Welcome to CS681!

Tue Thu 7pm to 8:15pm

S-1-006

Who am I?

- Academics
  - Associate Professor, UMass Boston (2010–)
  - Assistant Professor, UMass Boston (2004–2010)
    - Distributed systems, software engineering and AI
    - www.cs.umb.edu/~jxs/; dssg.cs.umb.edu
  - Post-doctoral Research Fellow, UC Irvine, CA (2000–2004)
  - Ph.D. in Comp Sci from Keio University, Japan (2001)
- Industrial
  - Consultant, cloud computing platform vendor, supply chain mgt. company
  - Tech Director, Object Management Group Japan
  - Co-founder and CTO, TechAtlas Comm Corp, Austin, TX
  - Programmer Analyst, Goldman Sachs Japan
- Professional
  - Member, ISO SC7/WG 19
  - Specification co-lead, OMG Super Dist. Objects SIG

Capstone Sequence: CS680-681-682

- CS680:
  - Lecture-based course
- CS681
  - Lecture-based course
- CS682
  - Project course
- CS 683 no longer exists.
- The department's web pages may not be perfectly accurate.
  - See https://www.cs.umb.edu/~dsim/cs622/spx.pdf

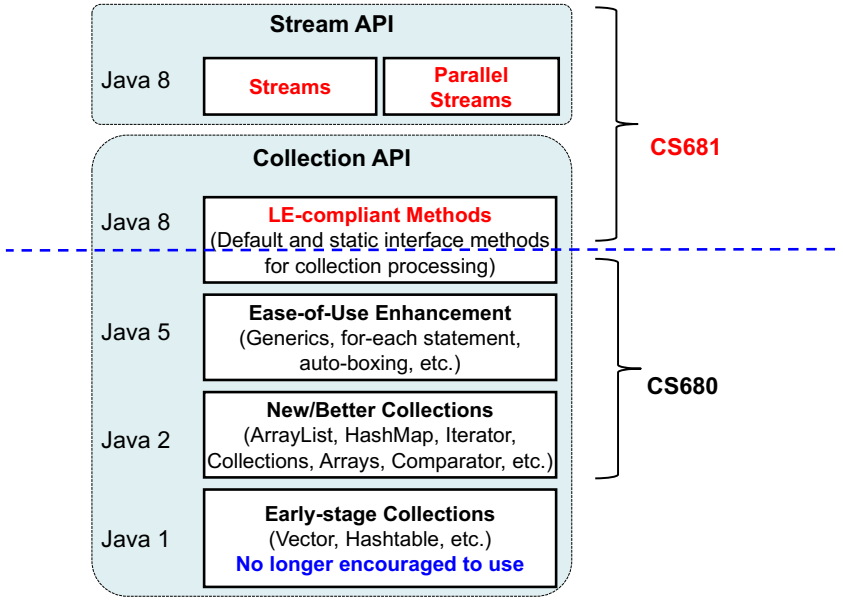
Course Topics: Advanced Software Engineering

- Functional programming with lambda expressions
- Concurrent programming (multi-threading)

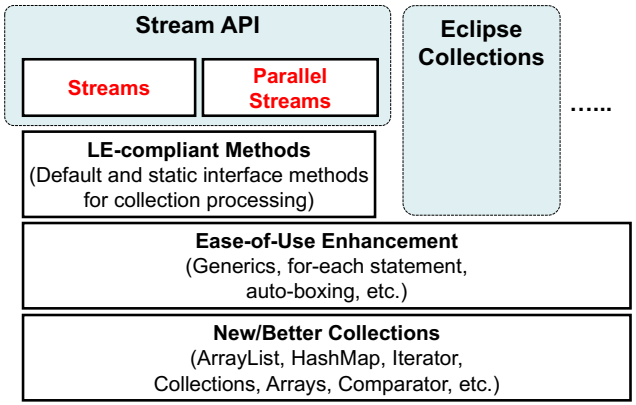
Course Topics # 1

- Functional programming with lambda expressions
  - Continuation from CS680
  - Collection processing with LEs
    - Stream API, which heavily uses Les
    - Design patterns: MapReduce

