Lecture 1: Introduction
Course Home Page

- Includes syllabus, tentative course schedule, professor / TA contact info, etc.
- Two TAs
  - Ashwin Lakshmanaswamy
  - Jonathan Hansford
Computing Environment

• Java 8
• Eclipse 4.5 (Mars)
• CS submit server
• Piazza (for on-line discussions)

For installation help on Java, Eclipse, submit server: www.cs.umd.edu/eclipse

JUnit 4+ recommended as well.
This Course

• “Programming Language Technologies and Paradigms”

• Could be a lot of things: logic / functional programming, testing, formal methods, theorem proving, interactive development environments, ...

• In actuality: *concurrency*
Concurreny?

• = “multi-threading”
  – Traditional applications are single-threaded: at any point during execution, at most one instruction can be executed next.
  – In multi-threaded applications, several instructions can be executed “next”!

• Programming languages include mechanisms for concurrency
  – Threads
  – Locks
  – Interrupts
  – Etc.
Why Concurrency?

• Performance
  If they can do operations simultaneously, applications run faster!

• Availability
  Compute-intensive parts of application need not slow down other parts (e.g. user interface)

• Application demands
  Many applications feature concurrency as part of system design (e.g. operating systems, communications protocols, simulations)
Course Focus

• How to program effectively using concurrency constructs in Java
• Towards this goal, we will:
  – Understand uses, pitfalls of concurrency
  – Gain proficiency in various mechanisms for managing concurrency
  – Do a number of projects in Java to put this understanding into practice
• Java is the vehicle, but the principles we learn will be applicable beyond
If Concurrency Is So Useful, Why Not Teach It Sooner?

• We do!

• However, concurrency is hard
  – Concurrent programs are hard to debug
  – Concurrent programs are hard to optimize
  – Concurrent programs are hard to test
Why Is Concurrency Hard?

- Nondeterminism!
  - Executing same program can yield different answers
  - Replaying a given execution is very difficult
- Concurrency breaks *procedural abstraction*
  - Procedural abstraction: a given sequence of instructions will always return the same result if started in the same state
  - Implication: you can think of a sequence of instructions as conceptually a single instruction
  - Basis for: compilation, method definition, etc.
Nondeterminism

• Suppose we have
  – Shared field `shared` that is initially 0
  – Two threads `t1, t2` with instance variables `myShared`, each of which does:
    ```java
    myShared = shared;
    myShared++;
    shared = myShared;
    ```

• What are possible values of `shared` afterwards?
  – 1, 2!
Procedural Abstraction

• Consider previous example, and suppose threads were launched via following:

```java
t1.start();
t2.start();
```

• If procedural abstraction holds
  – `t1.start()` is conceptually a single operation that increments shared
  – So is `t2.start()`
  – Only allowed answer would be 2!