Java Prework

IMC

August 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION

I hope you are as excited as we are about you joining IMC, and we're looking forward to having you in our Software Engineering Global Traineeship (colloquially known as Dev School).

Before we start the traineeship together, we want to ensure that you have a solid understanding of certain widely used open-source technologies. We have a limited amount of time available for the traineeship, so we want to be able to use that time for working directly with instructors to learn advanced topics and IMC-specific tooling. Therefore, it is essential that you spend time familiarizing yourself with these technologies prior to the start of the program. The attached documents describe the contents that you should focus on.

While we will provide you with recommended learning resources as a starting point, please note that independent research and exploration are expected from each of you. Self-directed learning is a crucial skill for success at IMC, and this is an opportunity to practice that!

For each technology, we have compiled a set of recommended learning resources that will serve as a great starting point for your studies. We have found these to be good introductory resources for covering fundamental concepts and providing practical examples to help grasp the essentials. If they do not fit your learning style, you are encouraged to find your own resources on Google, YouTube, or elsewhere. If you are already familiar with any material, feel free to skip the specific content.

Additionally, we are providing you with a checklist of concepts for each technology. This checklist will serve as a valuable tool to ensure that you cover the most important topics within each technology. It is highly recommended that you use this checklist as a guideline to gauge your progress and track your learning.

It is very important that you are completely comfortable with these basics, as they are mostly taken for granted in the training.

If you have any questions or need further clarification, please don't hesitate to reach out to us. We are more than happy to assist you.

CHAPTER

TWO

GIT

Git is the dominant VCS in the tech community, and is used by all development teams at IMC.

Additionally, most teams use the Gerrit code review tool, which heavily relies on advanced Git features around rewriting history.

2.1 Recommended Reading Material

2.1.1 Pro Git

The following chapters contain the most useful material about using Git.

- 1. Getting Started
- 2. Git Basics
- 3. Git Branching
- 7. Git Tools
  - 7.1 Revision Selection
  - 7.2 Interactive Staging
  - 7.3 Stashing and Cleaning
  - 7.5 Searching
  - 7.6 Rewriting history
  - 7.7 Reset Demystified
  - 7.10 Debugging with Git

Chapters 4-6 cover Git Servers and Workflows that are applicable to working in open source and on Github, but not immediately relevant to work at IMC.

The other sections of Chapter 7 cover more advanced Git tools that are useful, but will not be used in Dev School.

2.2 Recommended Exercises

Learn Git Branching is a good practical exercise for learning about the most useful Git commands.

2.3 Checklist

2.3.1 Essential Concepts

You should have a solid understanding of the following concepts.

- Git Fundamental Elements
  - Commits
  - Branches
  - Tags
  - References
  - Remotes
  - Repositories
- Commit Workflow
  - Working Directory
  - Stage/Index
  - Stash
- Checking out history
- Rewriting history
- `git reset`

2.3.2 Practical Skills

You must be able to do the following:

- Git commit workflow
  - Stage change and create commits
  - See your uncommitted or staged changes
  - Throw away your un-committed changes
  - Stash and unstash changes, stash multiple changes at a time
- Exploring Git History
  - See the history of commits in a repository
  - See the commit message for any given commit?
  - See what changes went into any given commit, or what are the differences between two arbitrary commits
  - Look at the code from a previous state in the repository history
  - Figure out what commit a bug was introduced in

- Branching and tagging
  - Create, switch, merge and delete branches
  - Copy commits from one branch to another
  - Create and delete tags
- Git Remotes
  - Clone from, pull from, push to a remote
- Undoing
  - Undo a `git commit` (without losing your code).
  - Revert a commit so that the code is in the same state as if the commit wasn't done, but the commit is left in history.
  - Delete a commit so that it is gone from git history entirely.
- Rewriting history
  - Modify the commit message of the most recent commit.
  - Modify the commit message of a previous commit
  - Combine two commits into one commit
  - Split one commit into two commits
  - Reorder two commits
  - Delete a commit
  - Change a branch to point to a completely different commit
- Fixing mistakes
  - Undo a `reset`
  - Recover a "lost" commit

2.3.3 Advanced Git Concepts and Tools

You are **not** required to know anything about these.

- Submodules / Subtrees
- Sparse Checkouts
- Hooks
- Git internals
  - Objects: Blobs, Trees, etc.
  - Packfiles
- `git rerere`

2.4 Additional Material

- An Introduction To Git and Github - Introduction to Git, but in video format.
- Think Like (a) Git

2.5 Exam

The *Git Exam* can be used to evaluate your understanding of git.

This exam is not graded - you should it to identify weaknesses in your understanding.

