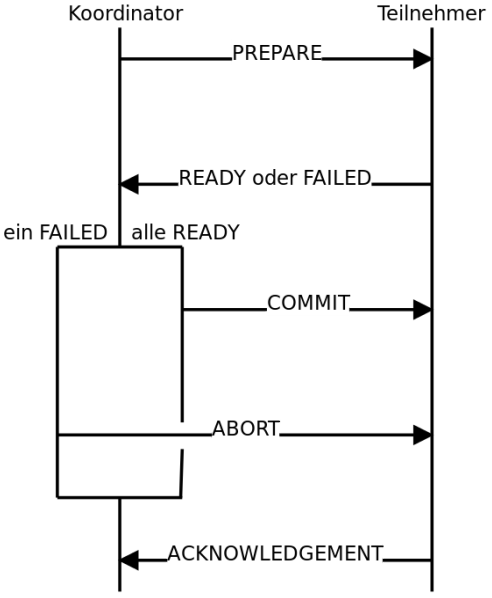
Code Review Exercise

Hand-in Date: 30.11.2018

Group: aweSoME
Alphonse Mariyagnanaseelan - 15-712-698,
Anna Katharina Fitze - 11-931-722,
Clara-Maria Barth - 15-931-041,
Jürg Bargetze - 12-729-901,
Timucin Besken - 14-924-609

Project Description

Abstract. This project was made by a team around Anna Katharina Fitze during the course “Distributed Database System“ at the University of Zurich during the fall semester 2018.



Two-Phase-Commit-Protocol

The program we chose to analyse is an implementation of the Two-Phase-Commit Protocol (2PC). It consists of several classes, the most important classes are the subordinate-, coordinator- and the logger-class. Due to the size of the class, we will only analyse the coordinator-class.
The 2PC is a protocol that ensures that all nodes (aka subordinates) participate in a transaction. If any subordinate cannot participate in a transaction, the transaction will not take place. To find an agreement over this situation, the 2PC was initialised. The first phase starts with the coordinator sending a PREPARE message to the subordinates. The subordinates will send back either a YES message or a NO message. With the coordinator receiving every YES/NO Vote, the first phase ends. If all votes from the subordinate were YES votes, the coordinator will send COMMIT to all subordinates. If one or more subordinates sent back a NO vote, the coordinator will send an ABORT message to all subordinates. The subordinates anyway send back an ACK (acknowledgement). The transactions and the second phase end with that ACK. The Coordinator as well as the subordinates log their messages before they send them, just in case any message will get lost or a node will crash. There are also implemented failure-handling which are not important for this analysis. We are considering the class Coordinator.java.

Possible Criteria

During our code review we tried to take a broad look in terms of criteria we checked the code for:

- Code Statements (understandable variables, units, bounds, spacing etc.)
- Comments/Documentation (no needless, obsolete, redundant comments, no code commented out etc.)
- Logic (loops and branches, bounds, correct conditions, division by zero, all cases covered etc.)
- Error Handling (error messages understandable and complete, edge cases etc.)
- Code Decisions (code at right level of abstraction, methods have appropriate number types of parameters, no unnecessary features, redundancy minimized etc.)

Defects Found

We took a look at the class called *Coordinator.java* which is part of the Prototype2PC project. The class has 596 lines of code whereas 249 are empty.
Generally we recommend to delete most of these empty lines and add comments to improve the code. Also this class is too big. They use the ExceptionHandler in an inappropriate way as they catch the whole function.

In the following table we list the defects we found during the manual review:

Line	Code	Defect	Recommendation
32-43		Some attributes are instantiated in the constructor but others are instantiated by default	Make it consistent
40	`this.loggedDecision = "";`		Use something else than a string e.g. an enum
45-66	[Broadcast method]	Switch case is not well readable	Switch is hard to read here, better us if/else
48,61,72		Why do you need to always add \n to msg?	Instead of repeating it every time you could just put msg = msg + “\n” at the beginning of the method
72-73	`writer.write(msg + "\n");` `writer.flush();`		Combine it
90, 132, 517			This could be moved directly in line 86 (resp. 134 and at 119 initialize with 1, resp. 519) but you might lose readability
142-205		This method should handle the error but the catch statement is missing	Also you’re already telling the compiler that the method can throw an exception, so handling the error is not really needed (but it might be preferred). Also the method should be shortened

146	`coordinatorLog.readLogBottom().split(" ")[0].equals("COMMIT")`	This logic should be moved to the Logger class, so that we don't have to worry about the implementation details and instead call it like:	`coordinatorLog.isLatestMsg("COMMIT")`
168-172		This is never ever possible	Test should be at a different location (e.g. in the Logger class)
174		Variable temp isn't helpful as a variable name and it's used only once.	Directly subtract it from `maxSubordinates`
177	`Printer.print("\nWaiting for " + maxSubordinates + " subordinate(s) to reconnect...\n", "white");`	Note that the concrete color used here (`"white"`) was not even defined in the `Printer` class, it coincidentally falls back to the case where given color was not found.	Here, we could also set a default parameter, which uses white as font color
185-191		Does exactly the same as 177-181:	This can be moved after the if/else statement
207-238		There are some repetitions of code in both if and else part.	Put the same lines of code outside the conditional clause
213, 220, 318, 433	`Printer.print("========== ===== COORDINATOR CRASHES ================\n", ...;`		add "==" in the end, to make sure all are the same size (→ consistency/uniform) Besides, better add a method called printTitle to class "Printer"
224	`if (scanner.nextLine().toUpperCase().equals("")) {`	".toUpperCase" if you're checking if it's an empty string it's not needed.	Remove `.toUpperCase`
224, 253, 255, 262 etc.		Possible *NullPointerException*	`if("".equals(scanner.nextline()))`
231	`Printer.print("", "");` `Printer.print("========== ===== COORDINATOR CRASHES ================\n", "red");`	Printing an empty string	Combine it so there is one line less
246, 247			Instantiation could be done with definition
251		if you need the index, would be better to use a normal for loop. What happens if msg.equals("Y")?	It should be if, else if, else if, else to be improve readability. Other solution: Switch/case
276-312		You're checking twice for if (decision), also it's impossible to have both cases that all subordinates answered yes and one did not vote but from the code structure it seems like it's possible.	If, else, else if … to improve readability
277-278 283-284		String is separated with +	Put it on one line

