1. (1pt) **Name:** ______________________________

INSTRUCTIONS: Carefully read each question, and write the answer in the space provided. If answers to free response questions are written obscurely, zero credit will be awarded. The correct answer to a free response question with a short answer (i.e., one word or phrase) will never contain any significant words used in the question itself (i.e., "crossword rules"). You are permitted one 8.5x11 inch sheet of paper (double-sided) containing **hand-written** notes; all other aids (other than your brain) are forbidden. Questions may be brought to the instructor.

For **TRUE** or **FALSE** and multiple choice questions, circle your answer.

On free response questions only, you will receive **20%** credit for any question which you leave blank (i.e., do not attempt to answer). Do not waste your time or mine by making up an answer if you do not know. (Note though that most questions offer partial credit, so if you know part of the answer, it is almost always better to write something rather than nothing.)

To get credit for this question, you must:

- Print your name (e.g., "Martin Kellogg") in the space provided on this page.
- Print your UCID (e.g., "mjk76") in the space at the top of **each** page of the exam.

Contents (blanks for graders only):

Writing your UCID on every page:	1	/ 1
I. Reading Quiz Redux:	3	/ 3
II. Very Short Answer:	23	/ 23
III. Matching:	24	/ 24
IV. Short Answer:	60	/ 60
V. "Your Choice" Reading Quiz	2	/ 2
VI. DBQs:	37	/ 37
VII. Extra Credit:	8	/ 0
Total:	158	/ 150

I. Reading Quiz Redux (3pts)

2. (1pt) **Debugging, Part I**: This reading (by Zeil) opens with an email from a student. In one short phrase, what was the student's mistake in their email? **Guessing without checking**.

3. (1pt) **Architecture, Part I**: The authors of both of these readings (Kästner and Ross) argue that one of the main benefits of good architecture is:
   - **A** code that doesn't break (as often)
   - **B** it makes it easier to hire programmers, since they don't have to think about design
   - **C** it leads to more "elegant" and aesthetically-pleasing code
   - **D** it enables better communication among team members

4. (1pt) **DevOps, Part II**: One of these readings (by Luu) lists a set of problems that commonly are the root cause of outages in post-mortems that the author has read. Which of the following is **NOT** one of the root causes discussed?
   - **A** error handling
   - **B** configuration changes
   - **C** buffer overflows
   - **D** hardware failures

II. Multiple Choice and Very Short Answer (23pts). In the following section, either circle your answer (possible answers appear in **bold**) or write a very short (one word or one phrase) answer in the space provided.

5. (2pt) Order the following items from most to least abstract (write a four character string in the following blank; the first character should be the letter corresponding to the most abstract answer, the second to the next most abstract, etc.). **BACD. BCAD gets 1 point.**
   - **A** architecture
   - **B** requirements
   - **C** design
   - **D** source code

6. (1pt) **TRUE** or **FALSE**: you are about to implement a new feature that requires you to modify code that you know has a lot of technical debt. Now is an appropriate time for a large refactoring.

7. (2pt) Which of the following are best practices related to continuous integration (CI)? (Circle all that apply.)
   - **A** the CI build should not be permitted to fail for long periods of time
   - **B** all changes to the project should be gated by CI
   - **C** developers should run the full CI build on their machine before committing
   - **D** developers don't need to run tests locally at all, since all tests will run in CI

8. (2pt) Which of the following languages is the "lingua franca" of multi-language projects: that is, which language do the native interfaces of most other languages target?

   A Java bytecode

   B x86-64 assembly

   C Python

   [✓ **D**] C

9. (2pt) A **client-server** architecture partitions tasks or workloads between the providers of a resource and requesters of that resource.

10. (2pt) The **Liskov substitution principle** states that any subclass object should be safe to use in place of a superclass object at run time.

11. (1pt) **TRUE** or [✓ **FALSE**]: in industry, most software systems must have a formal specification.

12. (2pt) Suppose that as an engineer, you recognize that the algorithm f that you've been tasked to implement is idempotent. How can you take advantage of this property?

    [✓ **A**] write tests checking that $f(x) = f(f(x))$

    B write tests checking that $f^{-1}(f(x)) = x$ (f^{-1} means "the inverse of f")

    C there are high-quality static analyses that enforce idempotency, so you should use one

    D idempotency isn't useful for software engineering

13. For each of the following terms, circle "good" if this is a good property for a test case or test suite to have, or "bad" if this is a bad property for a test case or test suite to have.

    (a) (1pt) **good** / [✓ **bad**] flakiness

    (b) (1pt) [✓ **good**] / **bad** hermeticity

    (c) (1pt) [✓ **good**] / **bad** high mutation score

    (d) (1pt) [✓ **good**] / **bad** high statement coverage

14. (1pt) The **dirty bit** / [✓ **verifying trace**] rebuilding strategy used by modern build systems avoids a problem arising in older build systems that rely on timestamps (the timestamps could have been computed in different time zones).