2.5.1 Git Exam

Theory

1. What does it mean when Git commits are described as immutable? How is that reconciled with the ability to amend or delete commits?
2. What does it mean to merge branches A into branch B?
3. What does it mean to rebase branch A on top of branch B?
4. What is a "fast-forward merge"?
5. How does a tag differ from a branch?
6. What are the most common use cases for tags?
7. What is a "commit-ish"?
8. What is the difference between `HEAD` and a "head"?
9. What is the `origin`?
10. What is the different between `git fetch` and `git pull`?
11. What is a merge conflict? How do you resolve a merge conflict?
12. How would you find the commit where a bug was introduced?
13. How do you check out the code for a previous commit so that you can debug it?
14. What is "Detached HEAD mode"? How do you get into detached HEAD mode? How do you get out of detached HEAD mode?
15. What does rewriting history mean? When might you do it? When should you not rewrite history? What can go wrong when you modify history?
16. When should you delete a commit vs revert it?
17. What is the difference between `git reset 29f5e54` and `git checkout 31f5e54`?
18. What is the difference between a soft, hard, and mixed reset?
19. When would you use `git reset`?
20. Is it possible to lose a commit? Or to lose your changes? Under what circumstances?
21. What is the `reflog`? When would you use it?

Practical

Please clone https://github.com/imc-trading/devschool-git-exam.

1. Check out the branch `resume`. Change the commit message from "Add Wok Experience" to "Add Work Experience". What commands did you run?
2. Add this "Work Experience" item in "Resume.md":

```
### The Normal Brand

- Web Developer -- 2019-2021
  - Implemented OAuth login process to support Social Login.
  - Added web analytics
```

Commit it on the branch `resume` with the message "Add Normal Brand item". What command(s) did you run?

3. The commit on the branch `resume` with the message "fill edu" has some issues. Fix it like this: 1. Change the message to "Add content for Education section". 2. The commit has introduced some whitespaces at the end of two lines. Remove those. What command(s) did you run? Explain your actions.
4. The commit with the message "Add empty lines" is not useful. Remove it from the history. What command(s) did you run?
5. There is a branch called `skills`. How can you show the content of the `Resume.md` file on that branch without switching to the branch?
6. On the branch `skills` there is a commit called "Add skills". Add that commit on top of the branch `resume` without switching branches away from `resume`. That should create a conflict. Fix it by incorporating both the previous and the new changes. What commands and actions did you do?
7. In `Resume.md` there is a line `### Illinois State University`. How can you find out what is the last commit that changed/introduced that line? What is the message of that commit? What commands did you run?
8. Create a branch called `letter_of_intent` forking off from the branch `job_applications`. Create a file there called `letter.txt` with the content `I need this job, please!`. Commit this file on the `letter_of_intent` branch with the commit message `Add letter of intent`. Add another commit that adds this line to `letter.txt`: `Sincerely yours, Norman`. Commit it with the message `Sign letter`. What commands did you run?
9. Rebase the `letter_of_intent` onto the `resume` branch without including the commits on the `job_applications` branch. Basically add the commits `Add letter of intent` and `Sign letter` on top of `resume` without changing `resume` and point `letter_of_intent` to the last commit. Do not use `cherry-pick`. What command did you run?
10. Merge the branch `letter_of_intent` into the branch `resume`. What commands did you run? Explain the merge strategy taken.
11. Merge the `job_applications` branch into the branch `resume`. What commands did you run? What merge strategy was taken? What happened to the git history of the `resume` branch?

CHAPTER

THREE

UNIX BASICS

All of our deployment servers are UNIX environments and you will need to be able to interact with them using the command line in order to effectively deploy and manage your apps at IMC.

3.1 Checklist

3.1.1 Essential skills

You must be able to do the following:

- Be familiar with the UNIX command line interface and its basic functionalities.
  - Be able to navigate through a file system, create and remove files and directories. Use the `cd`, `ls`, `mkdir`, `cp`, `mv` and `rm` commands.
- View files using `cat` and `less`
- Edit files using a CLI text editor.
  - We recommend `vim` but `emacs` and `nano` are also available.
    - `vimtutor` is a useful tool for learning.
  - Be able to open, edit and write files from the CLI.

3.1.2 Useful skills

The following skills are extremely useful, but not critical for Dev School.

- Search through files and directories for text patterns with `find` and `grep`
- Redirect output to a new command with the `|` operator.
  - Combine this with `grep`!
- Redirect output to a file with > and >>
- Understand environment variables, `env`, `export`, `echo`
- Monitor a system with `ps`, `top` and `htop`
- Manage processes (`kill`)

3.1.3 Bonus

- Use a CLI editor to create a simple bash script file which prints "Hello World" (`echo`).
- Make it runnable (`chmod +x`) and run it directly from the command line.
- If your script file is called `hello.sh` what is the difference between running `bash hello.sh` and `./hello.sh`? What is a shebang?

