

Unofficial Guide to Interning at Wikia

...well, at least on the Services Team

July 2015

Contents

1	What is this?	4
2	How to get started coding	5
2.1	IntelliJ	5
2.1.1	Setting up IntelliJ	5
2.1.2	I've never used an IDE. What is this and why can't I just use a text editor?	5
2.2	What is Pandora?	6
2.3	What is Helios?	6
2.3.1	What is Go?	6
2.4	What is MediaWiki?	6
2.4.1	What big is it?	6
2.4.2	How does it relate to us?	6
3	Stand-up? Backlog grooming? Demo's? What are all these meetings!	7
3.1	What is Scrum?	7
3.2	What are Sprints?	7
3.3	What are Stories?	8
3.4	What is JIRA?	8
3.5	What are problem designations?	8
3.6	What are Stand-Ups?	8
3.6.1	Why do they sometimes happen at 8AM?	8
3.7	What are the roles?	9
3.7.1	What do team members do?	9
3.7.2	What does the Product Owner do?	9
3.7.3	What is the Scrum Master?	9
4	So many tools! How do I keep track?	10
4.1	What is Dropwizard?	10
4.1.1	What is REST?	10
4.1.2	What is HTTP?	10
4.2	Gradle	12
4.2.1	What is a build script?	12

4.3	Jrebel	12
4.4	Project Lombok	12
4.5	Guice	13
4.6	Guava	14
4.7	Swagger	14
5	What is Git and why do we use it?	15
5.1	What is Git?	15
5.2	Why version control?	15
5.3	What is a repository?	16
5.4	What is my local repository?	16
5.4.1	I want to start coding! How do I set up my local repository?	16
5.4.2	I got my first ticket! What do I do?	16
5.4.3	I just finished coding! What do I do?	17
6	What is TunnelBlick?	18
6.0.1	What is a VPN?	18
6.1	What is Jenkins?	18
6.2	What is Vignette?	20
7	How do I debug?	21
7.1	IDE Debugging	21
7.1.1	Why use an IDE debugger?	21
7.1.2	What are breakpoints?	22
7.1.3	How do I do all this in IntelliJ?	23
8	How do I test my code?	24
8.1	What are unit tests?	24
8.2	What are end-to-end / integration tests?	24
8.3	What is cobertura?	25
8.3.1	What is line coverage?	25
8.3.2	What is branch coverage?	26
8.3.3	Why should I care about the difference between line coverage and branch coverage?	26
8.4	What are health checks?	26
8.5	Mocking	26
8.5.1	Mockito	27
9	Scrum Continued; How do I contribute to meetings?	28
9.1	Everyone knows so much! What ever will I do!	28
9.2	How do I gain an understanding of all these moving parts?	28
9.2.1	Backlog Grooming	28
9.2.2	Sprint Planning	29
10	Payroll? Hours? What is going on!	30
11	Writing your first API; Rambling Advice	31

<i>CONTENTS</i>	3
12 Mistakes I've made	32

Chapter 1

What is this?

When you join a new team, you have to tackle a huge repository of existing code and a ton of new tools. On top of that, the team will likely have established guidelines for how new code should be written, protocols for the process of committing new code, and a bunch of other unique development practices.

The purpose of this guide is to break down the moving parts of the Wikia Services Team into easy-to-understand components. Between IDE's and Mocking Frameworks and Scrum, there will likely be concepts that you are familiar with, as well as concepts that you've never heard of.

This guide is meant to be a living document. As time passes, especially in the tech world, tools change. Feel free to update this document as it becomes outdated and new developments occur.

Chapter 2

How to get started coding

2.1 IntelliJ

2.1.1 Setting up IntelliJ

The Services Team uses IntelliJ as their Java IDE. The Pandora repository wiki (github.com/Wikia/pandora/wiki/Getting-Started) includes a step-by-step guide on setting up Java and IntelliJ on your machine.

2.1.2 I've never used an IDE. What is this and why can't I just use a text editor?

If used properly, an IDE can save you a ton of time. How, you ask?

- **Code Navigation:** `Command + click` on a function or variable
- **Code Completion:** IntelliJ auto-completes classes and method names as you code. It also expands acronyms (i.e. NPE for "NullPointerException" or "No Page Error.")
- **Code Generation:** `Ctrl + return` can generate getters and setters or implement methods from an interface.
- **Code Coloring:** IntelliJ includes the standard keyword, string, and variable coloring. It also colors member variables, local variables, and parameters.
- **Refactoring:** IntelliJ is extremely effective for mistake-free renaming, even replacing setters, getters, and string usages,
- **Dependency importing:** When relying on a third party library that you have the source for, you can navigate to the code easily for reference.
- **Debugging:** IntelliJ has a built-in debugger, which we will discuss in-depth in the debugging chapter.