15. (2pt) Rice's theorem implies that:

    A you cannot achieve 100% statement coverage, no matter how many tests you write

    B tests can only show the presence of bugs, not their absence

    [✓ **C**] static analyses cannot be both perfectly sound and perfectly precise

    D static analyses will always miss some bugs

16. (2pt) Technical debt does not have a specific creditor (unlike financial debt: for example, if you borrow \$100 from the bank, you have to pay it back *to the bank*). Conceptually, when you take on technical debt, you are instead borrowing from **the future maintainers of the system**.

A. toil	**B.** object-oriented programming	**C.** centralized VCS	**D.** overhead
E. delta debugging	**F.** visitor pattern	**G.** bitrot	**H.** distributed VCS
I. watchpoint	**J.** statement coverage	**K.** explicit oracle	**L.** coupling
M. behavioral interview	**N.** conditional breakpoint	**O.** branch coverage	**P.** playbook
Q. technical interview	**R.** open-source software	**S.** fork	**T.** implicit oracle
U. factory pattern	**V.** free software	**W.** on-call rotation	**X.** unit testing
Y. cohesion	**Z.** functional programming		

III. Matching (24pts). This section contains a collection of terms discussed in class in an "Answer Bank" (choices **A.** through **Z.**). Each question in this section describes a situation associated with an answer in the Answer Bank. Write the letter of the term in the Answer Bank that best describes each situation. Each answer in the Answer Bank will be used at most once.

17. (2pt) **O: branch coverage** As James evaluates his project's test suite, he wants to distinguish between the `then` and `else` parts of `if` statements.

18. (2pt) **H: distributed VCS** When Abe wants to integrate changes from his co-worker, he first needs to commit his own changes.

19. (2pt) **U: factory pattern** Rather than permit clients to call constructors directly, Franklin provides another class that is responsible for object creation.

20. (2pt) **Z. functional** Teddy's compiler can take advantage of referential transparency.

21. (2pt) **M. behavioral interview** Donald is asked about a time that he had a disagreement with a co-worker.

22. (2pt) **P. playbook or runbook** When Joe is paged at 3AM, he immediately opens a list of steps about what to do that his team has prepared in advance.

23. (2pt) **S. fork or hard fork** John is part of an open-source community. He disagrees with a philosophical decision taken by the primary maintainers, so he creates a copy of the project and invites other like-minded community members to work on his copy instead.

24. (2pt) **T. implicit oracle** George is running a test-generation tool. It reports that there is a bug in his code, because it was able to cause the code to crash.

25. (2pt) **A. toil** As Ike's service becomes more popular, he finds he needs to do more operations work for each additional user.

26. (2pt) **L. coupling** When Barack makes one change, he finds that he needs to make a change in a seemingly-unrelated class at the same time.

27. (2pt) **I. watchpoint** Thomas knows that eventually there is an incorrect value in a particular variable, but he isn't sure what part of the code is responsible for updating that variable to the wrong value.

28. (2pt) **V. free software** Richard has strong philosophical beliefs about sharing, and he uses a copyleft license to enforce that others who use his code must respect those beliefs.

IV. Short answer (60pts). Answer the questions in this section in at most two sentences.

29. (3pt) Support or refute the following claim: developing a project in C is safer than developing the same project in Python. (By *safer*, we mean "less likely to result in code that suffers a fault at run time.") **Likely refute. C is hard to reason about and has a relatively weak type system. Even though Python's type system is enforced at run time, it still prevents more serious problems (e.g., buffer overflows or segfaults) than C does. Exceptions is also a good thing to mention here. This was a common error on the midterm (claiming that C is safer than Python). 1 point for saying that C has static typing (it's still less safe, but that is at least true).**

30. Delta debugging technically requires three mathematical properties of its "Interesting" function: "interesting" must be *monotonic*, *unambiguous*, and *consistent*. Of the three, real-world use of delta debugging tolerates violations of two; violations of the third cause delta debugging to fail to return a correct answer. For each property below, either write "FAIL" or describe a real-world situation where delta debugging is still useful despite the property being violated and justify your answer with 1-2 sentences explaining how the violation of the property changes the delta debugging output in the situation you propose. If you write "FAIL" for more than one of the sub-questions, you will recieve zero credit for this whole question.