CHAPTER

FOUR

DOCKER

Containers streamlines the process of deploying production services by enabling developers to package an application with all its dependencies into a standardized unit for software development. This approach, known as containerization, ensures that the application will run uniformly, regardless of any customized settings that machines might have that could differ from the machine used for writing and testing the code.

Docker is IMC's containerization tool of choice. Docker's lightweight, scalable, and consistent environment is particularly beneficial for creating, deploying, and running applications in a variety of environments, from local development machines to production servers, thereby increasing productivity and reducing overhead.

4.1 Recommended Tutorial

- Docker - Getting Started
  - Parts 1-9

4.2 Checklist

4.2.1 Essential Concepts

You should have a solid understanding of the following concepts. We've included some follow-up questions and related concepts under each point.

- Containers
  - What is a container?
  - Difference between virtual machines and containers
  - Advantages of containerization
  - What is Docker?
- Docker Images
  - What is a Docker image?
  - Dockerfile structure and commands
    - FROM, RUN, CMD, ENTRYPOINT, ENV, COPY, ADD, etc
  - Docker image layers
  - Image tags

- Docker Containers

4.2.2 Useful Concepts

You should have a general familiarity with the following concepts, but deep understanding is not necessary.

- Docker Volumes and Bind Mounts
  - Persistent data and Docker storage options
  - Docker volumes
  - Bind mounts
- Docker Networking
  - Host networking
  - Port binding
- Docker Best Practices
  - Dockerfile best practices
  - Docker development best practices

4.2.3 Practical Skills

You must be able to do the following:

- Build images using a Dockerfile
- Manage and delete images
- Create, start, stop, and remove containers
- Interact with running containers
- Read container logs
- Execute commands in a running container
- Inspect and monitor Docker containers
- Manage and delete containers
- Mount a local directory into a docker container
- Bind ports

4.3 Exam

The *Docker Exam* can be used to evaluate your understanding of git.

This exam is not graded - you should it to identify weaknesses in your understanding.

4.3.1 Docker Exam

1. What command(s) do you use to run MariaDB in Docker on your machine and map its port to the port `12345` on your machine? It should contain a database called `prework`.
2. What command(s) do you use on your local machine to open a prompt into the MariaDB server from step 1? You can use any Mysql client.
3. Clone `https://github.com/imc-trading/devschool-docker-exam`
4. How do you run step 1 but use `init.sql` to initialize the database?
5. What command do you use to open bash console in your running MariaDB container?
6. What commands do you use to list containers, list images, stop a container, start a container, remove a container?
7. In `server.py` you have a small Python server implementation. Please create a custom Docker image around it and a Docker compose that starts up this server and the MariaDB container from the steps above. The server should connect to the MariaDB container and it should expose its HTTP port to localhost's port 8082. How do you check that the server works? Note that the server has requirements that must be installed with `pip install -r requirements.txt`

CHAPTER

FIVE

KUBERNETES AND HELM

Kubernetes is a powerful tool for orchestrating containerized applications, offering automated management, scaling, and updating of applications across multiple hosts.

Helm is a package manager for Kubernetes that simplifies deployment of applications. It uses a packaging format called Charts, which are collections of files that describe a related set of Kubernetes resources. Helm provides a way to manage and maintain complex applications through a simple, declarative approach, making deployments repeatable and more manageable.

The next generation of IMC's deployment tools are built on top of Kubernetes and Helm, so it is valuable to have a basic understanding of how these tools work and how to work with them.

Note: Kubernetes is a complex topic with a lot of depth, most of which you don't need. Just make sure you know the K8s and Helm concepts listed in the *checklist below*. The recommended readings and tutorials are good places to start.

5.1 Resources

5.1.1 Recommended Reading

- Introduction to Kubernetes
- Config Maps

5.1.2 Recommended Tutorials

- Hello Minikube
- Helm Chart Tutorial

5.1.3 Alternatives

- Youtube - Kubernetes Crash Course

5.1.4 Reference documentation

- Kubernetes Documentation
- Helm Documentation

5.2 Checklist

5.2.1 Essential Concepts

You should have a solid understanding of the following concepts. We've included some follow-up questions and related concepts under each point.

- Kubernetes Basics
  - What is Kubernetes?
  - What are the benefits to using Kubernetes?
- Kubernetes Clusters
  - Nodes
- Kubernetes Components
  - Pods
  - Deployments
  - Services
- Helm
  - What is Helm and why use it?
  - Charts
    - Chart Structure
    - Chart Dependencies
  - Repositories
  - Releases

5.2.2 Useful Concepts

You should have a general familiarity with the following concepts, but deep understanding is not necessary.