288	`this.coordinatorLog.log("COMMIT", true, true, true);`	When reading the code, it is unclear which parameter is responsible for what, since they're all Boolean. Also, since the parameters are mostly the same, it might be sensible to set default parameters	In Java we can use function overloading, e.g:: `this.coordinatorLog.log("COMMIT", true);` Note that in this case the best approach might be to use something like the "Builder" design pattern.
296	`this.sockets.size() == 0`		Instead of comparing `size()` we can use `isEmpty()`, which is not only simpler but also better communicates the intention.
304-306		Mixed responsibilities	This should be done in method phaseOne(). CheckVotes instead of void should return bool decision.
336	`System.out.print("");`	Unnecessary empty string	Remove
404	`crashedSubordinateIndices.size() > 0`	Readability	Similar to L296, this should instead be `!crashedSubordinateIndices.isEmpty()`
444	`coordinatorLog.readLogBottom().split(" ")[0]`	Readability and law of demeter	Similar to L146, it could be: `coordinatorLog.getLatestMsg()`
455	`case "COMMIT":` `case "ABORT":`	Fallthrough	At least add comment
489, 493		Checks for same thing	Could be merged in only one statement
517			could be moved to L519 directly

Checkstyle:

There are the same defects more than once, I chose one respectively to avoid redundancy. Moreover the defect that the line is to long (upper limit 80 characters) has shown up very often, but is not mentioned in the table.

Defect	**Example**	**Occurences**
The class coordinator should be declared as final.	`public class Coordinator {`	1
The parameter xy should be declared as final.	`private Coordinator(int maxSubordinates) throws IOException {`	8
The variable 'maxSubordinates' hides a field.	`private Coordinator(int maxSubordinates) throws IOException {`	1
After "if" there is an empty space missing.	`if(this.coordinatorLog.readLogBottom().split(" ")[0].equals("COMMIT") ||`	7

	`this.coordinatorLog.readLogBottom().split(" ")[0].equals("ABORT"))`	
'||' should be on a new line.	`if(this.coordinatorLog.readLogBottom().split(" ")[0].equals("COMMIT") || this.coordinatorLog.readLogBottom().split(" ")[0].equals("ABORT"))`	6 (also with other operators)
The construct “if” must use curly brackets.	`for (String msg : votes) { if (msg.equals("N")) decision = false;`	16
The inline-conditional-operator should be avoided.	`boolean userInputPresent = false; String decisionMessage = decision ? "COMMIT" : "ABORT";`	1
The magic number ‘2000’ should be defined constant.	`System.out.print("Please press enter within " + Coordinator.TIMEOUT_MILLIS /2000 + " seconds to broadcast \"" + decisionMessage + "\" to the subordinates: ");`	1
Before/After ‘+’ there is an empty space missing.	`unreachableSubordinatesIndices.add(index); Printer.print("Unable to reach S" + (index+1) + " waiting for it to reconnect...",`	4 (also with other Operators)

PMD:

Defect	**Occurences**	**Lines**
When doing a String.toLowerCase()/toUpperCase() call, use a Locale	2	221, 339
A class which only has private constructors should be final	1	12
All classes and interfaces must belong to a named package	1	12
Position literals first in String comparisons	8	249, 251, 258, 258, 258, 388, 392, 392,
This statement should have braces	16	249, 269, 274, 277, 280, 304, 348, 406, 419, 431, 442, 456, 463, 488, 493, 546
Useless parentheses	5	329, 340, 580, 580, 580

FindBugs:

Defect	**Occurrences**	**Lines**
Reliance on default encoding which can cause the application behavior to vary between platforms (call to method that performs byte to string or string to byte conversion)	2	36, 328
system.exit shuts down the JVM which makes it hard or impossible for code to be invoked b other code; throw a runtime exception instead	1	365
The are precondition missing that check if the input given is not null therefore the methods throw IOExceptions	4	77, 371, 497, 578

Summary of the Recommendation

The overall codebase is inconsistent in implementation, many design patterns are violated. The program obviously grew organically, since it is called prototype. We think before implementing it would have been useful to create a class diagram and analyze it according to design principles to make the project extendible, readable and easy to maintain. The overall structural problems were not discovered by the tools. We optimized the manually found defects but not the tool found ones. We would actually recommend to rewrite the whole class instead of spending tedious time fixing simple defects. And there would also be the option for indentation to use a prettifier which would fix that problem.
The tools we used were quite useful and found a lot of problems but are rather stilistic ones.

Review Time and Defects Found

Every member of the group had to take a look at the code individually - it took everyone at least two hours. Three of us took a look at it manually and two of us used review tools. As you can see in the first table we found more defects manually and also there were lots of redundancies. Manually we found rather architectural issues.