    (a) (4pt) Monotonic **Violations of monotonicity cause DD to find *an* interesting subset, but not *the minimal* one. Many real-world use cases violate monotonicity. 1 point for restating the definition of monotonicity.**

    (b) (4pt) Unambiguous **FAIL**

    (c) (4pt) Consistent **Violations of conistency cause DD to be slower: quadratic instead of linear time. Best example of inconsistency is in the slides: when looking for a bug-inducing commit set, some combinations of commits may not build or run. 1 point for only restating the definition of consistent.**

31. (3pt) Support or refute the following claim: "Promotion-driven development" is a process anti-pattern. **Likely support: Promo-driven dev leaves systems saddled with technical debt, but helps individual engineers. Alternative refute: Our economic system encourages individual engineers to look out for themselves, so promo-driven dev is a reasonable plan: taking on tech debt that someone else (or the company, broadly) will need to service so that you personally can benefit is what the system incentivizes you to do, so you should do it to get ahead. Good answers must mention technical debt, regadless.**

32. (4pt) Suppose that you are an engineer at ganges.com, a new e-commerce startup whose business model is definitely not copied from another company named after a large river. You are responsible for a system that permits customers to view all of their previous e-commerce orders, whose performance you know scales with the number of orders that a given customer has ever placed with your company. Your manager has asked you to nominate a performance metric for this system, and you are considering "median latency" and "99.9% latency". Which of these metrics should you pick? Justify your choice of metric in 1-2 sentences. **Likely "99.9% latency". Customers whose latency is in the 99.9% will definitely be your "most active" customers, so it makes sense to optimize for them rather than for the "median" customer: they probably make vastly more orders than average!**

33. (3pt) You are an engineer working on a webservice similar to Covey.town. To run your service, a set of API keys is required (e.g., a key to Twilio). Your coworker wants to setup a CI server for your service, and suggests that you commit your API keys to your (public) git repo (so that they are easy to access on the CI server). Support or refute that your coworker's plan is a good one. **Almost certainly refute. Secrets like keys should NEVER be committed to public repositories, because they permit anyone else to arbitrarily use the API that they unlock, authenticated as you. This question was motivated by the fact that I observed at least one team committing their API keys to their repository for the course project.**

34. (3pt) Support or refute the following claim: as an engineer working on a service, deploying your service using a container to commodity cloud machines (i.e., cloud machines owned and operated by a cloud hosting provider like AWS, Microsoft, Google, etc.) is a good way to reduce toil. **Likely support. Container-based virtualization makes debugging easy, and cloud providers usually have better ops than you do.**

35. Each of the following pairs of italicized terms were discussed separately in lecture, but they are associated—that is, the elements of the pair are related. For each pair, explain the association in 1-2 sentences.

    (a) (3pt) *functional teams* and *microservice architecture*
    Big tech usually organizes its engineers into functional teams that work on a single microservice. Microservice architecture promotes siloing teams based on function in this manner.

    (b) (3pt) *hermetic builds* and *continuous integration*
    CI builds are run on a fresh machine, which means that the build itself must be hermetic: otherwise, the CI build will fail. Having CI that's always passing means that your builds need to be (mostly) hermetic.

    (c) (3pt) *automated formatters* and *toil*
    Manually conforming to a formatting guide is a form of toil.

36. Consider the following pairs of italicized tools, techniques, or processes. For each pair, give a class of defects or a situation for which the first element performs better than the second (i.e., is more likely to succeed and reduce software engineering effort and/or improve software engineering outcomes) and explain both why the better choice performs better and why the worse choice performs worse.

    (a) (3pt) *abstract factory pattern* better than *named constructors* **In class, we gave the example of a game with enemies that change based on difficulty. The same example applies here.**

    (b) (3pt) *taint analysis* better than *modern code review* **The best answer involves a situation where a source is introduced in one PR and a sink is introduced in another PR. A taint analysis will catch it, but MCR cannot (you only look at changed code). Any answer that mentions that MCR is incremental but TA is not receives 2 of 3 points.**

    (c) (3pt) *taking on technical debt* better than *writing unit tests* **Easy answer: your deadline is tomorrow.**

37. Show that the *test suite prioritization problem* is NP-complete.

    (a) (4pt) Formally define the *test suite prioritization problem.* **Given a set of tests $T = \{t_1, t_2, \ldots, t_n\}$ each with an associated probablility of finding a bug p_i and time cost c_i and a time budget b, find the subset of T that maximizes the probability of finding a bug while staying within the budget.**

    (b) (5pt) Show that the problem, as defined in part (a) above, is NP-complete. Hint: you should do a reduction to a problem that you already know is NP-complete. **The easiest solution is a straightforward reduction to the knapsack problem, which is known to be NP-complete. The knapsack problem is to fill a bag of capacity c with items from a set I while maximimizing the overall value of the selected items. Each item has a weight w (which counts against the bag's capacity) and a value v. Suppose we have an oracle for the knapsack problem. Then, we could use it to solve the test case prioritization problem: the testing budget is mapped to the bag's capacity and the tests are mapped to the items: each test's bug-finding probability is its value, while the test's time cost is mapped to the weight.**

38. You are being interviewed at a large technology company. The interviewer asks you to draw an architecture diagram for a new system on the whiteboard. All the interviewer tells you about the system whose architecture you should draw is that it "Design the high-level software architecture for a Mario-like platformer game. The game must have multiple levels, a player character, and enemies."