- Services
- Storage
  - Volumes
  - Persistent Volumes
  - Persistent Volume Claims
- Configuration
  - ConfigMaps

5.2.3 Practical Skills

You must be able to do the following:

- Kubernetes Basics:
  - Deploy a containerized application on a cluster.
  - Update the containerized application with a new software version.
  - Debug the containerized application.
- Basic Kubectl Commands
  - Get info about nodes, pods, deployments, services
  - Create and delete resources
  - Get logs
  - Open an interactive shell on a running pod for debugging
- Helm Basics
  - Read simple Helm Charts
  - Create a simple Helm Chart from scratch
  - Deploy a Helm Chart

CHAPTER

SIX

PYTHON BASICS

Python is the industry standard programming language for data analytics and IMC is no exception. No matter what your primary role is, you will be well served in being comfortable reading and writing basic Python code, and working with data analysis libraries like Pandas.

6.1 Recommended Reading Material

- Python 3 Tutorial
- Pandas Getting Started Tutorials - These three chapters:
  - Data Type Handled by Pandas
  - Reading/Writing Tabular Data
  - Selecting and Filtering
  - We recommend the rest of the chapters only if your work will involve data analysis on a regular basis.

6.2 Checklist

6.2.1 Python Language Basics

You should be familiar with the syntax for programming in Python, especially following:

- Syntax and Fundamentals
  - Defining variables and functions
  - Defining classes
  - Operators, control flow
  - `range` and `enumerate`
  - Standard library data structures
  - Comprehensions
- Concepts
  - Higher order functions
  - Generators

6.2.2 Pandas

You should be able to do the following using Pandas

- Data Frames
  - Data Selection and Filtering
  - Data Grouping and Aggregation - optional
  - Data Reshaping and Pivot tables - optional
  - Combine data from multiple tables - optional
  - Query time series - optional
- Data Input and Output
  - Read data from external sources such as CSV, Excel, databases, etc.
- Visualizations
  - Generate Charts and Tables - optional

CHAPTER

SEVEN

JAVA

7.1 Resources

7.1.1 Recommended Exercises

Write code! The best way to get better at a programming language is by writing code.

One way to do this is through Code Katas. There are many lists of Code Katas online. Here is one:

- Code Katas

7.1.2 Recommended Reading Material

- Trail: Learning the Java Language
  - Please read this if you are not familiar with the Java language or if you want to refresh your Java knowledge.
- Effective Java (3rd edition) [PDF]
  - This is essential reading for all Java developers, and we **highly** recommend you read this before you join a development team. That said, we don't want you to spend the entire pre-work time reading a book, so the entire book is not required reading.
  - The following sections are going to be the most useful for Dev School.
    - Item 5: Prefer dependency injection to hardwiring resources
    - Item 10: Obey the general contract when overriding equals
    - Item 11: Always override hashCode when you override equals
    - Item 12: Always override toString
    - Item 14: Consider implementing Comparable
    - Chapter 4: Classes and Interfaces
    - Chapter 5: Generics
    - Chapter 7: Lambdas
    - Item 58: Prefer for-each loops to traditional for loops
    - Item 59: Know and use the libraries
    - Item 51: Design method signatures carefully
    - Item 62: Avoid string where other types are more appropriate

    * Item 64: Refer to objects by their interfaces
    * Item 67: Optimize judiciously
    * Item 85: Prefer alternatives to Java serialization

7.1.3 Other Useful Resources

- Java Generics FAQs
  - Good if you are not comfortable

7.2 Checklist

7.2.1 Language Syntax

You should be very familiar with the syntax for programming in Java, including but not limited to:

- Syntax and Fundamentals
  - Defining enums, classes, methods, variables
  - Operators, control flow
  - Instantiating objects
  - Calling methods
  - Generics/Type Parameters
  - Exception handling
  - Javadocs
- "Modern" (since Java 8) Syntax
  - Lambdas, method references
  - `var`
  - Records
  - Switch expressions
  - Pattern matching

7.2.2 Essential Language Concepts

You should have a solid understanding of the following concepts. We've included some follow-up questions and related concepts under each point.

- Object-Oriented Programming
  - Objects vs Primitives
  - Classes and Objects
  - Inheritance and Polymorphism
  - Interfaces
  - Access Modifiers

    - `static`
    - Inner classes
- Collections and Data Structures
    - Java Standard Library Collections
        * Collection, List, Set, Queue, Map, NavigableMap, Iterator
        * ArrayList, LinkedList, HashMap, TreeMap, HashSet, TreeSet, PriorityQueue
        * What is a `ConcurrentModificationException`?
    - Generics and Type Parameters
- Exceptions
    - Throwables, Errors, and Exceptions
    - Checked vs Unchecked Exceptions
- Functional Programming
    - Functional Interfaces
    - Lambdas and Method References
    - Java Stream API
    - Anonymous Classes
- Concurrency and Thread-Safety
    - `volatile` and `synchronized`
    - `java.util.concurrent` Standard Library
- Other Language Features and Best Practices
    - Equals/hashCode contract
    - Comparable
    - Optional
    - Records

7.2.3 Useful Language Features

You should have a general familiarity with the following language features, but deep understanding is not necessary. We will not be using them during Dev School.

