February is a big month! Go do stuff

Colloquium - Assessing Text Intelligibility with Discourse Structure

One major aspect of communicating through text is to organize information into an understandable and well-structured discourse. This gives rise to two challenges in text processing: how to understand information

Spring Engineering Career Fair

The Spring 2017 Engineering Career Fair will be geared towards engineering and computer science students graduating in May and December 2017 as well as students seeking summer and fall 2017 internships and co-ops.

Informatics Showcase

"The University of Iowa Informatics Initiative (UI3), with UI Theme Semester and other cross-campus partners, proudly announces the second Informatics Week event on the University of Iowa campus February 13-17, 2017. This week is designed to showcase on-going informatics-related research, central resources available to support informatics activities, and the educational and career opportunities in informatics.

Spring Job & Internship Fair

The Pomerantz Career Center encourages all business and liberal arts and sciences majors to attend. This fair typically has around 1,400 students, and over 150 employers attend.

University of Iowa Computing Conference 2017

Welcome to the tenth annual University of Iowa Computing Conference, hosted by the UI ACM student chapter and the UI Department of Computer Science. We’ll have five speakers from across the country, from industry and academia alike. The UICC is run by students, for students, to promote computing as a science
CS 2230
CS II: Data structures

Meeting 8: Testing
Brandon Myers
University of Iowa
Today’s big ideas

• Testing
  • you should write tests
  • JUnit helps you organize and run your tests
  • a look at test driven development

• Different kinds of equality in Java
How do you know your program works?

What evidence did you have that your programs worked in HW1 and HW2?

https://b.socrative.com/login/student/
CS2230A ids 1000-4999
CS2230B ids 5000+
What do we mean by “your program works”? 

Usually, we mean that the implementation obeys the specification

```java
public void append(int d) { }
```

your implementation of LinkedList.append() is correct if it obeys the above specification
Many ways that people gather evidence that their program works

- **testing**: run it, automatically check the output
- **static analysis**: automatically check properties without actually running your program
- **handwritten proof**: write a mathematical proof that the program works
- **machine-checked proof**: automatically verify your proof that the program works is correct
- **model checking**: with help, the computer comes up with a proof for you, but only for “finite” number of cases
Evidence in CS2230

You will mostly rely on tests that you write.

- **testing**: run it, automatically check the output.

The compiler will check for simple errors for you (e.g., types and syntax).

- **static analysis**: automatically check properties without actually running your program.

Before you write code for an algorithm you want at least an informal proof that it is correct.

- **handwritten proof**: write a mathematical proof that the program works.
Approach in CS 2230

• you will learn to write and use tests

• you will learn to debug programs systematically

• you will learn to write “assertions” that check the invariants of your data structures
In many software projects, the tests get the final say

There is a “golden copy” of the code that passes all the tests

work-in-progress on new features in a separate copy. (might break things)

not allowed to “merge” the new feature into the golden copy unless all the tests pass

https://guides.github.com/introduction/flow/
Goal for today

create a class with this specification and test it

// insert the String s at the front of the list
void insertFirst(String s) {
}

// Remove the String at the front of the list
// and return it. Returns null if the list is empty.
String removeFirst() {
}

// Print out the elements of the list in order
void printList() {
}
Approach for today

We’ll use a methodology called test-driven development
• not the only choice
• but it can be very useful

identify a new feature (e.g., a new method)

- write minimal code to make the tests pass
- write 1 or more failing tests for the feature
Write JUnit tests for FLinkedList.removeFirst()
Implement `FLinkedList.removeFirst()` and fix it until all of our existing tests pass.
Write JUnit tests for FLinkedList.printList()?

writing tests for methods that print to the console is possible but a bit more complicated

Instead, let’s write a test for a helper method that would be useful to printList()!

    specifically, toString()
Since you are going to be dealing with equality a lot when writing JUnit tests...

```java
public static void assertEquals(Object expected, Object actual)

Asserts that two objects are equal. If they are not, an
AssertionError without a message is thrown. If expected and
actual are null, they are considered equal.

Parameters:

expected - expected value
actual - the value to check against expected
```

`assertEquals(expected, actual);`

```java
assertEq
```

```java
assertEquals(Object expected, Object actual) void

assertEquals(Object[] expecteds, Object[] actuals) void

assertEquals(double expected, double actual) void

assertEquals(long expected, long actual) void

assertEquals(String message, Object expected, Object actual) void
```
== isn’t always equal?

In Java, == does the expected for primitives.

```java
int a = 26;  // a == b is true
int b = 26;
```

```java
int a = 13;  // a == b is false
int b = 26;
```

Comparing two references checks if they are pointing to the same object.

```java
Patient p1 = new Patient("Marion", 100);
Patient p2 = new Patient("Marion", 100);
Patient p3 = p1;
```

```
// p1 == p2 is false
// p1 == p3 is true
Not pointing to the same object? not ==
```
The equals() method

```java
public boolean equals(Object o) {
    if (o instanceof Patient){
        Patient op = (Patient) o;
        return this.height==op.height &&
               this.name.equals(op.name);
    } else {
        return false;
    }
}
```

Every Java class already has an invisible equals method defined. But you have to override it with your own if you want to do something smarter like compare the fields.

Secondary new things in this snippet of code
- instanceof to check if o is a Patient
- casting o from Object to Patient

String.equals compares the content of the String instead of references
In discussion section next week

• practice with writing methods for linked lists
• more practice drawing boxes and arrows
• using JUnit
HW3: linked lists applied to browser tabs

You will write methods for a LinkedList class. Each method will provide new functionality for tabs (reorder, open, close, display, ...).
What to do now

• Quiz 2 / HW 2 due tonight

• HW 3 / Quiz 3 out now or soon