    (a) (1pt) What architectural style should you use? **Model-view-controller**

    (b) (4pt) Draw the architecture diagram. **Any simple MVC architecture is fine. The model should be clearly labeled as the underlying game simulation. Within the model, I expect to see at least the game world (with multiple levels), a physics engine or similar, and some kind of enemy and player models. The controller should accept player input.**

The view should be the UI. Generic MVC architecture diagram with no mention of question-specific things gets 2 points..

V. Document-based Questions (37pts). All questions in this section refer to a documents **A-G**. These documents appear at the end of the exam (I recommend that you tear them out and refer to them as you answer the questions).

Questions on this page refer to document **A** (1 page).

39. (6pt) Complete the author's argument. Why wouldn't engineers believe that "hard drive failure" is the cause of an S3 outage? One paragraph should be sufficient. **Direct quote from the author: «It's not that they would doubt that a hard drive failed; we know that hard drives fail all of the time. In fact, it's precisely because hard drives are prone to failure, and S3 stays up, that they wouldn't accept this as an explanation. S3 has been architected to function correctly even in the face of individual hard drives failing. While a failed hard drive could certainly be a contributor to an outage, it can't be the whole story. Otherwise, S3 would constantly be going down. To say 'S3 went down because a hard drive failed' is to admit 'I don't know how S3 normally works when it experiences hard drive failures'.» Any argument that relies on the assumption that hard drives will fail and that systems like S3 should be designed to mitigate those failures will receive full credit. A common error (-1) was to fail to make the connection to architecture or design, and just assert that this is the way that these systems are. Another common error was appealing to Amazon's reputation. An engineer wouldn't believe a data storage service provided by anyone serious would fail this way.**

40. (6pt) Support or refute the following claim in at most one paragraph: a similar argument means that no engineer will believe that an outage is caused by "human error". **Likely support: humans also fail at a predictable rate. The best responses will cite Dan Luu's article that we read that argued that human-in-the-loop in ops is an anti-pattern.**

Questions on this page refer to documents **B-G**.

41. (25pt) Your team consists of two other software engineers, you (also a software engineer), and your manager, who has a non-technical background. Your two engineer coworkers are having a dispute about how to use Java's `Optional` type. In particular, their dispute is about whether your company style guide should *encourage* the use of the `.equals()` method on `Optional`s or whether calls to that method should be *forbidden*. Documents **B** and **C** are your two coworkers' arguments (written as emails). Documents **D**, **E**, **F**, and **G** are the evidence that they cite. Your manager does not have a technical background, and asks you to make a decision: neither of your other coworkers are going to budge on their position. Write a polite, well-reasoned email to your team with your decision. Note that you must *explicitly* take a side: your manager is insisting that your vote is the deciding one for style guide. In your argument, you must reference at least two of documents **D-G** (though I recommend referencing all four) and at least two things we discussed in class that are not mentioned in documents **D-G**. Answer in at most five paragraphs; you may continue onto the next (blank) page. **Either side can be ok, as long as it is well-reasoned. I am primarily looking for how you use the documents to support your argument, and especially that the other things we discussed in class that you reference are both relevant and correct. Common mistakes/deductions/bonuses:**

- **just restating the argument made by one of the coworkers without adding much (-15), while still clearly taking a side**
- **failing to interrogate *why* ErrorProne might have the rule it does (-5)**
- **failing to mention anything else we discussed in class (-5), but otherwise well-reasoned**
- **missing the difference between .equals and ==, -5**
- **saying that it's important to resolve this so that we don't have to argue about it again down the line (+1 on its own, +2 if analogy to autoformatting)**
- **saying that a doc says something it does not (deduction varies by how wrong you are)**
- **nonsensical arguments (deduction varies)**

(continue your answer to question 41 on this page)

VI. "Your Choice" Reading Quiz (2 points).

Each question in this section is a reading quiz question for one of the "Your Choice" readings. Select **one** of the questions and fill in the question below (question number 42) with the letter of the question you are answering (and its answer, of course!). **DO NOT CIRCLE ANSWERS TO SUB-QUESTIONS IN THIS SECTION.** You may answer additional questions in the extra credit section (Section VII), if you have done more than one "Your Choice" reading.

42. (2pt) Write the letter of the question you are choosing to answer: **Any number.**
Write the answer to that question: **The answer.**

(a) Ko and Myers' *Designing the Whyline: A Debugging Interface for Asking Questions about Program Behavior*: The system described in the article operates inside of tool aimed primarily at students. Which tool?

A Scratch
B Roblox
C MicroWorlds
[✓ **D**] Alice

(b) Cleve and Zeller's *Locating Causes of Program Failures*: [✓ **TRUE**] or **FALSE**: the paper includes a case study of the proposed technique on the `gcc` compiler.