- Annotations
- Java Serialization - know that you should never use it
- Finalizers
- Soft/weak references

7.2.4 Design Patterns

You should be familiar with the following common design patterns.

- Common Design Patterns
  - Singleton
  - Factory
  - Builder
  - Visitor
  - Decorator
  - Strategy
  - Command

7.3 Exam

The *Java Exam* can be used to evaluate your understanding of Java concepts.

This exam is not graded - you should it to identify weaknesses in your understanding.

7.3.1 Java Exam

Basic Java Concepts

1. Which of the following best describes a class?
   - A way to group related functions and data
   - A blueprint for creating objects, containing data members and methods that define the state and behavior of objects
   - An instance of a blueprint, containing specific values for its data members
   - A mechanism for organizing code into reusable components
2. What is the primary purpose of interfaces?
   - To define a contract or blueprint for implementing classes
   - To encapsulate data and behavior
   - To create a class hierarchy
   - To share code between classes
3. Which of the following best describes an object?
   - A collection of related data and functions in a single unit
   - A blueprint that defines the structure and behavior of other objects
   - A specific instance of a class that contains state and behavior defined by the class
   - A mechanism for organizing and reusing code
4. What are the key properties of interfaces? Select all that apply
   - Interfaces can have mutable static variables

- Interfaces can have concrete methods
- Interfaces can have abstract methods
- A class can implement multiple interfaces
- Interfaces can have instance variables
- Interfaces can have constructors

5. What are the key properties of abstract classes? Select all that apply
   - Abstract classes can have abstract methods
   - Abstract classes can have constructors
   - Abstract classes can have concrete methods
   - Abstract classes can have mutable static variables
   - A class can extend multiple abstract classes
   - Abstract classes can have instance variables
6. Which of the following best describes the purpose of a constructor?
   - To create a new object instance of a class
   - To initialize an object's state when it is created
   - To compare two objects for equality
   - To define the structure and behavior of a class
7. What is method overloading?
   - Calling a method from a parent class in a derived class
   - Redefining a method from a parent class in a derived class
   - Defining a method with the same name but different parameters in the same class
   - Defining a method with the same name and parameters in two different classes
8. What is the primary role of the Java heap?
   - To store static variables and methods
   - To allocate memory for object instances during runtime
   - To store local variables and method call information
   - To store the source code of a Java program
9. What is the purpose of the 'static' keyword?
   - To make a variable or method available without creating an object
   - To make a variable or method available only within a method
   - To make a variable or method unchangeable
   - To make a variable or method available only within a class
10. What is the difference between a static inner class instead of a non-static one?
    - There is only one global instance of a static inner class
    - A static inner class is not tightly coupled to the enclosing class.

- A static inner class can be instantiated without an instance of the enclosing class - A static inner class can access the instance variables and methods of the enclosing class

11. What is the purpose of the 'final' keyword?
    - All of the above
    - Prevents a class from being inherited
    - Prevents a variable from being reassigned
    - Indicates that a method cannot be overridden
12. Is a field with a `final` field always immutable?
    - Yes, a `final` field is always immutable
    - No, a `final` field is only immutable if the containing class is also `final`
    - No, a `final` field can be mutable if it is a reference to a mutable object
    - None of the above
13. How do you indicate that a class is immutable?
    - By declaring all its instance variables as final
    - By declaring all its instance variables as static
    - By implementing the Immutable interface
    - By declaring all its methods as final
    - None of the above
14. What is the main difference between primitive types and objects (reference types)?
    - Primitive types are passed by value, while objects are passed by reference
    - Primitive types have methods and attributes, while objects do not
    - Primitive types are user-defined, while objects are built-in types
    - Primitive types are immutable, while objects are mutable
15. Which of the following is true about Java's autoboxing and unboxing feature?
    - It automatically converts primitives to their corresponding wrapper classes and vice versa
    - It automatically converts arrays to ArrayLists and vice versa
    - It automatically casts objects to a more specific type
    - It automatically detects type mismatches
16. When passing arguments to a method, which of the following statements is true?
    - Primitive types are passed by value, and objects are passed by reference
    - Primitive types are passed by reference, and objects are passed by value
    - Both primitive types and objects are passed by value
    - Both primitive types and objects are passed by reference
17. What are varargs?
    - A way to pass multiple arguments of different types to a method
    - A shorthand syntax for creating an array