Collection and Stream APIs in Java



Extra Utilities



- Eclipse Collections
  - https://www.eclipse.org/collections/
  - https://github.com/eclipse/eclipse-collections-kata
  - Used to be Goldman Sachs (GS) Collections
    - https://github.com/goldmansachs/gs-collections

Course Topics # 2

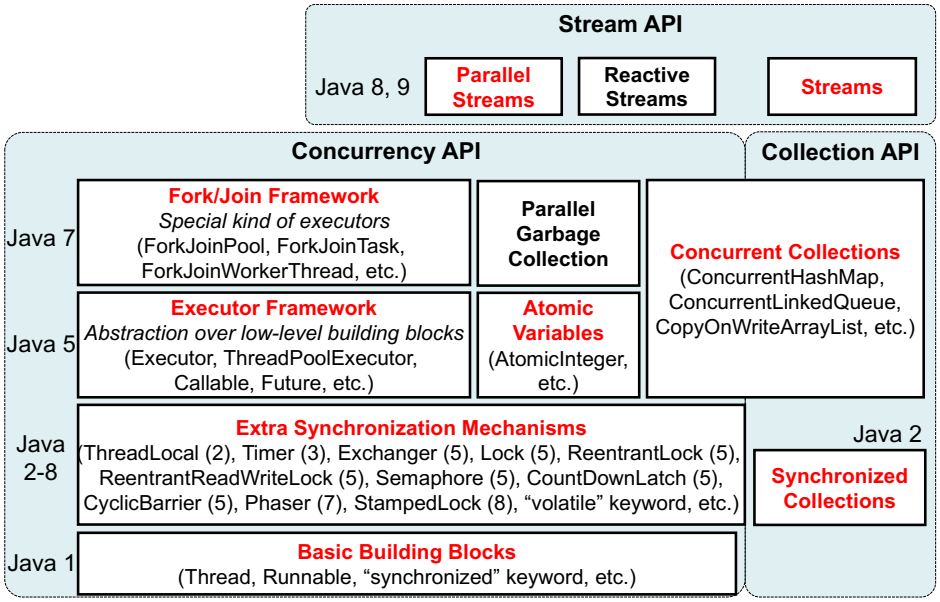
- Concurrent programming (multi-threading)
  - Mechanisms, data structures, libraries and frameworks for concurrency (multi-threading)
  - Concurrent object-oriented design patterns
    - e.g., Concurrent MapReduce, Producer-Consumer
  - Concurrency with LEs (incl. parallel streams)

Concurrency as a Part of SE? Yes!

- Concurrency was for special kinds of software engineers to develop special kinds of software.
  - Concurrency in programming: Since mid 60s
    - PL/I ("TASK" statement, '65), Concurrent Pascal ('75), Concurrent Smalltalk ('86), Objective-C ('86), Java ('96) , … C# ('02), Erlang ('06), etc.
  - Concurrency in OSes: Since 70s
    - Multix ('70), Mach (cthreads, '85) , NeXTSTEP (cthreads, '86), OS/2 ('87), Solaris libthread ('93), Windows NT ('94), pthreads (POSIX threads '95), Windows ('95), Mac OS X ('01), Linux ('03), iOS, Android, etc.
  - Early applications
    - Super computing and real-time computing in defense, aviation, financial trading (algorithmic and high freq. trades), etc.