(c) Garlan's *Software Architecture*: Which of the following does Garlan identify as an emerging area within software architecture (at the time of writing)?

[✓ **A**] product lines
B design patterns
C UML diagrams

(d) Kellogg et al.'s *Verifying Object Construction*: This paper is concerned with errors arising from misuse of a particular design pattern. Which one?

[✓ **A**] builder pattern
B visitor pattern
C factory pattern
D named constructor pattern

(e) Kim et al.'s *A Field Study of Refactoring Challenges and Benefits*: [✓ **TRUE**] or **FALSE**: the study found a significant difference between developers' opinions about what "refactoring" entails and the academic definition in use at the time the article was written.

(f) Malkawi's *The art of software systems development: Reliability, Availability, Maintainability, Performance (RAMP)*: the paper describes a number of case studies. Which of the following is **NOT** a case study described in the paper?

A modeling of water flow in floods

B managing data produced by telecommunications switches

C just-in-time logistics

D intermodulation complexity for radio frequency devices

(g) Dean and Barroso's *The Tail at Scale*: **TRUE** or **FALSE**: the article argues that a simple way to deal with possibly-incorrect results (especially in the presence of non-determinism) in a distributed system is to send the request to multiple replica servers, and then return the most common result.

(h) Xu et al.'s *Do Not Blame Users for Misconfigurations*: Section 2 of the paper describes the design of SPEX, a tool for:

A inferring configuration constraints

B injecting errors to expose misconfiguration vulnerabilities

C detect some kinds of error-prone configuration design and handling

D all of the above

E both **A** and **B**

F both **A** and **C**

(i) Terrell et al.'s *Gender differences and bias in open source: pull request acceptance of women versus men*: Which of the following did the study find (select all that apply):

A across all studied PRs, women's PRs were accepted more often than men's

B across all studied PRs, mens's PR were accepted more often than women's

C considering only PRs submitted by project outsiders, women's PRs were accepted more often than men's

D considering only PRs submitted by project outsiders, men's PRs were accepted more often than women's

VII. Extra Credit. Questions in this section do not count towards the denominator of the exam score.

43. (1pt) In section III (Matching), there is a theme to the names used in the situation descriptions. What is the theme? **First names of American presidents.**

44. (1pt) In section III (Matching), one of the first names is associated with the answer to the question. Which question is it, and what is the last name of the associated person (i.e., the last name associated with the answer, not with the answer to the previous question)? **Question 17: "Richard" is "Richard Nixon" wrt presidents, but "Richard Stallman" wrt free software. The reading for the "free and open-source software" lecture was written by Stallman, and his name came up at least twice during the lectures themselves.**

45. (1pt) Curriculum design: support or refute the following claim: we discuss "CS theory" so often in this class because it is very common for software engineers to encounter undecidable problems in their day-to-day work. **Likely refute. Undecidable problems are rare in practice, but it is extremely important to be able to recognize them when they occur, so we over-emphasize them in class. As an analogy to training a machine learning model, this is like synthetic minority oversampling: we make you deal with the problem often enough in class (=training) that when it does occur in practice, you'll recognize it.**

46. (1pt) Would you be willing to serve on an engineer panel in the future (i.e., once you have a job)? If so, leave an email address that you expect to still monitor after graduating in this space. If not, write "no" to receive full credit. **Any email address or "no" receives the point.**

For the remaining extra credit points, answer additional questions from Section VI ("Your Choice" Reading Quiz). You may answer up to five additional times. However, if you get *any* of these questions wrong (including the question in Section VI), you will receive *no credit for any "Your Choice" Reading Quiz questions*. So, you should only answer questions about readings that you actually did read!

47. (1pt) Write the letter of the question you are choosing to answer: **Any letter.**
    Write the answer to that question: **The answer.**

48. (1pt) Write the letter of the question you are choosing to answer: **Any letter.**
    Write the answer to that question: **The answer.**

49. (1pt) Write the letter of the question you are choosing to answer: **Any letter.**
    Write the answer to that question: **The answer.**

50. (1pt) Write the letter of the question you are choosing to answer: **Any letter.**
    Write the answer to that question: **The answer.**

51. (1pt) Write the letter of the question you are choosing to answer: **Any letter.**
    Write the answer to that question: **The answer.**

Document A:

> One of the services that the Amazon cloud provides is called S3, which is a data storage service. Imagine a hypothetical scenario where S3 had a major outage, and Amazon's explanation of the outage was "a hard drive failed".
>
> Engineers wouldn't believe this explanation.
>
> - Lorin Hochstein

Document B:

from: coworkerb@yourcompany

to: yourteam

Subject: Equality on java.util.Optional

Dear team,

I've noticed that our style guide doesn't say anything about whether it's okay to call `.equals()` on `Optional`s, and I've noticed a few places in our code where we do call that method on an `Optional`. I don't think we should do that, so I'd like to add a provision to the style guide that forbids it (and maybe enable a static analysis check in one of our linters, like ErrorProne, to enforce it).

