to that method, so that you will be thinking about how the method might be used even before you write the method body.

Think about the whole set of inputs. It is not always feasible to test every single value of input because set of input can be infinitely large such as set of integers, so we need some kind of strategy to make sure although our test cases do not include the entire of inputs; they are enough to guarantee that our program will work properly. We will give you an example of set of order pair (x, y) of integer here to illustrate this principle.

First, we will visualize the set of order pair (x, y) as a square. There are about 2^{64} points in this square. Testing all of them will unnecessarily waste of time, so we need to find a set of points in this square that is in interesting regions. In this case, we may want to first test a few points in all four quadrants such as (15, 17), (-10, 20), (-100, -200), or (134, -27). Then, we may want to try some extreme values or *corner cases* such as -2^{31} (minimum integer), or $2^{-31} – 1$ (maximum integer), or 0. There are other interesting values we may forget as well such as 1, or -1. We can also apply this idea to different sets of inputs. For example, if our set of inputs is a class, the extreme value will be **`null`**.

Develop test cases that provide test coverage. A group of test cases provides test coverage if exercising them will cause every single part of the program to be executed at least once.

Test only one thing at a time. If you test several things at once and the test fails, you will have a harder time figuring out why. Also, it is easier to lose track of what you are testing and harder to make sure that you have covered all the cases.

Test each method thoroughly as it is completed. Once a method is tested and its correctness is assured (as much as possible), you will be quite sure that any error that arises is not in that method. It lies elsewhere. This reduces, to a large extent, the area of the program to be investigated when developing test cases, testing, and debugging.

Verify the documentation. As you test, ask yourself questions that you would not normally think of while writing the code. Make sure the documentation answers those questions.

Approaches to creating test cases

Exhaustive testing. Testing a program on all possible inputs, sounds good but is generally infeasible for most programs because of the number of possible test cases.

Blackbox testing, or functional testing. In making up test cases, one looks only at the specification of the program (not the program itself) in deciding what test cases to try. The program is a black box, and you cannot see inside it.

Glassbox testing, or structural testing. One looks at the program itself when developing test cases and uses the structure of the program to recognize possible trouble spots and develop test cases to exercise them.

Testing a class

Testing a class is harder than testing a method because it generally contains many methods. This is what we should do when we test a class.

Declare at least one and perhaps more variables with the class as their type. Then, create instances of the class and assign them to the variables.

Check that the constructor of the class works properly. This means checking whether the fields of each new instance were properly initialized. Function `toString` can be useful in testing. If `toString` has been written, it is easy to print the contents of a class instance using it:

```
    System.out.println(x.toString());
```

This test will also help test method `toString`.

Test all the other methods. This testing can follow the principles described earlier.