- Concurrency is currentrly critical/important/useful for many software engineers to develop many kinds of software.
  - Not only "special" applications but also "normal" apps can enjoy, or even require, concurrency.
    - e.g., Web apps, smartphone/tablet apps
  - **Goals**: ***responsiveness*** and ***performance*** improvement
    - Better I/O handling
    - Multi-core CPUs

Concurrency API in Java



Course Work

- Lectures
- Homework
  - Reading
  - Coding (in Java)
- Individual project

Grading

- Homework (60%)
- Project deliverables (30%)
- Quizzes (10%)
  - Occasionally, at the beginning of lectures
  - May not have quizzes at all, depending on the class size
- No midterm and final exams.

- Some/many HWs have submission deadlines.
  - Do your best to meet the deadlines.
    - If you regularly meet them, you will get some extra points.
  - You can miss the deadlines. You DO NOT have to notify me that you will miss or have missed a deadline.
    - Up to a few days late: No problem. I favor better code than timeliness. Focus on your work, not making excuses to me.
    - Beyond that: Depends.
- In principle, you cannot replace your HW later on.

My Email Addresses

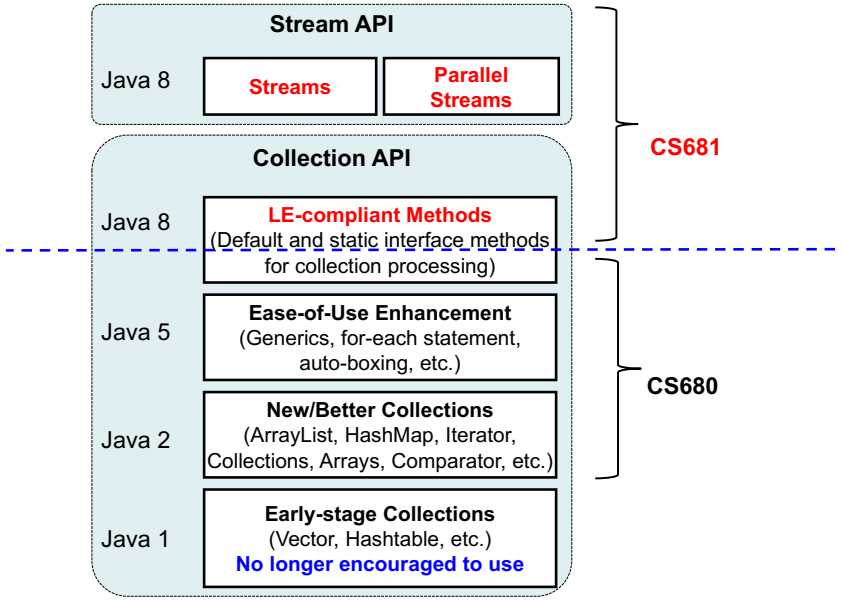
- Questions → **jxs@cs.umb.edu**
  - I regularly check this account.
- HW solutions → **umasscs680@gmail.com**
  - I occasionally check this account.
    - Once a week or so.

Your Email Address

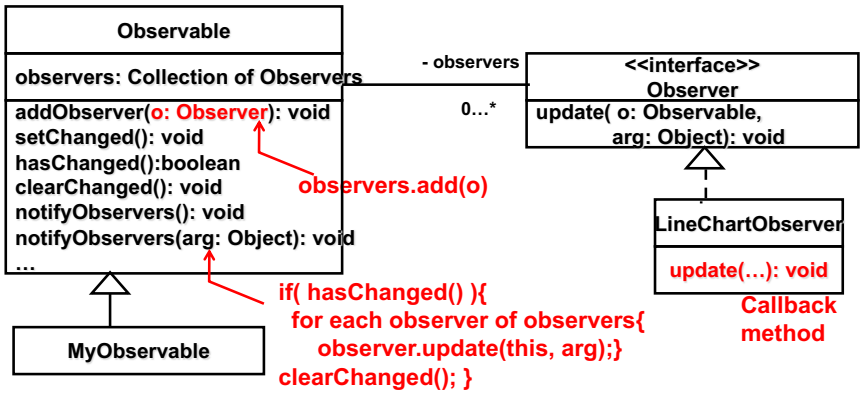
- Send your (preferred) email address to **jxs@cs.umb.edu** ASAP.
  - I will use that address to email you lecture notes, announcements, etc.
  - You will use that address to submit your HW solutions.