- A way to declare a variable number of arguments in a method
- A way to declare optional method arguments

18. Which of the following best describes the 'Throwable' class?
    - A class that represents an error or exception that can be caught and handled
    - A class that represents exceptions that can be caught and handled by the programmer
    - A class that represents errors that occur during the execution of a program
    - A class that represents exceptions that occur due to programming errors
19. What is the main difference between the 'Error' and 'Exception' classes?
    - 'Error' represents errors that occur during the execution of a program, while 'Exception' represents exceptions that occur due to programming errors
    - 'Error' represents errors that occur during the execution of a program, while 'Exception' represents exceptions that can be caught and handled by the programmer
    - 'Error' represents exceptions that can be caught and handled by the programmer, while 'Exception' represents errors that occur during the execution of a program
    - 'Error' represents exceptions that occur due to programming errors, while 'Exception' represents errors that occur during the execution of a program
20. Which of the following statements best describes unchecked exceptions?
    - Unchecked exceptions are exceptions that the Java compiler requires to be caught or declared in a method's 'throws' clause
    - Unchecked exceptions are exceptions that occur due to external factors, such as invalid user input or file I/O errors
    - Unchecked exceptions are exceptions that are automatically caught and handled by the Java Virtual Machine (JVM)
    - Unchecked exceptions are exceptions that are subclasses of the 'RuntimeException' class
21. In which of the following scenarios should a checked exception be used?
    - When the exception occurs due to programming errors and needs to be caught during development
    - When the exception occurs and may be propagated up the call stack without being caught or declared
    - When the exception should be automatically caught and handled by the Java Virtual Machine (JVM)
    - When the exception is caused by external factors, such as invalid user input or file I/O errors, and the programmer must be aware of and handle the exception
22. Which of the following is true about Java's garbage collection?
    - It must be manually invoked by the programmer
    - It is a part of the Java standard library
    - It is a feature exclusive to Java
    - It automatically deallocates memory for objects that are no longer needed
23. What is the equals() and hashCode() contract?
    - A requirement that if two objects are considered equal by the equals() method, their hash codes must also be equal

- A requirement that if two objects have the same hash code, they must also be considered equal by the equals() method
- A guarantee that two objects of the same class will always be considered equal by the equals() method
- A guarantee that two objects of the same class will always have different hash codes

24. What happens if the equals() and hashCode() contract is broken for a class?
    - The JVM will automatically fix the issue and repair the broken contract
    - The behavior of some standard library collections may be undefined or incorrect
    - The `equals()` method may not be used to compare instances of this class
    - An `IllegalContractException` will be thrown if the class is used in a `HashMap` or `HashSet`
25. What is the difference between Comparable and Comparator?
    - Comparable and Comparator are both interfaces used to define different ways of ordering objects
    - Comparable is an interface used to define a natural ordering of objects, while Comparator is a separate class used to define an external ordering of objects
    - Comparable is a separate class used to define an external ordering of objects, while Comparator is an interface used to define a natural ordering of objects
    - There is no difference between Comparable and Comparator
26. What is necessary in order to use a class with sorted collections?
    - The class implements Comparable
    - The class implements Comparator
    - The class implements both Comparable and Comparator
    - A Comparable for the class is provided to the collection
    - A Comparator for the class is provided to the collection
27. Which of the following statements about the toString() method are true?
    - An ideal toString() includes all of the interesting information in an object
    - Programmatic access to the information returned by toString should always be provided
    - The toString() method is used to provide a string representation of an object
    - The toString() method should always include private fields in the output
    - Overriding the toString() method can improve the debugging experience
    - The toString() method is automatically called when an object is used as a string

Intermediate Java Concepts

1. What is the main purpose of generics?
    - To reduce the amount of code needed to implement collections and algorithms
    - To improve the performance of Java programs by reducing the need for casting
    - To enables a class, interface, or method to work with different types without a need for casting
    - To enable a class, interface, or method to work only with a specific type
    - To allow a variable to be declared with no specific type

