Software Engineering

Assignment 3

Kunal Kumar

170010012

Part 1

Short essay on style checking for java programming, giving an overview of the important practices and the reasons

Java is one of the most popular programming languages – be it Win applications, Web Applications, Mobile, Network, consumer electronic goods, set top box devices, Java is everywhere.
More than 3 Billion devices run on Java. According to Oracle, 5 billion Java Cards are in use.
More than 9 Million developers choose to write their code in Java and it is very popular among developers as well as being the most popular development platform.

Need for style checking:

Almost always you will join teams working on existing software and there is a pretty good chance that most of the authors have left or switched to different projects, leaving you stranded with portions of code that make you question humanity. There are around 14 standards of designing in coding. Problem occurs when you work with large team, and you use different designing pattern and others are using different, It becomes very difficult to understand others code and do required changes. Reuse at code level is common in software development.

Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard. Style checker can check many aspects of our source code. It can find class design problems, method design problems. It also has the ability to check code layout and formatting issues.

Idea of Design patterns :

Researchers working in Object Oriented Software Engineering field found that design patterns can be formulated as solutions to commonly occurring problems in object oriented design.

1. Solutions to commonly occurring design problems.
2. Object oriented designs and not code in specific programming languages.
3. Represented abstractly, explained with examples.
4. Abstract design is mapped to a concrete problem.
5. Provide the structure for the pattern
6. Design of different Classes/Interface and Module.
7. Organised with documentation comment
8. Declaration of packages with comment

Classification of Patterns:

1. Creational Patterns – concerned about ways to create new objects
2. Structural Patterns – concerned about the composition of objects and classes
3. Behavioral Patterns – concerned about ways in which objects interact

Benefits:

We can add new functionality without changing structure of an object. We can define new class which wraps older whose methods are kept unchanged.

Prefer and Avoid:

Formatting and indentation are all about organizing your code to make it easy to read, and it includes spacing, line length, wraps and breaks and so on

Indentation:

Use 2 or 4 spaces and stay consistent. This will help to understand the portion of a class, subclass or module.

Line Length:

Upto 70 to 110 characters depending on affect on readability. It's important to eliminate the need for horizontal scrolling and place line breaks after a comma and operator. As the computer width is of some fixed size, if we write more characters in a single line , we will have to scroll the screen which is not a good practice.

Overview of Important practices with reasons:

Prefer returning Empty Collections instead of Null

If a program is returning a collection which does not have any value, make sure an Empty collection is returned rather than Null elements. This saves a lot of "*if else*" testing on Null Elements.

Use Strings carefully

If two Strings are concatenated using "+" operator in a "for" loop, then it creates a new String Object, every time. This causes wastage of memory and increases performance time. Also, while instantiating a String Object, constructors should be avoided and instantiation should happen directly.

Avoid unnecessary Objects

One of the most expensive operations (in terms of Memory Utilization) in Java is Object Creation. Thus it is recommended that Objects should only be created or initialized if necessary.

Dilemma between Array and ArrayList

Developers often find it difficult to decide if they should go for Array type data structure of ArrayList type. They both have their strengths and weaknesses. The choice really depends on the requirements. Arrays have fixed size but ArrayLists have variable sizes. Since the size of Array is fixed, the memory gets allocated at the time of declaration of Array type variable. Hence, Arrays are very fast. On the other hand, if we are not aware of the size of the data, then ArrayList is More data will lead to ArrayOutOfBoundException and less data will cause wastage of storage space. It is much easier to Add or Remove elements from ArrayList than Array. Array can be multi-dimensional but ArrayList can be only one dimension.

Avoiding Memory leaks by simple tricks

Memory leaks often cause performance degradation of software. Since, Java manages memory automatically, the developers do not have much control. But there are still some standard practices which can be used to protect from memory leakages.Always release database connections when querying is complete. Try to use Finally block as often possible. Release instances stored in Static Tables.

Importing Packages or Library

We should avoid importing like "*import java.util.**" as it will import unnecessary packages, which will increase run time of our code. Import only those packages which you are going to use.

If-checks:

IMO writing well-formatted code makes it easy to spot typos and errors to the author and the code reviewers.

Switch:

When it comes to switch , it's best to always have a default case even without code. Use /* *falls through* */ to indicate the control falls to next case.

Exception messages:

When throwing an exception write good and poorly indented messages.

Iterators and Streams:

Streams are becoming more common and at times it can be very complex hence, it's important to indent for easy to read.

Declarations and Assignments:

One declaration per line is recommended since it encourages comments. Put declarations only at the beginning of blocks (A block is code surrounded by curly braces { and }). Do not wait to declare variables until their first use; it can confuse the unwary programmer and hamper code portability within the scope.It's also important to avoid local declarations that hide declarations of the higher-levels and is to avoid confusions.

Spacing & line breaks

Avoid the temptation of saving 1–2 lines of code at the expense of readability. Here are all the best practices when it comes to spacing and blank lines (A white space does make a difference).
One blank line between methods and Spring developers recommends two blank lines after constructors, static block, fields and inner class. Space pad operators i.e. Use int foo = a + b + 1; over int foo=a+b+1. This help user in understanding code and visualising. Separate all binary operators except "." from operands using a space. Open brace "{" appears at the end of the same line as the declaration statement or method and closing brace "}" starts a line by itself indented.

Documentation and Comments

It's worth mentioning that almost all code goes changes throughout its lifetime and there will be times when you or someone is trying to figure out what a complex block of code, a method, or a class is intended to do unless clearly described. There are times that the comment on a complex piece of code, method, class does not add any value or serve its purpose. This usually happens when commenting for the sake of it. Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Let's get started. There are two types of comments.

Implementation Comments — are meant to comment out code or comment about a particular implementation of the code.
Documentation Comments — are meant to describe the specification of the code from an implementation-free perspective to be read by developers who might not necessarily have the source code at hand.

The frequency of comments sometimes reflects poor quality of code. When you feel compelled to add a comment, consider rewriting the code to make it clearer.

Code can change hands numerous times in its lifetime, and quite often the original author of a source file is irrelevant after several iterations. We find it's better to trust commit history and OWNERS files to determine ownership of a body of code.