Testing

Introduction

Software must be tested, for everyone makes mistakes —misinterpretations of specifications, logical mistakes, typos. So one tests with *test cases*. A test case is a set of inputs to the software along with the expected outputs.

It is best to save test cases and organize them in some fashion so that testing with them can be easily repeated whenever a change is made. The JUnit testing classes one develops in DrJava or Eclipse provides this ability.

Unit testing refers to testing the smallest testable parts of an application. The typical unit is a single method or small group of methods. The goal of unit testing is to isolate the unit from the remainder of the code and test and debug it until one is assured that it is correct. After one unit is thoroughly tested, another unit that relies on the first unit could be tested/debugged. Thus, one could steadily incorporate more and more units until testing is complete.

Unit testing may seem time consuming and tedious. But in the long run, it is the simplest, most efficient, and most effective way of testing. If an error is detected, it should be in the current unit being tested because previous methods were already tested. So, if earlier units were tested properly, one doesn't have to look far for the error.

Best practices for unit testing

- Test one unit at a time.
- Ensure that unit test cases are independent of each other. Any given behavior should be tested in one and only one test procedure. Otherwise, if you later change that behavior, you'll have to change several tests.
- Name your unit tests clearly and consistently. With JUnit testing, the name of the testing procedure should be enough unless a program is very big. Make sure your test cases are readable so that anyone picking up unit test cases can execute them without any issues.
- Always fix bugs identified during unit testing before moving on to something else.

Black box testing

Black box testing refers to testing a unit based solely on its specification. The unit is a black box, and you can't look inside it, so develop test cases based on the specification. A software company may have a team whose only job is to do black box testing. Their do their job in as evil manner as possible, looking for weird cases that the software developers might not have handled properly.

When you first start developing a method based on its specification, it makes sense to make up test cases, writing down the input and corresponding output. This can help you uncover ambiguities and misunderstandings. You will also have some test cases with which to start testing.

When developing tests cases, concentrate on the size and type of every input to the unit. For example, if an input value can be any int, have test cases for Integer.MIN_VALUE, -1, 0, 1, and Integer.MAX_VALUE. If one input is the hour of the day, in the range 0..23, develop test cases for 0, 12, and 23.

White box, or structural, testing

White box testing refers to the ability to look at the code to help develop test cases. The name is silly. Whether a box is black or white doesn't matter; you still can't look in it. *Glass box* or *transparent box testing* would be better terms. It is also called *structural testing*. Let's look at important facets of structural testing.

- **Test each statement of a unit** in at least one test case. Otherwise, how can you know that it is correct?
- **Test each branch of a unit** in at least one test case. For example, if there is an if-statement with condition *B*, at least two test cases are needed, one with *B* false and one with *B* true.
- **Test each expression thoroughly.** Of particular importance here are boolean expressions like b && (c || d). Here we need four test cases: (1) b is false. (2) b is true and c is true (and d is false), (3) b is true and d is true (and c is false), (4) b is true and c is false and d is false. Thus, we identify various ways in which the expression can be true or false and writing a test case for each one.
- **Test extreme or corner cases.** Here is an example. Consider the loop **for** (**int** k= 0; k < n; k= k+1) {…} where it is known that n is in 0..50. Include test cases for the extreme cases n = 0, 1, and 50 and also for some inner value like n = 20.