Hibernate

* Object Relational Mapping Framework
  + Designed to connect our entity classes to our database tables
* Pros
  + Abstracts away a lot of tedious boilerplate code from JDBC
  + Decouples our Java Application from the specific Database
  + Provides better query interfaces than in Java
* Setting up hibernate
  + Hibernate.cfg.xml
    - The main configuration file
      * Location of database
      * Username and password
      * Location of entity classes
* Core Interfaces
  + Configuration
  + SessionFactory
  + Session
  + Transaction
  + Query/Criteria
* Eager vs Lazy loading
  + Eager you get the object and all nested objects immediately in one query
  + Lazy you do not get nested objects until you actually call on the nested objects
* Annotations
  + @Entity
  + @Table
  + @Id
  + @GeneratedValue
  + @Column
  + Multiplicities
    - @OneToMany
    - @ManyToOne
    - @ManyToMany
* Object states in Hibernate
  + Transient
    - Not saved id of 0
  + Persisted
    - Object is saved to database non 0 id
    - Is currently associated with an active session
  + Detached
    - Object that was saved has a non 0 id
    - The session it was part of is closed
* Ways to query
  + Criteria
    - OOP approach
    - Create a criteria interface
    - Add restrictions then execute
  + HQL
    - OOP version of SQL
  + SQL
    - Not recommended
    - Ties you to the database
    - Removes the nice automatic object casting of hibernate

Dependency Injection

* An object is created with the inner objects its dependent on
  + Rather than the class determining the dependency
  + The dependency is passed in when it is created
* Pros
  + Makes our code very modular
  + Allows us to pass in mocks so that we can test a layer independent of the functionality of its dependencies (so test a service regardless of if the DAOs it needs actually work)

Mocking

* Mocking is the creation of a ‘fake’ object
  + Mockito.Mock(SomethingService.class)
* This fake object emulates a real one. Has the same type and method signatures but no implementation.
* We can stub the method calls.
  + When we invoke a method return a specific result
  + Mockito.when(object.method()).thenReturn(someValue)
* Mocking allows us to independently test our layers regardless of dependencies functionality

Integrated Tests

* When you run a test on a method that requires the dependencies to work
  + Call a service method with a real dao inside of it

Agile/Scrum

* Agile
  + It is a mindset
  + Emphasizes being flexible to change
  + Core tenets
    - Demos over documentation
    - Customer communication over contracts
    - Collaboration over processes and tools
    - Responding to change over following a plan
* Scrum
  + Implementation of Agile
  + Roles
    - Product Owner
    - Stakeholders
    - Scrum Master
    - Scrumlings (team members)
  + Sprint
    - An iteration of work
    - Usually about 2 weeks long
  + Scrum ceremonies
    - Day 1
      * Product Backlog (what you are going to work on this sprint)
      * User Story grooming
        + Creating/updating user stories
        + Point allocation to user stories

Rough estimate to how difficult this user story is to implement

* + - * Assign tasks to developers
    - Day 2-13
      * Daily Standup
        + Everyone meets together

What you are working on

If you need help

If you can offer help

Any problems

* + - Day 14
      * Sprint Retrospective
        + Everyone gets together

What was completed

What was not completed

Analyze any metrics

Discuss lessons learned

Launch pad to plan the next sprint

* + - * + Prepare a demo

BDD (Behavior Driven Development)

* Design from the perspective of the end users
* Pros
  + Prevents you from coding features that aren’t necessary
  + Can give insights on to what your code needs to be able to handle
    - What your entities contain etc…
  + Design a UI that feels right to the user
* TDD
  + Write a test and then create code to pass that test
* BDD
  + Writing acceptance criteria and creating an application to pass that acceptance criteria
* User Story
  + A scenario that an end user would do on the application
  + As a (end user) I want to (something) so that (reason)
* Acceptance criteria
  + The actual steps that a person must perform to fulfill that user story
    - Should be pretty specific
    - Click on this button, or be on this page

Cucumber

* BDD framework
* Written in Gherkin syntax
  + Gherkin keywords
    - Feature
    - Scenario
    - Given
    - When
    - Then
* Our cucumber Scenarios can be automated with Selenium to make e2e tests

Selenium in Java

* Selenium automates browsers
* IT IS NOT A TESTING FRAMEWORK
* Selenium WebDriver is the core of selenium
  + This WebDriver allows you to interact with the browser
  + Need a specific webdriver for each browser
* Page Object Model
  + Create a class whose instance variables relate to web elements on the page
  + Create a POM in selenium by using the pageFactory method
  + Fields can be annotated in a POM to populate them via different @FindBy
    - Id
    - className
    - name
    - css
    - xpath (directions to an element)
    - link text (get element by hyperlink)
* Waiting in Selenium
  + Implicit waits
  + Explicit waits
  + Thread.sleep (not encouraged)

Creating e2e tests with Cucumber + Selenium +JUnit

* Feature files
  + Hold the scenarios that you are trying to automate
* Step Implementations
  + Classes that have methods that are the selenium implementation a scenario step
* JUnit Runner File
  + A JUnit test case that you can actually run
  + @CucumberOptions(features =”someplace”, glue “stepimplementations” )

Continuous Integration

* CI is the process by which multiple developers will work together to build an application
* The developers share a remote repository
* They will push their code into that repository
* Other developers will review their code
  + If it is good it will be incorporated into a main branch for other people to add new features to
  + If it is bad it will be rejected to make changes
* Steps to adding a feature
  + Make a branch off of the main branch
  + Make your coding changes
  + Push to update your branch in the remote repository
  + Make a pull request
  + Hopefully whoever reviews your code approves of it and merges your branch into the main
* Continuous Delivery
  + Creates a deliverable
* Continuous Deployment
  + Runs the deliverable so that end users can immediately use the application

Jenkins

* Continuous integration tool
* Jenkins can be configured to automatically build and run tests on our github repository
* This allows developers a way to see if their code works
  + Also allows other developers to verify that their code works before integrating it back into the main branch
* Jenkins terminology
  + Jenkins Job – A specific project/repository that we are watching Ex BookAPI
  + Build – An attempt at getting the source code and building/testing the application
  + Webhook- A notification sent by github that a change has been made in the repository and that Jenkins so perform another build