The Javadoc for the `Optional` type (**Document D**) is pretty clear that comparing equality on optionals "may have unpredictable results". Also, I found a blog post (**Document E**) that makes the same point: it says "we shouldn't use [equals] directly with Optionals".

Thanks, [Coworker B]

Document C:

from: coworkerc@yourcompany

to: coworkerb@yourcompany

cc: yourteam

Subject: re: Equality on java.util.Optional

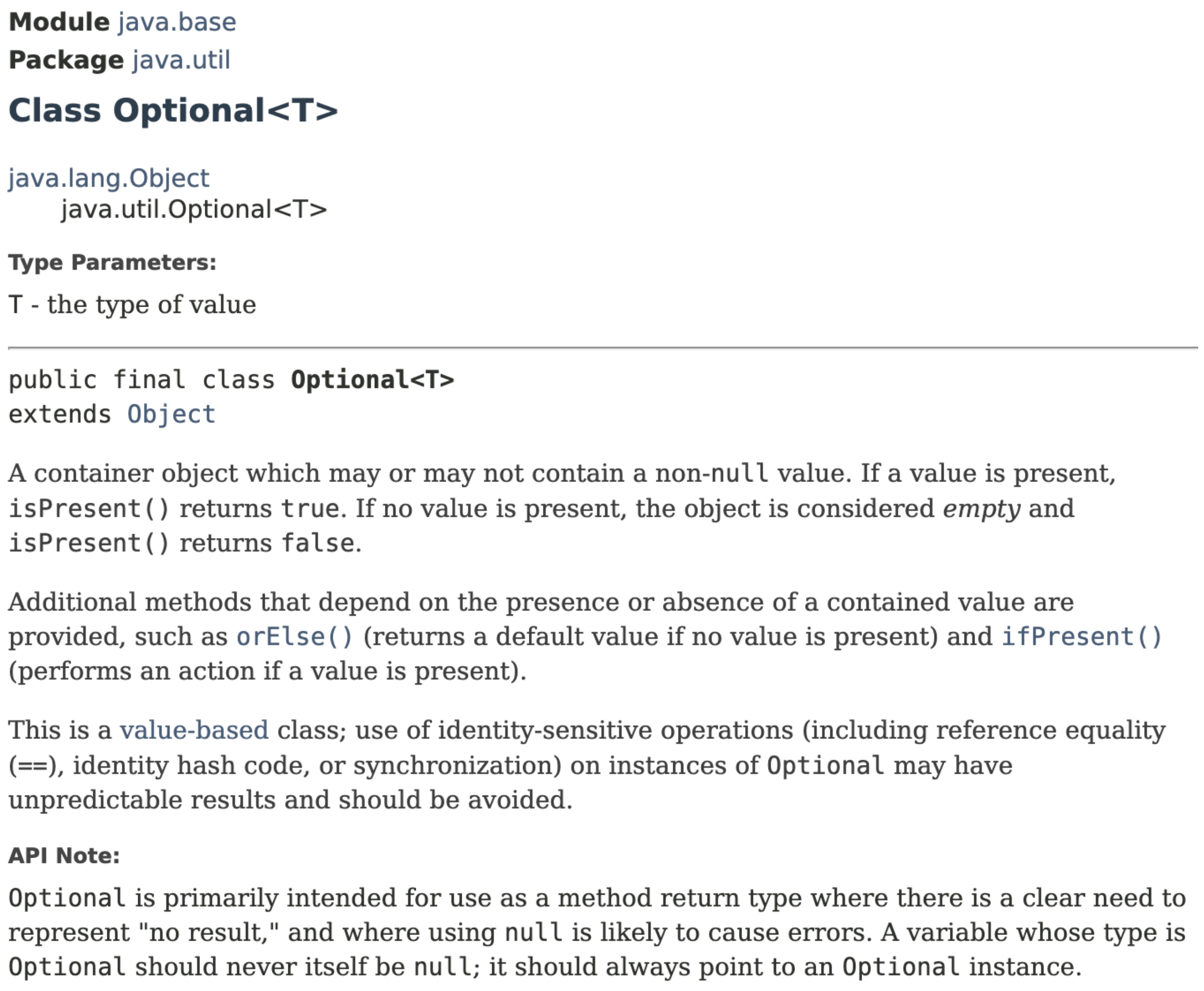
[Coworker B],

You are wrong about this: it's fine to compare `Optional`s with `equals`! ErrorProne **already** has a rule that enforces this (**Document F**), even, so your suggestion to make ErrorProne enforce the opposite doesn't make a lot of sense. If it's good enough for Google (they wrote ErrorProne, you'll recall), it's probably good enough for us. And, anyway, there are plenty of blogs that say it's okay to use `equals`, too; for example, I found one (**Document G**) with just a few minutes of searching. After all, anyone can write a blog!

