CSD-4464 Java EE

Class 2: Recursion & Unit testing

Lambton College

Agenda

- Recursion
- Tiny bit of Maven Black Magic
- Unit Testing

Lambton College

What is recursion?!?

- Recursion is a self referencing method, that breaks problems down into smaller, easier to manage pieces called base-cases
- Divides and conquers
- Generally your quickest to a solution style of programming

- Example – factorials (base case is 1, recursive with n-1)

Factorial = n*n-1*n-2*n-3….n-m*1

Pros

- Makes difficult iterative problems easy, especially problems that can easily be divided into parts
- Quick development
- Code can be easier to read

Cons

- Stack overflow error – Java has a limit to the amount of nested function calls it can have, generally based on JVM settings
- Default Stack limit is usually around 6500-8500 nested calls
- Generally more costly than the iterative approach (performance wise)

Example recursive stack

```
public int factorial(int x) {
   if(x == 1) return x;
   return x*factorial(x-1);
}
```

Turns into

```
(factorial 3)
(* 3 (factorial 2))
(* 3 (* 2 (factorial 1)))
(* 3 (* 2 (* 1)))
(* 3 (* 2 ))
(* 3 2)
 6
```

Tail-Call Optimization

- A tail call is a fancy term that refers to a situation in which a method or function call is the *last* instruction inside of another method or function
- Not fully supported in java***
- Allows for recursive functions to no longer take extra stack space

Automated Testing and you

- Two types of automated testing – Unit Tests, and Integration Tests
- Saves time in the long run, you won’t have to manually test every change!
- Helps prevent bugs from being deployed as it catches the issues at build time.
- Allows for CI pipelines

Automated Testing Frameworks

- Junit
- Mockito
- TestNG
- Powermock
- Jtest
- And many more!

JUnit

- JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.
- De facto default / most popular Testing framework
- Uses a test runner per test suite
- Most popular version – 4 & 5 (versions aren't compatible**)

Unit Testing

- Used to test a single method at a time, you should try to keep the scope of the test minimal
- Projects tend to have many unit tests, and aim to have a coverage rate of >=90%
- Useful for proving that added code is correct and catches bugs before they get to integration points
- Make sure to do regression / negative testing

Integration Tests

- Important for large scale projects
- End to ends tests (enter your application from its entry points, tests its responses to be correct)
- Tests the expected behavior of your application

@Test

- The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case
- JUnit first constructs a fresh instance of the class then invokes the annotated method
- Any exceptions thrown by the test will be reported by JUnit as a failure
- If no exceptions are thrown, the test is assumed to have succeeded

@Before (ju4) / @BeforeEach (ju5)

- An annotation you can apply to a method that will cause Junit to run that method before each of the tests in the suite
- Generally used for doing the required setup for tests (e.g constructing dependencies)
- The @Before methods of superclasses will be run before those of the current class, unless they are overridden in the current class. No other ordering is defined.

@After (ju4) / @AfterEach (ju5)

- Annotating a public void method with @After causes that method to be run after the Test method
- All @After methods are guaranteed to run even if a Before or Test method throws an exception
- The @After methods declared in superclasses will be run after those of the current class, unless they are overridden in the current class
- Useful for cleaning up any resources or side effects your tests may create

Assert.* (ju4) / Assertions.* (ju5)

- Class that contains a set of assertion methods useful for writing tests.
- Only failed assertions are recorded.
- These methods can be used directly: Assert.assertEquals(...), however, they read better if they are referenced through static import:
- assertEqual(x,y) assertTrue(x) assertFalse(x) ect.

Matchers

- Matchers is an external addition to the JUnit framework, and were added by the framework called Hamcrest
- JUnit 4.8.2 ships with Hamcrest internally
- Allows for English language style asserts
- e.x

```
assertThat("this string", is("this string"));
assertThat(”123", is(equalTo(”123”)));
```

JUnit4 vs 5 Handling exceptions

Junit 5 uses lambdas in a wrapper

```
assertThrows(Exception.class, () -> {
        code expected to throw exception
});
```

Junit 4 uses annotations on the test methods

```
@Test(expected = Exception.class)
```