CMSC 433
Programming Language Paradigms and Technologies
William Pugh
March 22nd
Course grades

- Overall course grade 50 - 97

- Early warning grades:
  - < 62  F (3)
  - < 70  D (1)
  - < 78  C (9)
  - < 88  B (16)
  - 88+  A (9)
Midterm

• Midterm grades 29-95, average 62.3
• If you want a regrade, do not make any marks
  • Thrilled to correct any mistakes in grading
  • Hard to correct for "unfair problems" or "you took off too many points for my mistake"
• need to be consistent for all students
Thread jukebox

• I'd thought I was clear that a request to play a song should simply enqueue a request to play the song, and not block until the song was finished playing.

• evidently not

• but if you wanted a blocking solution, simply synchronizing the play method and not exiting the play method until the song is complete would work.
HTML forms

• Big problems:
  • price provided a form parameter
  • GET rather than POST
• Also: http rather than https
  • not really a HTML problem
• Not a problem:
  • the fact that both forms defined the same action
Decorators

• This problem stumped a lot of people
  • wasn't specifically called on on study guide
  • but was covered in class

• You should have been prepared for this, but
  • bumped minimum score on problem up to 5
    • if you scored less than 5, score was adjusted to 5
Nonsensical code

• Too many students wrote down code that just made no sense

• Not even close to syntactically valid Java code, or making any sense

• Just, wow

• test anxiety is the most charitable interpretation

• but you've got to be able to write code on a whiteboard or paper
A decorator

- Implements an interface
- Constructed using another object implementing that interface
  - wrap/delegates to that instance
  - intercepting some method calls
  - perhaps providing additional methods
public class NonRemovingIterator<E> implements Iterator<E> {
    final Iterator<E> i;
    public NonRemovingIterator(Iterator<E> i) {
        this.i = i;
    }
    public boolean hasNext() {
        return i.hasNext();
    }
    public E next() {
        return i.next();
    }
    public void remove() {
        throw new UnsupportedOperationException("remove not supported");
    }
}
public class EnhancedIterator<E> implements Iterator<E> {
    boolean canCallCurrent;
    E current;
    final Iterator<E> i;
    public EnhancedIterator(Iterator<E> i) {
        this.i = i;
    }
    public boolean hasNext() {
        canCallCurrent = false;
        return i.hasNext();
    }
    public E next() {
        current = i.next();
        canCallCurrent = true;
        return current;
    }
    public void remove() {
        i.remove();
        canCallCurrent = false;
    }
    public E curr() {
        if (!canCallCurrent) throw new IllegalStateException();
        return current;
    }
}
Common mistake

• A lot of people declared a field
  \[ E \text{ current} = \text{null} \]

• and assumed that if it was null, calling \text{curr()} wasn't legal.

• But null is a legal value that might be returned by iteration
Testing a semaphore

static class CheckSemaphoreWithTwoPermits extends MultithreadedTestCase {

    final Semaphore semaphore = Semaphore(2);

    public void thread0() {
        semaphore.acquire(2);
        assertEquals(0, getTick());
        semaphore.acquire(1);
        assertEquals(1, getTick());
    }

    public void thread1() {
        waitForTick(1);
        semaphore.release(1);
        assertEquals(1, getTick()); // optional sanity check
    }
}
Faulty Semaphore

Bug in my release implementation; was

```java
public synchronized void release(int p) {
    if (p == 0) this.notifyAll();
    this.permits += p;
}
```

Should be

```java
public synchronized void release(int p) {
    this.notifyAll();
    this.permits += p;
}
```
**Intended bug**

- The problem described is exactly the problem that you would get if the call to wait wasn't in a loop

- Buggy:

```java
public synchronized void acquire(int p) {
    if (this.permits < p)
        this.wait();
    this.permits -= p;
}
```
Correct code

```java
public synchronized void acquire(int p) {
    while (this.permits < p)
        this.wait();
    this.permits -= p;
}
```
Code review

- I looked at everyone's project 4 implementation
- We *need* to do more code reviews
static fields

• One project that defined all fields as static

• Some projects that defined some (seemly arbitrary fields) as static
putIfAbsent

- Understand why it is different than put
- If you are going to ignore return value, be sure you understand and can articulate why
My project 4 and 5 solutions
Core mastery track

• I'm considering allowing you all to take one of two tracks for the rest of the semester:
  • standard track
  • core mastery track

• Idea of the core mastery track: If you don't understand concurrency well, it might be better for you better to master concurrency and other concepts we've already covered than to add new topics you won't master either
Core mastery

• More concurrency projects and exercises
• fix old broken submissions
• parallel sorting
• concurrent observer implementation
• skip the map reduce and hadoop projects
• still responsible for concepts
• Flexible final exam (do 3 out of 4, …)
Core mastery

- Two different grade curves
- Best possible letter grade with core mastery is B-
  - you need to show A level skills on a smaller core of material
- Solid competency on core material earns you a grade of C
- Intended for those students struggling with the course, worried about getting a grade below C
- perhaps current overall grade < 75
Survey

- I'll email out a survey asking about interest in the core mastery track and a few other questions
- can do survey anonymously
- I'm also happy to respond to questions/concerns in person or email about the core mastery track
This week

• Cookie signups
• Open source idea
• Start work on project 6
• No office hours Friday through Monday
  • sorry, but both John and I made travel plans
  • some forum/email assistance possible
Open source project

• if you didn't submit anything before break, don't panic
  • but get it done this week
• If you want to look at some FindBugs results for some large projects, go to
  • http://findbugs.cs.umd.edu/cloud/
Concurrency refresher

- 3 different small concurrent abstractions
  - BoundedPoint
  - ConcurrentHashBag
  - KeyedExchanger
BlockingBoundedPoint

• A point constrained to be in the region 0 … maxX, 0 … maxY

• Point moveBy(int deltaX, int deltaY)
  • atomically adjusts the location of the point
  • blocking until the move would be in bounds
ConcurrentHashBag

• A Bag, also known as a multiset, is like a set except that we can have an multiple instances of a value.
  
  • e.g., \{ "a", "a", "b" \}

• Implementation should be efficient and concurrent.
  
  • don't simply use synchronized. Want to allow for concurrent noninterfering operations.
KeyedExchanger

- java.util.concurrent.Exchanger
- allows two threads to exchange values
- first thread to call exchanger waits until second thread arrives
- two threads exchange values, data structure reset
- keyed exchanger: call passes key and value
- only match against calls with equal keys
Remainder of semester

- 7 full weeks, 15 classes
- open source project
- Concurrency refresher
- distributed computing warm up
- implementing map reduce from scratch
- Simple hadoop example
- deploying hadoop to Amazon web services