There's no need to change the style guide. But, if you insist on a change, we change it to explicitly say that `equals` on an `Optional` is fine.

[Coworker C]

Document D:



Module java.base
Package java.util

Class Optional<T>

java.lang.Object
    java.util.Optional<T>

Type Parameters:

`T` - the type of value

```
public final class Optional<T>
extends Object
```

A container object which may or may not contain a non-`null` value. If a value is present, `isPresent()` returns `true`. If no value is present, the object is considered *empty* and `isPresent()` returns `false`.

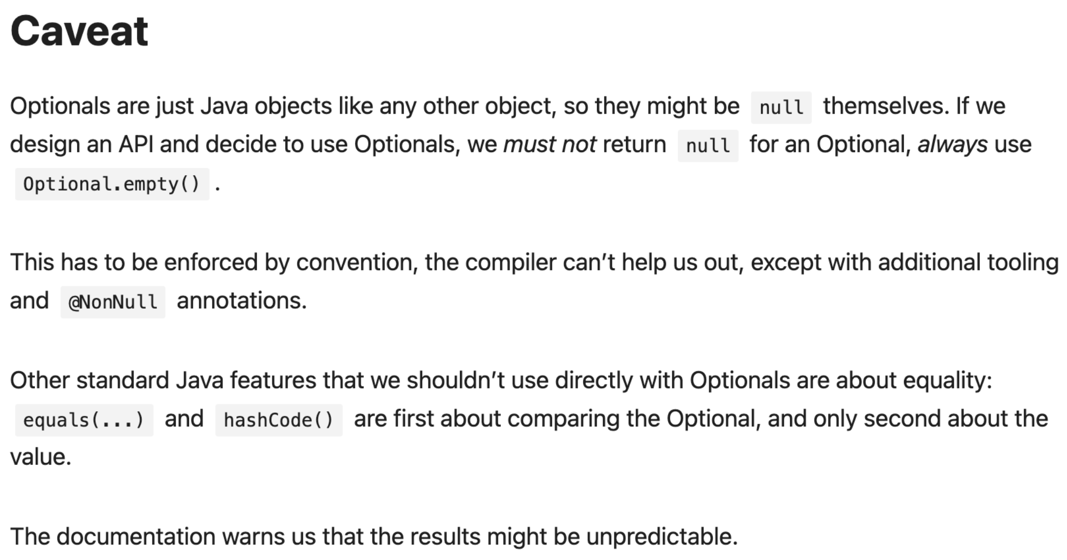
Additional methods that depend on the presence or absence of a contained value are provided, such as `orElse()` (returns a default value if no value is present) and `ifPresent()` (performs an action if a value is present).

This is a value-based class; use of identity-sensitive operations (including reference equality (==), identity hash code, or synchronization) on instances of `Optional` may have unpredictable results and should be avoided.

API Note:

`Optional` is primarily intended for use as a method return type where there is a clear need to represent "no result," and where using `null` is likely to cause errors. A variable whose type is `Optional` should never itself be `null`; it should always point to an `Optional` instance.

Document E:



Caveat

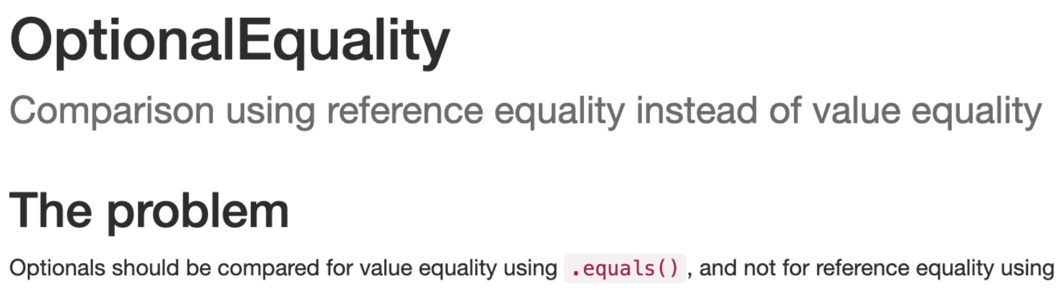
Optionals are just Java objects like any other object, so they might be `null` themselves. If we design an API and decide to use Optionals, we *must not* return `null` for an Optional, *always* use `Optional.empty()` .

This has to be enforced by convention, the compiler can’t help us out, except with additional tooling and `@NonNull` annotations.

Other standard Java features that we shouldn’t use directly with Optionals are about equality: `equals(...)` and `hashCode()` are first about comparing the Optional, and only second about the value.

The documentation warns us that the results might be unpredictable.