2.2 What is Pandora?

The Pandora repository, located at github.com/Wikia/pandora, is where all of Wikia's services are located. All of our Java API's are located in this repo. Pandora also includes all of the tools and support libraries that are central to these services as well as all the tests for these services. As stated on repo's wiki, the purpose of Pandora is to:

- Provide a simplified guidelines compliant content interface to articles
- Enable rapid prototyping and iteration of the above
- Hide the complexity of MediaWiki
- Decouple content which will enable HyperMedia API designs, better caching, and the ability to fan-out requests.

2.3 What is Helios?

The Helios repository, located at github.com/Wikia/helios, is the authentication service for MediaWiki. It is written in Go. We've decided to treat authentication as user creation instead of just registration.

2.3.1 What is Go?

Go is a static-typed language with syntax that is loosely similar to X. Go emphasizes good composition while delivering concurrency with the static execution speed of C. Go supports garbage collection, type safety, dynamic-typing, among other features.

2.4 What is MediaWiki?

MediaWiki is a large open-source project that was started in 2004. It is written in PHP and provides a lot of core functionality for content management. Wikipedia and Wikia run on MediaWiki.

2.4.1 What big is it?

MediaWiki is 5 million lines of code, which is part of the reason that it is sometimes extremely difficult to dive right into. These 5 million lines are:

- 1/3 core MediaWiki
- 1/3 core extensions
- 1/3 Wikia extensions

2.4.2 How does it relate to us?

As a company, we have decided to slowly move off of MediaWiki and onto our own independent services. Currently, several other teams do a lot of development in MediaWiki, but the services team does not. Our company's long-term goal is to eventually stop creating extensions for MediaWiki and only create services independent of MediaWiki. That is what the services team is working on.

Chapter 3

Stand-up? Backlog grooming? Demo's? What are all these meetings!

As you likely noticed from your calendar, the Services Team has a lot of meetings. This is because we use the Agile Process. The Agile Process is a group of software development methodologies that are meant to make our collaboration and self-organization as effective as possible.

3.1 What is Scrum?

Scrum is a subset of Agile. It is a particular set of practices that must be followed for a process to be consistent with the Agile framework. Scrum emphasizes maximizing the amount of productive time available for useful work to get done.

3.2 What are Sprints?

Sprints are the amount of work to do in a set amount of time. For the Services Team, every sprint starts on a Friday and lasts two weeks. Before the start of a new sprint, we have **Backlog Grooming** meeting, where we discuss the tasks that are currently in the backlog and comment on how we will tackle them. Then, we have a **Sprint Planning** meeting where we choose the number of tasks that we want to get done in the upcoming sprint. At the end of that sprint, we have a **Demo**, which is a slide presentation that shows off the work that is done. Then, we have a **Retrospective** meeting, where we look back on the sprint and consider what went well, what didn't go well, and what we will change next sprint. The Retrospective is very important to Scrum because it allows the team to quickly recognize what went wrong and fix it. If a task

didn't go as planned or there is a problem with our approach, we can quickly pivot towards a new set of tasks and focus.

3.3 What are Stories?

Stories is a term for an overarching task that needs to get done. Stories only describe *what* needs to get done, not *implementation*. Stories get broken down into **tickets**, which are smaller components of the larger task.

3.4 What is JIRA?

Jira is the Project Management Software we use to keep track of our Sprints. You can use it to view our sprint progress, look at our stories and tickets, and create new stories and tickets at <https://wikia-inc.atlassian.net/>.

3.5 What are problem designations?

When a problem is discovered, it is given an urgency designation:

- P1: This is an **URGENT** problem. The site is down! Drop everything and fix this immediately.
- P2: Something major is broken; fix this within 48 hours.
- P3: Something is broken, fix this within 2 weeks.
- P4: Everything else goes here!

3.6 What are Stand-Ups?

Stand-Ups are daily meeting that we have to discuss the following:

- What did I accomplish yesterday?
- What will I do today?
- What obstacles are impeding my progress?

We call them "Stand-Ups" because the members are typically standing. The discomfort of standing is intended to keep these daily meetings concise.

3.6.1 Why do they sometimes happen at 8AM?

Since Wikia has an office in Poznan, some of the Services Team members are in Poland. In order to maintain communication, two Stand-Up meetings a week are held at 8AM via BlueJeans (our video-conferencing platform) between the US and Poland team.

3.7 What are the roles?

There are three main roles in the Scrum process. Team members, Product Owner, and Scrum Master. As of July 2015, Geoff Benson is both the Product Owner and the Scrum Master. The rest of the services team are team members.

3.7.1 What do team members do?

Team members are the majority of the Services Team. They can do whatever they want to get stories done and have flexibility in their approach.

3.7.2 What does the Product Owner do?

The Product Owner is the one who decides what order things need to get done in. He can make overarching decisions regarding the sprint.

