More testing

4/3/26

Administrivia

- I owe you homework…
- Talk at noon today (by alum about careers) is cancelled (because his flights were)
- Can still do the survey with questions about parallel and distributed computing (until 4/8)

Recall: JUnit

- Create a new class, normally called Test*Something*
- Give it the import statements
- Select the test tube and enable tests (select JUnit)
- Declare object references as attributes, create the objects (and set them up) in @Before
- Write testing methods (@Test) with assertions
  - I almost exclusively use assertEquals

Exceptions

- Errors that aren't explained via the method's return value

- Ex: What should remove() return when the list is empty?
  - One idea is null; implies List does not contain null

Throwing an exception

throw new IndexOutOfBoundsException();

exceptions are objects

create one of the correct type

Which of the following lines of a List remove method is incorrect?

```
T remove(int i) {  //remove and return value at position i
        if((i < 0) || (i > num))                                    //A
            throw new IndexOutOfBoundsException(); //B
        T retVal = vals[i];                                         //C
        num--;
        vals[i] = vals[num];                                        //D
        return retVal;
}                       //E: Not exactly one of the above
```

Which of the following lines of a List remove method is incorrect?

```
T remove(int i) {  //remove and return value at position i
        if((i < 0) || (i > num))                                    //A
            throw new IndexOutOfBoundsException(); //B
        T retVal = vals[i];                                         //C
        num--;
        vals[i] = vals[num];                                        //D
        return retVal;
}                       //E: Not exactly one of the above (A&D)
```

Detecting (and dealing with) an exception

```
try {
    //do stuff that may generate exceptions
} catch(ExceptionType ex) {
    //handle the exception
} catch(AnotherExceptionType ex) {
    //handle another kind of exception
}
```

Test cases for exceptions

```
try {
    //do stuff that should generate exceptions
    fail();
} catch(ExceptionType ex) {
    //NOTHING HERE
}
```

Programming by contract

- Class descriptions can be viewed as a contract between the implementer and the user; if the user meets the preconditions, then the method works as advertised
  - Can test this independent of the rest of the code
  - Why not make unit tests for each class **before** it is implemented?

Process for Test-Driven Development

- Write stub code: minimal code with correct signatures that compiles

- Repeatedly:
  - Write test cases for some of desired functionality
  - Implement that functionality
  - Use test cases to make sure it works

Find promises and write assertions for Refrigerator

- @Before method to set up test fixture
- @Test methods w/ assertEquals(correct, call)
- To test that an exception is thrown:

```
try {
        //do stuff that should generate exceptions
        fail();
} catch(ExceptionType ex) {
        //NOTHING HERE
}
```