Document F:



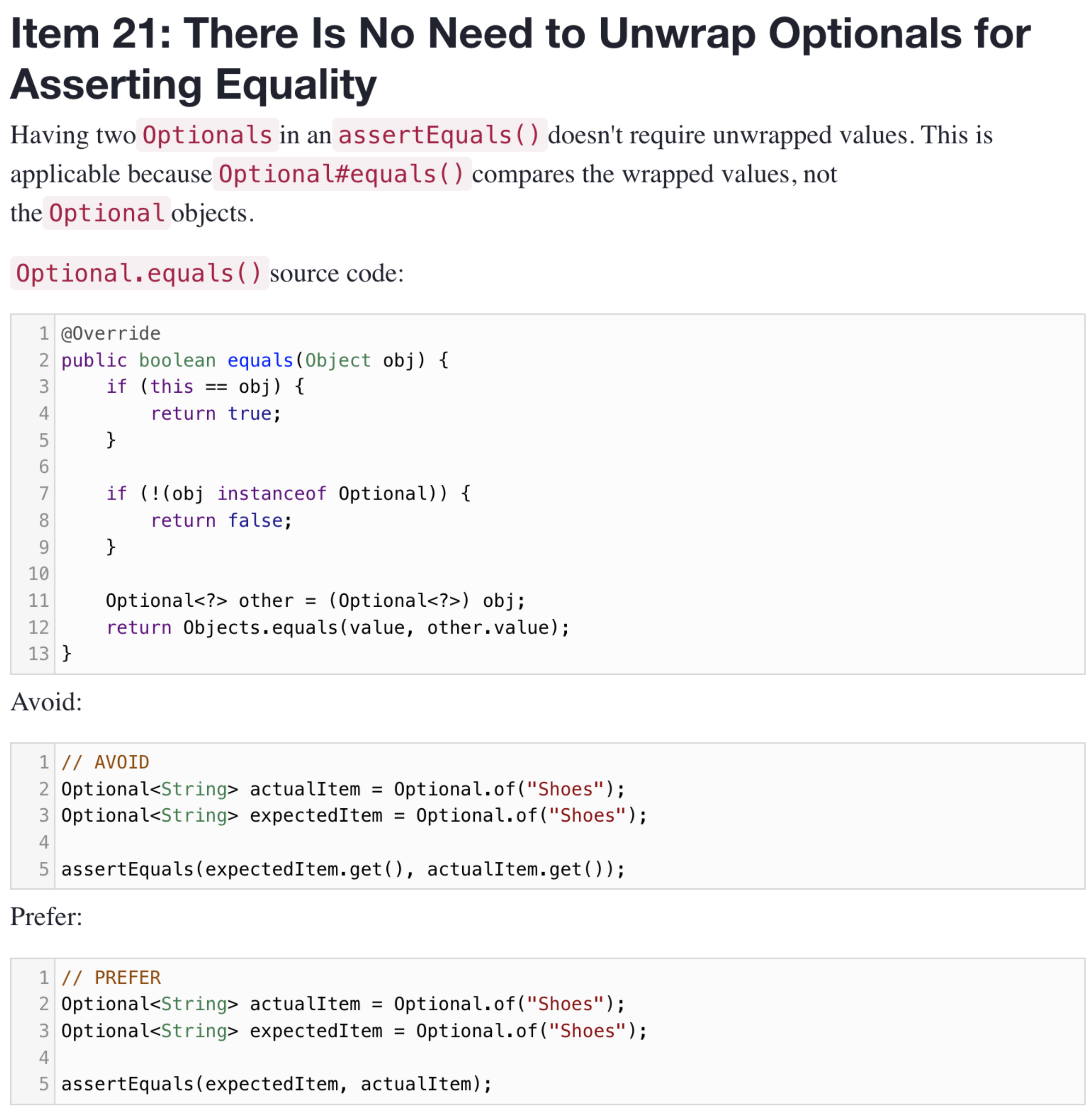
OptionalEquality

Comparison using reference equality instead of value equality

The problem

Optionals should be compared for value equality using `.equals()`, and not for reference equality using `==` and `!=`.

Document G:



Item 21: There Is No Need to Unwrap Optionals for Asserting Equality

Having two `Optionals` in an `assertEquals()` doesn't require unwrapped values. This is applicable because `Optional#equals()` compares the wrapped values, not the `Optional` objects.

`Optional.equals()` source code:

```java
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }

    if (!(obj instanceof Optional)) {
        return false;
    }

    Optional<?> other = (Optional<?>) obj;
    return Objects.equals(value, other.value);
}
```

Avoid:

```java
// AVOID
Optional<String> actualItem = Optional.of("Shoes");
Optional<String> expectedItem = Optional.of("Shoes");

assertEquals(expectedItem.get(), actualItem.get());
```

Prefer:

```java
// PREFER
Optional<String> actualItem = Optional.of("Shoes");
Optional<String> expectedItem = Optional.of("Shoes");

assertEquals(expectedItem, actualItem);
```