TASK

Thinking Like a Software Engineer II — Diving into a Large Codebase and Writing Maintainable Code

Visit our website

Introduction

WELCOME TO THE DIVING INTO LARGE CODEBASES AND WRITING MAINTAINABLE CODE TASK!

More often than not, a Software Engineer is required to work with an already established codebase. For someone who has not worked with large, already existing codebases, facing thousands of lines of code can be a daunting task. This task will explain how to tackle large codebases as well as how to write maintainable code.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

WHAT IS A CODEBASE?

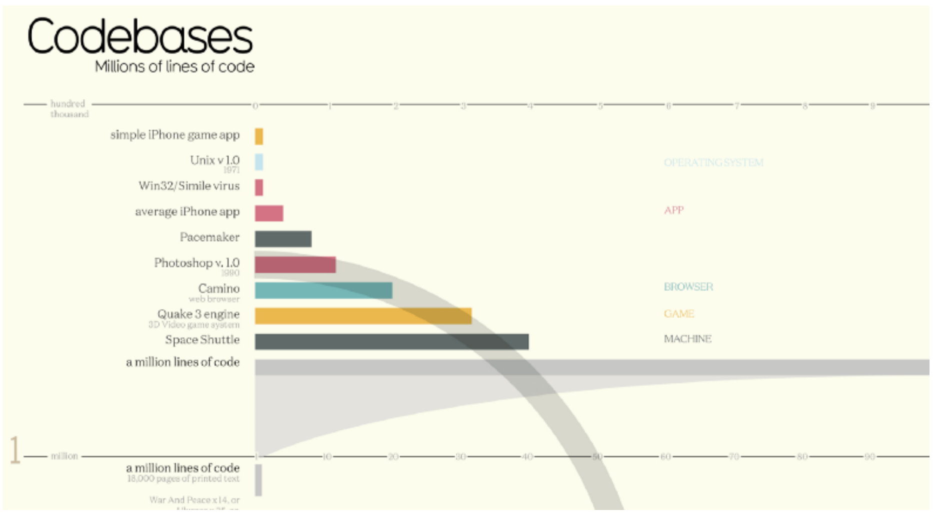
So, what exactly is a codebase? A codebase refers to a whole collection of source code that is used to build a particular software system, application or software component. Only human-written source code files are included in a codebase. Therefore, source code files that are generated by tools and binary files are not included. Codebases also generally include configuration and property files as they are necessary for the build.

A codebase is normally stored in a source code repository that belongs to a version control system. A source code repository is a place where a large amount of source code is kept, either publicly or privately. They are typically used for backups and versioning. Code repositories are also used for projects with multiple developers to handle the various source code versions and to aid in resolving conflicts when different developers submit overlapping modifications.

A note from the

HyperionDev Team

Have you ever wondered how many lines of code there are in the Google codebase? Believe it or not, there are actually 2 billion lines of code in the Google codebase and all of them fit into a single code repository! Below is an infographic (found in full **here**) showing the biggest codebases in history, as measured by lines of code.



2 billion
1
10
20
30
40
50
60
70
80
90
Google
all internet services
*Human Genome = 3,300 billion "lines" of code

WORKING WITH LARGE CODEBASES

Diving into a large unknown codebase can be intimidating, especially if you have never worked with large codebases before. As a Software Engineer starting in a new company, you will often need to jump straight into the codebase and become productive as quickly as possible. Fortunately, there are several tips that can help you tackle a large codebase and feel less overwhelmed.

1. **Try to understand the purpose of the application:**
   Before even looking at the source code, it is important to understand the purpose of the application. By doing this, you have a better context for what the different objects in the code represent. Understanding the business domain of the application provides context for why features exist and why the code is written in certain ways.

2. **Use the application:**
   It is always a good idea to use the actual production application as an end-user. By doing so, you can get a high-level overview of the application.

Have someone who is well versed with the application give you a tour. Someone who knows the features well will be able to provide you with much insight.

