CSC207.01 2014S, Class 05: Classes and Objects

Overview

- Preliminaries.
  - Admin.
  - A few notes on writeup 3.
  - Questions.
- A brief introduction to objects.
- An exercise: Modeling Ushahidi.

Preliminaries

Admin

- Study break at 8 pm tonight in the commons. Peter is making cookies.
- For second years: Special extra credit for attending the lunch session you were invited to.
- I *think* I've responded to all of the writeups I've received as of 10 pm last night.
- I think I've responded to HW1.
  - FAQ updated.
  - Lessons added.
- No lab today! We'll do some small group and large group design.
- Does anyone need a partner for the assignment?
- Extra credit:
  - CS Extras, Thursday: Ushahidi, Android, and 207 by Spencer, Daniel, and Lea.
  - CS Table Friday: The ACM Code of Ethics
  - Convo Feb. 5. (I'll give my "Why go to convo" lecture closer to the date.)

A Few Notes on Writeup 3

- If your code is incorrect, you have an obligation to let me know that you know that it is incorrect.
- I'm not sure why we no longer get an error based on the overflow, but I'd swear I did last semester.
- A question about changing the preconditions: Are your new preconditions biased toward a particular implementation?
- For those of you who said "I'd change the implementation", how would you

- For your unit testing, the following is not a great test statement

  for (int base = -100; base < 100; base++) { expected = 1; for (int power = 0; power < 10; power++) { assertEquals ("Testing", expected, SampleMethods.expt (base, power)); expected *= base; } // for each power } // for each base

- If the test fails, you'll only see the "Testing" message. You'd probably prefer to see the base and power too.

```
assertEquals ("Testing " + base + "^" + power,
              expected, SampleMethods.expt (base, power));
```

- I'll admit that I often end up doing something like

```
int result = SampleMethods.expt (base, power);
if (result != expected)
  {
    fail ("For " + base + "^" + power + ", expected " +
          expected + ", got " + result);
  } // if we did not get the expected result.
```

- Doesn't 100^10 overflow? You probably want to have a more sensible stopping condition (perhaps involving a while loop).

- For testing the `double` version, you'll probably need to have a "close enough" metric (see documentation). assertEquals (message, expected, formula, ACCURACY)

Questions on HW2

Are you okay with the seemingly inefficient looping (or recursive) solution to `isOdd`?

Yes.

But smart programmers who think in C can probably find a more efficient solution.

Can we use a loop for `oddSumTo`?

Yes, but you shouldn't need one.

A brief introduction to objects

- Objects encapsulate data and methods that work with those data.
  - structs
  - plus functions on those structs (methods)
  - plus hiding the fields of the struct
- Kinds of methods
  - Constructors: Build new objects (given some data)
  - Observers: Extract information (e.g., ask for title of library book)
  - Mutators: Change the object
    - Check out a library book
- Like all classification systems, this breaks down rather quickly
  - Complex c = new Complex(1,2);
  - c.multiplyBy(c) - returns a new complex number
    - So is multiplyBy a Constructor?

        - Or is it an observer?
        - Or is it both
- As you design an object
    - What information will the client want to get?
    - What information will we allow the client to change?
    - What information do we need to build a new object?

An exercise: Modeling Ushahidi

Context:

- Crowdmapping with anonymity and some verification

Design Issues:

- What are the objects you'd want to model this system?
- What methods would you associate with those objects?

Objects

- Map (database?) - Contains the locations of the posts -
- Incident - Information on a single incident
    - Categoriezed, giving you different types of incidents?
- User - With classification (or subclassing) to inform us about capabilities
- Server - Something you submit information to and get information from
- Location - Used for incidents (and maybe for the map)
    - Longitude
    - Latitude
    - Place name (optional)
- Time - Used for incidents

Time

- Constructors
    - Inputs: Year, month, day, hour, minutes, seconds ...
    - Inputs: Timezone, Year, month, day, hour, minutes, seconds ...
    - Inputs: NONE - Give the current time (now)
    - Inputs: Unix time model (milliseconds since "the beginning of time")
    - Inputs: Month and day (rest is implict or unnecessary)
    - Inputs: String description
- Observers
    - getMonth, getDay, getYear, getWeekday, ...
        - With and without timezone
    - getDifferenceBetween (another date)
- Mutators

- Do we want to edit dates?

Incidents

- Constructors
- Observers
- Mutators