3.7.3 What is the Scrum Master?

The Scrum Master helps make sure the process runs smoothly. The Scrum Master serves as an adviser on the best way to tackle problems and makes recommendations based on past experience.

Chapter 4

So many tools! How do I keep track?

Don't worry! Here is a brief overview of the coding tools that the Services Team uses:

4.1 What is Dropwizard?

The Services Team uses a collection of different libraries in our web services. Dropwizard brings all these libraries together and does the wiring for our **RESTful services**.

4.1.1 What is REST?

REST stands for Representational State Transfer, and it is a simple way to organize interactions between independent systems. REST is an architecture style for designing networked applications. RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.

4.1.2 What is HTTP?

HTTP is the protocol, or set of rules, that allows for sending documents back and forth on the web. This protocol determines which messages can be exchanges and which messages are appropriate replies to others.

In HTTP, there are two different roles: **server** and **client**. The client always initiates the conversation and the server replies. HTTP messages are made of a header and a body?

What is a HTTP message body?

The body of an HTTP message contains the data you want to transmit over the network.

What is a HTTP message header?

The header of an HTTP message contains metadata, including instructions on how to use the data in the body. In the case of a request, it contains important HTTP methods.

What is a URI?

URIs, or Uniform Resource Identifiers, are how you identify the things that you want to operate on. For example, a web page is a type of resource and all webpages can be accessed using a specific type of URI called an URL.

What are HTTP verbs?

Each HTTP request specifies a certain HTTP verb, or method, in the request header. This is the first all caps word in the request header. For example,

`GET HTTP/myresource` means the `GET` method is being used.

There are four HTTP verbs: `GET`, `PUT`, `DELETE`, `POST`. HTTP verbs tell the server what to do with the data identified by the URI.

GET

is the simplest type of HTTP request method; browsers use this method each time you click a link or type a URL into the address bar. It instructs the server to transmit the data identified by the URL to the client. Data should never be modified on the server side as a result of a `GET` request.

PUT

requests are used when you wish to create a resource identified by a URL. For example,

`PUT /client/connie`

might create a client, `connie`, on the server.

DELETE

`DELETE` requests are used to delete the resource identified by the URL of the request. For example, if the user did something undesirable and needed to be removed from the system, we could: `DELETE /client/connie`

POST

POST requests are used to update a resource identified by the URL. Unlike all the other HTTP methods, POST requests are not necessarily idempotent.

```
DELETE /client/connie
```

4.2 Gradle

Gradle is an extensive build tool and dependency manager for programming projects. Gradle also provides build-by-convention support for many types of projects including Java, Android and Scala.

4.2.1 What is a build script?

Your build scripts are how you tell Gradle how to build your application. The application itself can be represented by many Gradle projects. A Gradle project does not represent your application as a whole, but instead can represent many different things. It could represent a library jar file you want to build for your project, it could be a distribution zip file, or it could just represent something you want done with the app, such as deploying it to a server.

One benefit of having multiple projects in a single application is that it is simple to have separate dependencies or to build tasks for the different application parts.

How does Gradle know what counts as a project in your application? It will create a project for each 'build.gradle' file in your application. On the previous example, both the Mobile and Wear directories in a Wear application would have a 'build.gradle' file letting Gradle know that it is a separate project. For each project in your build, Gradle will create a Project object. This object allows you to access Gradle features such as adding tasks, properties and dependencies. Properties are defined in the 'build.gradle' file, typically in an extra block.

4.3 JRebel

JRebel instantly reloads changes to Java code and saves developers time. It works as a java agent on top of the JVM. It looks for changes in the folder with your compiled classes. If you compile a class, JRebel notices the change and compares the differences between class definitions. Then at the bytecode level it adds/removes/replaces fields/method bodies/etc. without restarting the JVM.

The JRebel Activation Code can be provided by Nelson.

4.4 Project Lombok

Lombok is a tool that generates code for you, but not in the way that your IDE generates it. Your IDE generates getters and setters, or equals and

more, and then puts the code in your uncompiled class file. Lombok generates the same thing, but does it in the class file; all you need to do is add some annotations like `@Setter`, `@Getter`, `@Builder`, `@Data`, etc. to your code and Lombok generates the setters, getters and so on for you. At the same time, you can override whatever is generated by just writing the method yourself instead.