Toward Streams API

Collection and Stream APIs in Java



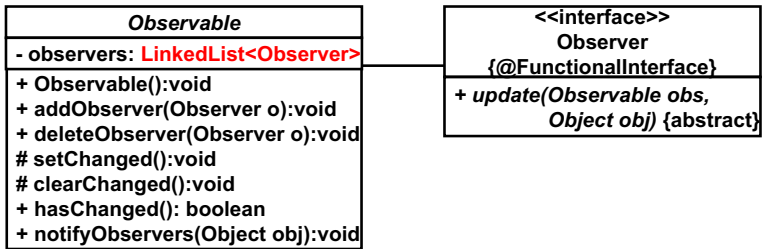
Recap: *Observer*



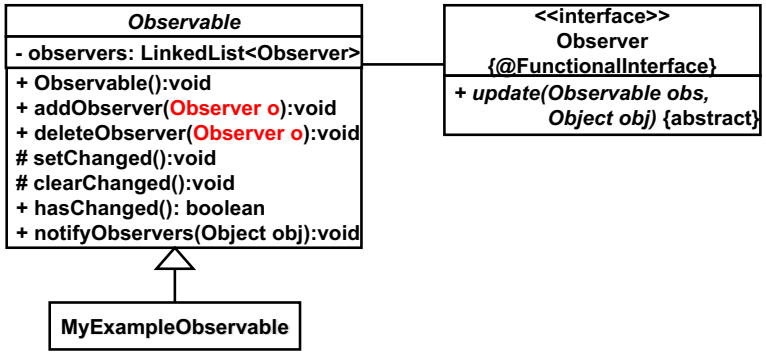

```
anObservable.addObserver( new LineChartObserver() );
anObservable.notifyObservers( ... );
```


HW 1: Implement *Observer* with LEs

- Define your own `Observable` (class) and `Observer` (interface)
  - DO NOT reuse `java.util.Observable` and `java.util.Observer`
  - Define `Observable` as an abstract class.
  - Define `Observer` as a functional interface.
    - `update()` as the only abstract method in that interface.
  - Implement all the methods that are available for `java.util.Observable`
    - c.f. Java API doc for expected behaviors/responsibilities for the methods
  - Use `LinkedList` to hold all registered observers.
    - c.f. CS680 slides on "ArrayList v.s. LinkedList"



- Use a lambda expression rather than defining a class that implements `Observer` and pass it to `addObserver()` and `deleteObserver()`.



- Use your `StockQuoteObservable` and `DJIAQuoteObservable`, which you implemented as the subclasses of `java.util.Observable` for HW 8 of CS680

- Example client code

```
StockQuoteObservable observable = new StockQuoteObservable(...);
observable.addObserver( (Observable o, Object obj)->
                              {System.out.println(obj);} );
observable.addObserver( ... );
observable.setQuote(...);
observable.notifyObservers();
```


- Submission Requirements
  - Follow the submission requirements in CS680.
    - Use Ant.
    - Submit .java files and an Ant script (e.g. build.xml).
    - Never send me binary code.
    - Avoid attaching a .zip file. Use another archive file format.
  - No need to do unit testing.
  - Due: September 25 (Tue) midnight

HW 2: Implement *Command* with LEs [Optional]

- Just like HW 1, you can implement the *Command* design pattern with LEs.
  - Define the `Command` interface as a functional interface.
    - Have `Command` define the abstract method `execute()`.
  - Implement the body of `execute()` as a LE.
    - Do not define classes that implement `Command`.

- Explain how to do that with a specific example.
  - You can use an example in CS680. Your own example is fine too.
  - Submit pseudo code or compilable/runnable code.