- To make Java more compatible with other programming languages

2. If you have a `List<Number> numbers`, which of the following are allowed?
    - `Object o = numbers.get(0);`
    - `Number n = numbers.get(0);`
    - `Integer i = numbers.get(0);`
    - `int x = numbers.get(0);`
    - `numbers.add(x); // x is type int`
    - `numbers.add(i); // i is type Integer`
    - `numbers.add(n); // n is type Number`
    - `numbers.add(o); // o is type Object`
    - `numbers.add(null);`
3. If you have a `List<? extends Number> numbers`, which of the following are allowed?
    - `Object o = numbers.get(0);`
    - `Number n = numbers.get(0);`
    - `Integer i = numbers.get(0);`
    - `int x = numbers.get(0);`
    - `numbers.add(x); // x is type int`
    - `numbers.add(i); // i is type Integer`
    - `numbers.add(n); // n is type Number`
    - `numbers.add(o); // o is type Object`
    - `numbers.add(null);`
4. If you have a `List<? super Number> numbers`, which of the following are allowed?
    - `Object o = numbers.get(0);`
    - `Number n = numbers.get(0);`
    - `Integer i = numbers.get(0);`
    - `int x = numbers.get(0);`
    - `numbers.add(x); // x is type int`
    - `numbers.add(i); // i is type Integer`
    - `numbers.add(n); // n is type Number`
    - `numbers.add(o); // o is type Object`
    - `numbers.add(null);`
5. Which of the following can cause a `ConcurrentModificationException`?
    - Multiple threads are assigning to a variable concurrently.
    - Multiple threads are calling modifying a non-thread-safe Collection concurrently.
    - One thread is modifying a `HashMap` while another thread is reading from it.
    - One thread is modifying a `HashMap` while another thread is iterating over it.

- One thread is modifying a `ConcurrentHashMap` while another thread is iterating over it.
- A single thread is modifying a `HashMap` while it is iterating over it.
- A single thread is modifying a `ConcurrentHashMap` while it is iterating over it.

6. Which of the following are true about writing thread-safe code.
   - Writing thread-safe code is hard
   - Synchronizing all methods of a class ensures thread-safety
   - It is safe to use a `HashMap` from multiple threads as long as the threads are only reading
   - It is safe to use a `HashMap` from multiple threads as long as no more than one thread is writing
   - A `volatile` field can be accessed from multiple threads without thread-safety concerns - A write to a `volatile` field is always visible to other threads
   - Using a thread-safe collection such as `CopyOnWriteArrayList` ensures thread-safety for all operations on that collection
7. Which of the following is true of using the Java Stream API?
   - It provides a functional programming approach to work with collections
   - It allows for parallelizable operations
   - It supports lazy evaluation of intermediate operations
   - It can reduce code complexity and improve readibility in some cases
   - It is always an improvement over imperative code
8. What is an anonymous class?
   - A class that is used to define an unnamed tuple object
   - A class that is defined inside another class and can access its enclosing class's variables and methods
   - A class that is defined with no name and can be used to implement an interface or extend a class inline
   - A class that is used to define a custom data type in an anonymous context
9. What is a lambda expression?
   - An anonymous function that can be used as a method parameter or a variable assignment
   - A shorthand way of defining a class
   - A way to create an anonymous object
   - A way to declare a constant variable
10. What is a method reference?
    - A type of anonymous class that can be used to define functions inline
    - A way to invoke a method on an object using a shorthand syntax
    - A reference to a method that can be stored or passed as a value
    - A way to create new instances of a class using a factory method
11. What is a functional interface?
    - An interface that defines a single abstract method and can be used as a lambda expression or method reference

- An interface that defines multiple abstract methods and can be used as a lambda expression or method reference
- An interface that can only be implemented by classes in the java.util.function package
- An interface that is used to define a natural ordering of objects

12. What is a Java annotation?
    - A comment in the code
    - A way to attach metadata to code elements such as classes, methods, and fields
    - A way to define a class, interface, or method
    - A shorthand syntax for creating an object
13. Which of the following statements are true about records?
    - Records are a new feature introduced in Java 16
    - Records can be used to define simple data-holding classes with a concise syntax
    - Records are always immutable
    - Record fields are `final` by default, but can be made non-`final`
    - Record fields are always private
    - Records automatically generate equals(), hashCode(), and toString() methods based on their state
    - Records can be subclassed like regular classes
    - Records cannot have non-default constructors
    - Records support all of the same features as regular classes, but with a different syntax
14. Which of the following statements are true about serialization?
    - Java provides built-in serialization by implementing the `Serializable` interface
    - It is recommended to use Java Serialization instead of a third-party serialization library
    - Supporting serialization can increase the maintenance cost of a software system due to compatibility concerns
    - JSON is the best serialization format
    - `Jackson`, `Avro`, and `Protocol Buffers` are commonly used serialization libraries

CHAPTER

EIGHT

MAVEN

Maven is a widely adopted build tool and dependency management system that simplifies and automates the building, testing, and packaging of Java projects. With its convention-over-configuration approach, extensive plugin ecosystem, and robust dependency resolution, Maven enhances project development by promoting standardized project structures, ensuring consistent builds, and facilitating seamless collaboration among developers.