Below is an example of using lombok to generate getters and setters using the `@Data` annotation. `@Builder` creates a builder for this class, while `@AllArgsConstructor` and `@NoArgsConstructor` create constructors:

```
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Builder
@AllArgsConstructor
@NoArgsConstructor

public
@Data
class DefaultTask implements Task{
    @JsonProperty
    private String taskMessage;

    @JsonProperty
    private String contentType;

    @JsonProperty
    private String delayTime;

    @JsonProperty
    private String uniqueIdPrefix;

    @JsonProperty
    private String uniqueId;

    @JsonProperty
    private String deduplicateId;

    public String toMessage() {
        return taskMessage + ", id: " + uniqueId;
    }
}
```

4.5 Guice

Guice is a dependency injector, or a way to inject objects into your application without having to explicitly reference those objects. For example:

```
1 @Inject
2 public class MyClass(InjectObject injectedobject){
3     this.injectedobject = inject object;
4 }
```

The above code would provide an instance of `InjectObject` into my code, as long as I had the following code in my module (module is Google Guice's term for where you create your bindings and define your dependencies):

```
1 @Provides
2   InjectObject provideInjectObject(){
3       InjectObject standardInjectObject = new InjectObject(
4           standardParameters);
5       return standardInjectObject;
6   }
```

4.6 Guava

Guava is a collection of several of Google's core libraries used for Java-based projects. It includes libraries for collections, caching, common annotations, string processing, I/O, and more. We often use Guava in our Java repo, Pandora.

4.7 Swagger

Swagger is a framework for defining your API. For our Java code, we can use Swagger annotations throughout our code. Swagger uses these annotations to automatically create JSON definitions. These definitions can be used to generate documentation as well as generate code.

Chapter 5

What is Git and why do we use it?

5.1 What is Git?

Git is a version control system, or VCS. Specifically, it is a distributed VCS.

5.2 Why version control?

Nobody likes to lose their work. Maybe you're the type of person who constantly pounds the "Save" button on your documents or stores all his files in the cloud. VCS address that paranoia. It helps you track changes to your code.

Version Control Systems provide a documented history. As you make changes to your code, you will eventually reach a good stopping point. To save the current status of your project, you still mark the files you changed and "commit" your changes to the VCS. The VCS will store exactly how your code looked at that moment. If you make more changes later, but look back and decide that your new changes are bad, you can always restore your code to a previous commit.

Version Control Systems also allow teams to work together, all using the same files. If you've ever tried to edit a document with multiple people (without using Google Drive), you've likely run into issues where somebody edits an older version and is stuck adding their changes to the current version. Or worse, they send out their version and overwrite all the new changes. Using git, there is one remote repository that everyone on our team can work on, make locally make changes to, commit their changes to, and pull changes from when the repository is updated.

5.3 What is a repository?

A repository, or "repo", is a directory where your projects can live. Wikia/-pandora is a remote repository on Github. You will also have a local repository on your computer.

5.4 What is my local repository?

Your local repository is a copy of all the files in the remote repository on your own machine. Your local repository

5.4.1 I want to start coding! How do I set up my local repository?

Download git for OS X (<https://git-scm.com/downloads>)

Create your working copy of Wikia/pandora:

1. Navigate to the folder on your machine where you want to store your local repo.
2. Use the terminal for the following command: `git clone https://github.com/Wikia/pandora.git`
You have a working copy of the entire pandora repository!

5.4.2 I got my first ticket! What do I do?

Make a branch with your ticket name.

This command creates a local copy of all the files in master.

```
>>> git branch SERVICES-000
```

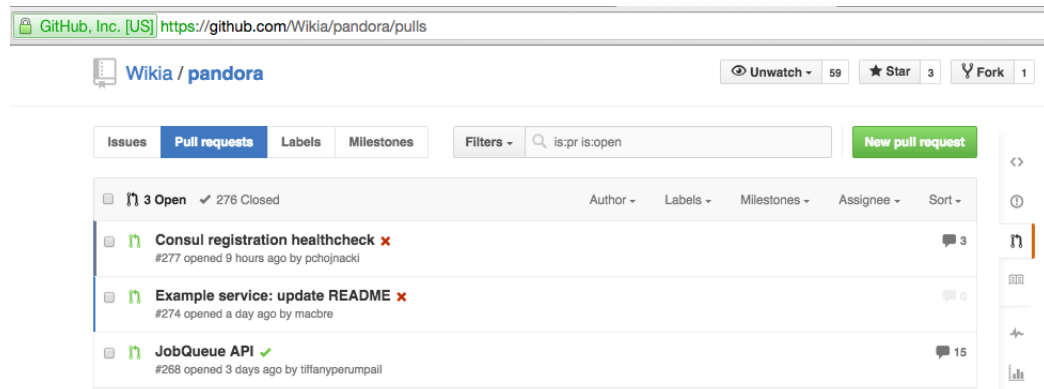
This command allows you to start working on your branch.

```
>>> git checkout SERVICES-000
```

Now, you can start making your changes on this branch!

5.4.3 I just finished coding! What do I do?

Once your changes are finalized on your branch, push your changes and then navigate to the github website to make a pull request. Once you've made your pull request, Jenkins will run unit and integration tests to confirm that your code has not broken anything. This may take a few minutes. If these checks don't all pass, you will need to fix your code. See "What is Jenkins" for more information.



Chapter 6

What is TunnelBlick?