3. **Do a cursory review of the code:**
   Don't worry about the details at this point. Focus on the broad picture. Pay attention to how the code is organised.

4. **Talk to the other developers working on the app:**
   It is always beneficial to talk to other people who have worked on the app. They will be able to summarise what the application does and how major features work. They can also warn you about parts of the code that are poorly written and that you should avoid. Most importantly, remember to ask questions if you get stuck!

5. **Look at tests:**
   Tests give you a better understanding of what edge cases the application might be handling, or what business logic is important. Integration tests can also tell you what the most important user workflows are.

6. **Review the application environment:**
   At this point, when you are still trying to get comfortable with the code, you do not need to know every detail about how to push to production. However, you do need to understand how to get it running on your development machine.

7. **Start coding:**
   Don't waste time by spending days reading code trying to understand every single component of the application right away. Instead, start diving in with some small bugs or features. This helps you focus on a small section of the application without worrying about how everything else fits together.

8. **Add documentation:**
   It is essential that you add documentation to make it easier for the next person that might work on the application. You can also start a glossary to keep track of what things mean, either just for yourself to use or to share with other people.

HOW TO WRITE MAINTAINABLE CODE

It’s all good and well to write code that works. But code, by nature, is constantly changing. When the requirements of what the code needs to do changes, your code needs to be able to adapt. Furthermore, your code will be seen and worked on by many people — from development to final testing and beyond. This is why it is important for your code to be easily maintainable — having easily maintainable code means that your code is readable, neat, easy to understand, easy to test and easy to adapt. In a later task, you will learn how to do this more practically using software documentation, but for now, it is important for you to follow these guidelines:

1. **Make your code readable.** It’s not always necessary to perform an action in one line of code, even if it does look elegant. It is usually better to take up a little more space in order to make it clear what you are doing. This includes keeping modules in separate files. Remember to use descriptive variables to help readers understand your code.
2. **Code for the future.** Hard-coding (coding in actual values into your code) can make it difficult to scale your code. It is also important to make sure your modules don’t overlap too much and your code is not overly complicated — trying to detangle heavily nested loops in order to scale code can take up a lot of unnecessary time.
3. **Make your code testable.** One more advantage of keeping your code in separate modules is it makes it easy to test because each module or unit can be tested separately. You will learn more about unit testing in later tasks.

Compulsory Task

Follow these steps:

- Look the poorly written, difficult to read program below:

```
import java.util.Scanner;

public class WhatDoesThisAppDo {
public static void main(String[] args) {
final int NUMBER = 5;
int int2 = 0; int int1 = 0; long startTime = System.currentTimeMillis();
String string1 = " "; Scanner string2 = new Scanner(System.in);

while (int1 < NUMBER) {
int number1 = (int)(Math.random() * 10);
int number2 = (int)(Math.random() * 10);
if (number1 < number2) {
int temp = number1; number1 = number2; number2 = temp;
}
System.out.print(
"What is " + number1 + " - " + number2 + "? ");
int answer = string2.nextInt();
if (number1 - number2 == answer) { System.out.println("You are correct!");
int2++; // Increase the correct answer count
}
else
System.out.println("Your answer is wrong.\n" + number1
+ " - " + number2 + " should be " + (number1 - number2));
int1++;
string1 += "\n" + number1 + "-" + number2 + "=" + answer +
((number1 - number2 == answer) ? " correct" : " wrong");
}
long endTime = System.currentTimeMillis();
long testTime = endTime - startTime;
System.out.println("Correct count is " + int2 +
"\nTest time is " + testTime / 1000 + " seconds\n" + string1);
}
}
```

- Firstly, without running the program, determine what the above code does. Write your answer in a text file called **answers.txt**.

- What is the output of this programme? (Provide your own input data if necessary) Write your answer in the **answers.txt** file.

- Rewrite this programme so that it is easier to read. Make sure your code has:
    - descriptive variable names
    - descriptive class names
    - at least 3 comments
    - consistent indentation
    - code Grouping

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.