Historically, Java projects at IMC were built were Maven. Some newer projects are being migrated to Bazel, but a vast majority of projects are still on Maven with no near-term plans to migrate.

8.1 Recommended Reading Material

- Apache: Maven in 5 Minutes
- Apache: Maven Getting Started Guide
- Baeldung: Apache Maven Tutorial
- Baedung: Various Maven Tutorials

8.2 Checklist

8.2.1 Concepts

You should understand the following concepts:

- Build Tool Basics
  - The purpose and benefits of build automation tools
- Maven's Design Philosophy
  - "Convention over Configuration"
- Maven Project Structure
  - Standard directory layout
  - Projects and Modules
  - Project Object Model (POM)
  - Parent POMs
- Maven Build Lifecycle
  - Build Phases

        * compile vs package
        * install vs deploy
        * test vs verify
      - Plugin Goals
      - Default Lifecycle
      - Default Plugin Bindings
- Dependencies
    - Project Dependencies
    - Transitive Dependencies
    - Dependency Scopes
    - Dependency Management
    - Bill of Materials (BOM)
- Version Management
    - Maven version conventions
    - SNAPSHOT versions
    - Version Ranges
- Multi-module Projects
    - What is the rationale for multi-module projects?
- Build Profiles

8.2.2 Practical Skills

Know how to do the following:

- Navigate a Maven project
    - Find the pom file and source, test, and resources directories.
    - Read a POM file
- Create a simple Maven project
    - Define a POM with
        * Project metadata
        * Dependencies
        * Child modules
    - Specify dependencies
        * Use build properties to specify versions
        * Use Dependency Management
        * Use BOMs
- Maven Build Commands
    - Build a project with and without running tests

  - Build a project on multiple cores
  - Run a specific test
  - Build one module only
- Maven Plugins
  - Know the functionality of the core Maven plugins
  - Know how to search for plugin configuration
  - Run a plugin goal instead of a lifecycle phase
  - Configure plugins in a POM
- Build Profiles
  - Create and use profiles
  - Activate and deactivate profiles
    - Automatically and manually
- Query a project
  - Find the versions of all dependencies of a project
  - Find which dependency pulls in a specific transitive dependency

CHAPTER

NINE

INTELLIJ

An IDE is essential to improving your productivity as a Software Engineer. IntelliJ is the preferred Java IDE for the vast majority of Java teams at IMC, and the IDE with the most internal support.

9.1 Recommended Reading

- IntelliJ - Getting Started
- IntelliJ - Feature Trainer

9.2 Recommended Viewing

- Youtube - Be More Productive With IntelliJ IDEA
- Youtube - IntelliJ IDEA Debugger Essentials

9.3 Practical skills

Be able to use the following features:

- Reading Code
  - Code Navigation
  - Find Usages
- Writing Code
  - Inspections
  - Live Templates
  - Nullability Analysis and Annotations
  - Local History
  - Code Style and Formatting
- Refactoring
  - Renaming
  - Change Signature

    - Extract/Introduce refactorings
    - Inline
    - Safe Delete
    - Migrate
- Running
    - Running a Java application
    - Running tests
- Debugging
- Maven Integration

CHAPTER

TEN

JAVA TESTING

10.1 Tools

These are the preferred testing libraries for the majority of Java development teams at IMC, and what we will be using during Development School. There are alternative libraries for most of these, but they are sufficiently similar that there's no need to learn the other ones.

- Unit Testing Framework: JUnit 5
- Mocking: Mockito
- Assertions: AssertJ
- Maven Runner: Surefire/Failsafe

10.2 Recommended Reading Material

The above documentation websites have relatively high quality examples and tutorials, so we recommend starting with those.

10.3 Checklist

10.3.1 Concepts

Understand these concepts

- Unit tests vs Integration or End-to-end testing
- Mocks vs Fakes vs Spys

10.3.2 Practical Skills

Know how to do the following:

- JUnit
  - Test Fixtures
    - Run simple unit tests
    - Ignore tests
    - Setup/tear down test state
      - `@Before`, `@After`
      - `@BeforeClass`, `@AfterClass`
  - IDE integration
    - Run tests in IntelliJ
- Mockito
  - Mocking
    - Create mocks
    - Stub method calls
    - Verify interactions
    - Argument captors
    - Use Spys
    - Mock Fluent APIs
    - Use `@InjectMocks` and `MockitoJUnitRunner`
- AssertJ
  - AssertJ Core functionality
- Surefire/Failsafe
  - Run tests with Maven
  - Run a specific tests
  - Skip tests
  - Debug tests run with Maven

10.3.3 Advanced Features

We won't be using these in dev school, but they are useful and used in some places at IMC.

- Advanced JUnit Features
  - Rules
  - Parameterized Tests
  - Theories