Tunnelblick is a free, open source graphic user interface for OpenVPN on OS X. It provides easy control of OpenVPN client and/or server connections.

6.0.1 What is a VPN?

A VPN or Virtual Private Network is a network connection that enables you to create a secure connection over the internet to private networks.

Our VPN provides employees with secure remote access to our company network. By connecting to the company's network, an individual employee can access all the company's resources and services on-site as well as remotely.

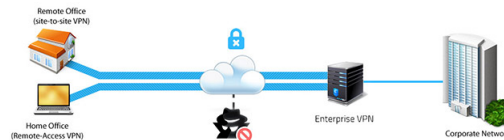
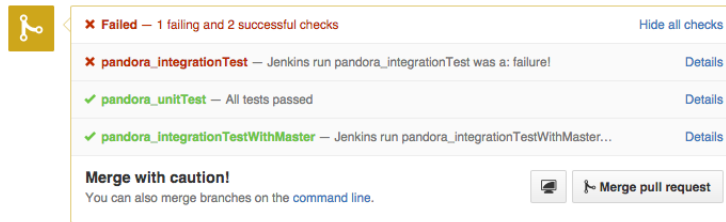



Figure 6.1: VPN connecting to a Corporate Network

6.1 What is Jenkins?

Jenkins is a continuous integration tool written in Java. Whenever you push to the Pandora repo, Jenkins will run all the unit and integration tests as well as health check tests to make sure that the new changes have not broken anything. If your Jenkins tests don't pass, here is what to do:

- Click details on the failed test:

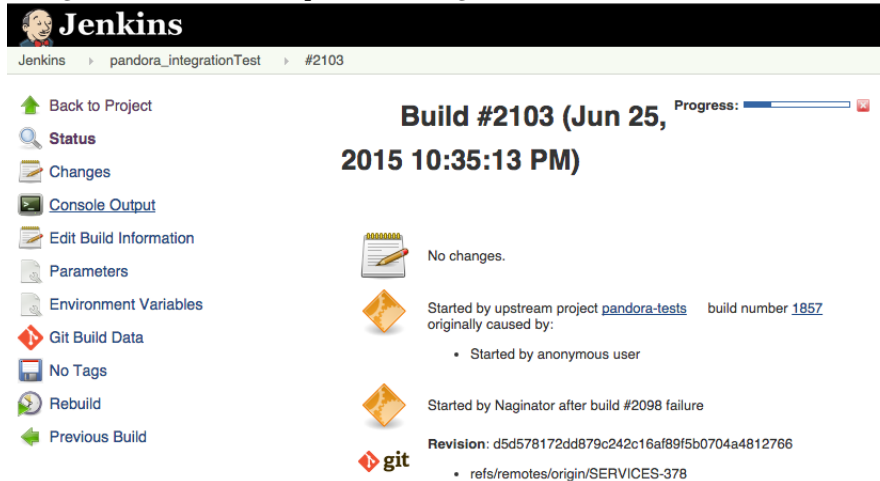



✖ Failed — 1 failing and 2 successful checks
[Hide all checks](#)

- ✖ **pandora_integrationTest** — Jenkins run pandora_integrationTest was a: failure! [Details](#)
- ✔ **pandora_unitTest** — All tests passed [Details](#)
- ✔ **pandora_integrationTestWithMaster** — Jenkins run pandora_integrationTestWithMaster... [Details](#)

Merge with caution!
 You can also merge branches on the [command line](#).
 [Merge pull request](#)


- Navigate to Console Output on the right hand bar:



Jenkins
 pandora_integrationTest #2103

[Back to Project](#)
[Status](#)
[Changes](#)
[Console Output](#)
[Edit Build Information](#)
[Parameters](#)
[Environment Variables](#)
[Git Build Data](#)
[No Tags](#)
[Rebuild](#)
[Previous Build](#)

Build #2103 (Jun 25, 2015 10:35:13 PM)

Progress: 

No changes.

Started by upstream project [pandora-tests](#) build number [1857](#) originally caused by:

- Started by anonymous user

Started by Naginator after build #2098 failure

Revision: d5d578172dd879c242c16af89f5b0704a4812766
 • [refs/remotes/origin/SERVICES-378](#)

- Scroll down until you find BUILD FAILED in bright red to locate where the error occurred:

```

13:40:06
13:40:06 * What went wrong:
13:40:06 Execution failed for task ':testing:end-to-end:integrationTest'.
13:40:06 > Unable to connect to the child process 'Gradle Test Executor 17'.
13:40:06 It is likely that the child process have crashed - please find the stack trace in the build log.
13:40:06 This exception might occur when the build machine is extremely loaded.
13:40:06 The connection attempt hit a timeout after 120.0 seconds (last known process state: STARTED, running:
true).
13:40:06
13:40:06 * Try:
13:40:06 Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log
output.
13:40:06
13:40:06 BUILD FAILED
13:40:06
13:40:06 Total time: 24 mins 14.095 secs
13:40:07 Build step 'Invoke Gradle script' changed build result to FAILURE
13:40:07 Build step 'Invoke Gradle script' marked build as failure
13:40:07 [/usr/bin/curl, -s, -X, POST, -H, Authorization: token 4a86734f7adad549f412e7107c4493fde1c27f12, -d,
{"state":"failure","context":"pandora_integrationTest","description":"Jenkins run pandora_integrationTest was a:
failure!","target_url":"http://qa-s2.wikia-
prod:8080/job/pandora_integrationTest/2075/"}],
https://api.github.com/repos/Wikia/pandora/statuses/67f051f581eae6f7b11aa7039ec376c3c4f3a467]
13:40:08 {
13:40:08 "url": "https://api.github.com/repos/Wikia/pandora/statuses/67f051f581eae6f7b11aa7039ec376c3c4f3a467"

```

6.2 What is Vignette?

Vignette is a thumbnailer. It's a proxy to imagemagick, which you can read more about at <http://www.imagemagick.org/script/index.php>.

Chapter 7

How do I debug?

The first three things I double-check when my code isn't working in the office are:

- Am I connected to VPN?
- Have I configured my environment variables and argument?
- Have I synced Gradle?

If none of these checks reveal the problem, then we'll have to delve into using the IntelliJ Debugger.

7.1 IDE Debugging

7.1.1 Why use an IDE debugger?

You may be thinking, why do I need a debugger? I can already look at trace messages in my code. What more do I need!

Here are some benefits of using an IDE debugger:

- Add **breakpoints** to temporarily suspend execution of your program at a certain point (No more print-statement debugging!)
- Be able to **watch variables** and see when they change
- Conditional breakpoints; stop the application only in exceptional circumstances to allow you to analyse the stack and variables.
- View the call stack at any point in time, giving you a context for your current stack frame.
- Change variable values while the program is running
- Be able to skip or repeat sections of code, to see how the code will perform. This allows you to test out theoretical changes before making them.

- Alert you when certain exceptions are thrown, even if they are handled by the application.

7.1.2 What are breakpoints?

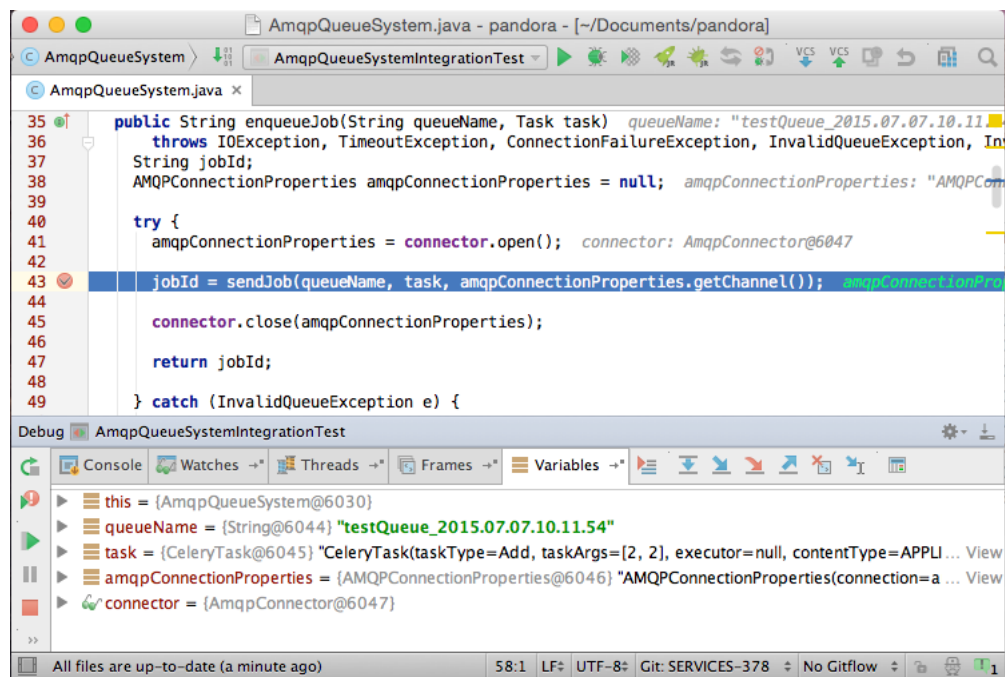
A breakpoint is a signal that tells the debugger to temporarily suspend execution of your program at a certain point. When execution is suspended at a breakpoint, your program is said to be in break mode. Entering break mode does not terminate or end the execution of your program. Execution can be resumed at any time.

You can think of break mode as being like a timeout. All the elements remain, functions, variables, and objects remain in memory, for example, but their movements and activities are suspended. During break mode, you can examine their positions and states to look for violations or bugs. You can make adjustments to the program while in break mode. You can change the value of a variable, for example. You can move the execution point, which changes the statement that will be executed next when execution resumes.

Breakpoints are a powerful tool for pinpointing where a problem is occurring in your code. Rather than stepping through your code line-by-line or instruction-by-instruction, you can allow your program to run until it hits a breakpoint, then start to debug. This speeds up the debugging process enormously. Without this ability, it would be virtually impossible to debug a large program.

7.1.3 How do I do all this in IntelliJ?

Below is an example screenshot of using the debugger in IntelliJ. By clicking on the gray section to the left of your code, you can add breakpoints, which are represented by red circles (see below). In order to run your code in debug mode, click the green "bug" icon next to the standard run icon. Then, once you hit a breakpoint, you can navigate to the variable tabs on the lower portion of your window to view the current state of the variables.



Chapter 8

How do I test my code?

When you are developing your code, you can always manually test your code. However, manual tests are the worst kind of tests because there is no log of those tests and they are difficult to reproduce. Therefore, the team creates thorough unit and end-to-end tests for all new code.

8.1 What are unit tests?

A unit test is a test written by the programmer to verify that a relatively small piece of code is doing what it is intended to do. It should have **no dependencies** on code outside of the section being tested. They are narrow in scope, they should be easy to write and execute, and their effectiveness depends on what the programmer considers to be useful. The tests are intended for the use of the programmer, they are not directly useful to anybody else, though, if they do their job, testers and users downstream should benefit from seeing fewer bugs.

Part of being a unit test is the implication that things outside the code under test are mocked or stubbed out. Unit tests shouldn't have dependencies on outside systems. They test internal consistency as opposed to proving that they play nicely with some outside system.

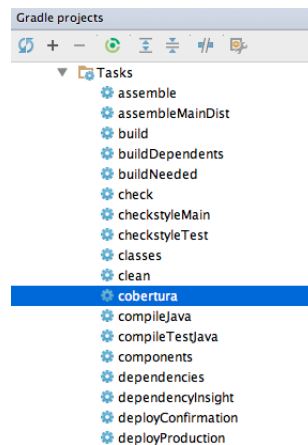
8.2 What are end-to-end / integration tests?

An end-to-end / integration test is done to demonstrate that different pieces of the system work together. Integration tests cover whole applications, and they require much more effort to put together. They usually require resources like database instances and hardware to be allocated for them. The integration tests do a more convincing job of demonstrating the system works (especially to non-programmers) than a set of unit tests can, at least to the extent the integration test environment resembles production.

These tests are referred to as end2end tests in the Helios repo and integration tests in the Pandora repo, but they are the same thing.

8.3 What is cobertura?

Cobertura is a free Java tool that calculates the percentage of code accessed by tests. It can be used to identify which parts of your Java program are lacking test coverage. You can run cobertura through gradle, by clicking on the command shown below.



After running this command, you can access the report generated in a reports/cobertura folder. An example report is provided below:

Coverage Report - All Packages

Package /	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	38	50%	62%	1.078
net.iww.fdb5	3	77%	91%	1
net.iww.fdb5.logic	3	57%	73%	0
net.iww.fdb5.lucene	2	45%	58%	0
net.iww.fdb5.model	5	66%	80%	1.286
net.iww.fdb5.model.ibatis	9	70%	N/A	1
net.iww.fdb5.model.rss	1	69%	0%	1.222
net.iww.fdb5.struts	13	21%	29%	1.571
net.iww.taglibs.tooltip	2	0%	N/A	1.2

Report generated by [Cobertura](#) 1.7 on 3/13/06 12:52 PM.

8.3.1 What is line coverage?

Line coverage measures how many statements you took (a statement is usually a line of code, not including comments, conditionals, etc). The line coverage report represents the percent of lines executed by the test run.

8.3.2 What is branch coverage?

Branch coverages checks if you took the true and false branch for each conditional (if, while, for). You'll have twice as many branches as conditionals. The branch coverage report represents **the percent of branches executed by the test run**.

8.3.3 Why should I care about the difference between line coverage and branch coverage?

Consider the following example:

```
1 public int getNameLength(boolean isCoolWikiaIntern) {  
2     User user = null;  
3     if (isCoolWikiaIntern) {  
4         user = new Connie();  
5     }  
6     return user.getName().length();  
7 }
```

If you call this method with `isCoolWikiaIntern` set to true, you get 100% line coverage. That sounds great, until you realize that there will be a `NullPointerException` if you call with `isCoolUser` set to false. However, you have 50% branch coverage in the first case, so you can see there is something missing in your testing (and often, in your code).

8.4 What are health checks?

Every service has a health check. These tests are run periodically on the running application. They are meant to confirm that the service is able to successfully connect to any databases, servers, or other systems that it is dependent on. Health checks should connectivity, not functionality. Therefore, health checks should be fairly straightforward to create.

8.5 Mocking

Mocking is primarily used in unit testing. An object under test may have dependencies on other (complex) objects. To isolate the behavior of the object you want to test you replace the other objects by mocks that simulate the behavior of the real objects. This is useful if the the real objects are impractical to incorporate into the unit test.

n short, mocking is creating objects that simulate the behavior of real objects.

8.5.1 Mockito

Mockito is an open source testing framework for Java. Mockito allows the creation of mock objects in our unit tests. Below is an example of a unit test that uses Mockito.

```
public class PricingServiceTests {
    private static final String SKU = "3283947";
    private static final String BAD_SKU = "-9999993434";
    private PricingService systemUnderTest;

    @MockitoAnnotations.Mock
    private DataAccess mockedDependency;

    @Before
    public void doBeforeEachTestCase() {
        MockitoAnnotations.initMocks(this);
        systemUnderTest = new PricingServiceImpl();
        systemUnderTest.set.DataAccess(mockedDependency);
    }

    @Test
    public void getPrice() throws SkuNotFoundException {
        stub(mockedDependency.getPriceBySku(SKU)).toReturn(new BigDecimal(100));
        final BigDecimal price = systemUnderTest.getPrice(SKU);
        // Verify state
        assertNotNull(price);

        // Verify behavior
        verify(mockedDependency).getPriceBySku(SKU);
    }
}
```

Chapter 9

Scrum Continued; How do I contribute to meetings?

Whenever you are the newest joining member of a group, the rest of the members will have accumulated knowledge about the project that you are not privy to. Add being an intern fresh out of college on top of that, and team meetings can seem pretty intimidating.

Instead of dwelling on these fears, let's talk about how to mitigate them. This chapter will specifically focus on Scrum-style meetings, but the sentiments can be applied to joining any pre-existing team.

9.1 Everyone knows so much! What ever will I do!

No one in the meeting knows everything about all the moving parts. Usually, the team members have specialized knowledge on different aspects of the project. Use that to your advantage! If you don't know something, **ask**. Odds are you aren't the only one who doesn't know all the ins and outs of the current task.

Additionally, explaining a project component in detail can be helpful to the developer as well. Being able to clearly and concisely explain what you are doing and why you are doing it is a major component of development.

9.2 How do I gain a deeper understanding of the project?

9.2.1 Backlog Grooming

During backlog grooming, the team comes together to look over the tickets in the backlog. They add new stories and epics, extract stories from existing

epics, and estimate effort for existing stories. Basically, this is your chance to get up to date on everything you missed before you joined the team! Pay close attention during these meetings to understand what every story consists of and ask questions about anything you don't understand.

9.2.2 Sprint Planning

Every iteration begins with the sprint planning meeting. At this meeting, the Product Owner and the team negotiate which stories a team will tackle that sprint. During this meeting, you have a chance to fully understand all of the stories that the team will be working on this sprint and be up to date with what everyone on the team is doing.

When you estimate the amount of effort to assign each story, you will gain a better understanding of all that moving parts that the story involves.

Take advantage of this meeting! It will help you stay up-to-date on all the upcoming stories and contribute meaningfully in future meetings.

Chapter 10

Payroll? Hours? What is going on!

Interns are on payroll, so we need to log our hours every two weeks. Wikia uses ADP, <https://workforcenow.adp.com>. HR can provide you with the details on registration. Once you are registered, you will be able to log into ADP and enter your hours. Once they are entered, make sure it approve your timesheet and have your manager approve it as well.

Chapter 11

Writing your first API; Rambling Advice

When writing an interface, think about what the client needs to know. **Only** include what the client needs to use/know about that service. Everything else is implementation, and should be abstracted away.

When writing services, make your service as general as possible. I ran into this issue often where I would accidentally make my service too specific to our current implementation without even realizing.

When naming, think about making your names as specific and clear as possible. Unclear names make for frustrating code.

Whenever you write/do anything, be **intentional**:

- What package you put your file in
- The name of that package
- The name of your class
- The way work is divided among methods

It's not just about getting the job done and working ; it's about making code that easy to understand and change when necessary

Chapter 12

Mistakes I've made

I have a ton more to add to this list, these are just a few things that come to mind:

- On Git, I deleted a file without knowing it. (Ended up spending half an hour debugging the issue before going to Nelson.) What did I learn? Always `git status` before committing!
- For my first API, I wrote interfaces that were so specific they didn't serve a purpose. That experience inspired the last chapter. What did I learn? Keep everything generic! Make sure the API does not reference your implementation.
- Sometimes, I had pretty off days. I would put in my best effort, but I'd still end up making silly mistakes or misunderstanding simple concepts. What did I learn? Not everyday will be your best day, so don't be too hard on yourself.
- This isn't a mistake, but best of luck! You're going to do great :-)