DOWNLOAD PDF BEST CODING PRACTICES IN JAVA

Chapter 1 : Best coding practices - Wikipedia

Best coding practices You're now about halfway through this learning path. You have enough Java syntax under your belt to write basic Java programs.

These are interesting topics both from an academic and from a practical point of view. Try to avoid returning null from API methods whenever possible. Your API consumers should be able to chain methods whenever applicable: Whether you match something or not is irrelevant to the next method call. Nulls often arise also because of lazy initialisation. In many cases, lazy initialisation can be avoided too, without any significant performance impact. In fact, lazy initialisation should be used carefully, only. If large data structures are involved. Avoid returning nulls from methods whenever possible. Never return null arrays or lists from API methods While there are some cases when returning nulls from methods is OK, there is absolutely no use case of returning null arrays or null collections! An array of strings naming the files and directories in the directory denoted by this abstract pathname. The array will be empty if the directory is empty. So this is wrong in three ways: Arrays or Collections should never be null. All relevant state is transferred in each request and in each response. This is essential to the naming of REST: This is awesome when done in Java as well. Think of it in terms of rule number 2 when methods receive stateful parameter objects. Things can be so much simpler if state is transferred in such objects, rather than manipulated from the outside. Take JDBC, for instance. The following example fetches a cursor from a stored procedure: Each object is incredibly stateful and hard to manipulate. Concretely, there are two major issues: It is believed that the above usage fulfils what is known as Fair Use The Rule: Implement more of a functional style. Pass state through method arguments. Manipulate less object state. Short-circuit equals This is a low-hanging fruit. Short-circuit all your equals methods to gain performance. Try to make methods final by default Some will disagree on this, as making things final by default is quite the opposite of what Java developers are used to. If you do need to override a method do you really? Of course, you probably want to restrict the actual type to something more confined in a real-world situation, e. T can always be inferred to Object. In fact, you might as well just not use generics with the above methods. More importantly, you may think that you can overload the above method, but you cannot: But what happens to this call here? And if you cannot, never overload such a method. Conclusion Java is a beast. Unlike other, fancier languages, it has evolved slowly to what it is today. Stay tuned for more top 10 lists on the subject!

Chapter 2 : 9 Best Practices to Handle Exceptions in Java

Let's identify and learn some java best practices around software design, code and performance which can transform a piece of code into excellent programs.

Usually, if you work on a software project and especially at the beginning of it, the code quality is not the first thing you pay attention to. However, the question how to improve code quality and efficiency should be one of the main concerns for the developers, technical team lead, system architect, and even project manager. It is only a matter of time when it will catch you up and cause the problems if you will just leave it unaddressed. That is why it is much better to manage this problem beforehand. At Romexsoft, we commonly use Java for our projects and always look at the ways how to improve code quality in Java. We do this by following software quality improvement techniques and using Java Code Quality tools that help us not only to improve our code but also our productivity. In this article, we will take a look at the most used development methodologies and top-notch tools that we use to create better code. The intent is to consider the common approaches to code quality and help teams choose methodology and tool, that makes their process efficiently. What Is a Software Development Methodology? Software development methodology is a set of patterns and rules that are used to structure, plan, and control the process of software solutions developing. In order to manage the development process efficiently, the project manager must select the development methodologies that suit the appropriate project the most. Pair Programming Pair programming is a proven methodology that decreases bugs in your code. It is done by two developers who share one workspace a keyboard and a screen and it looks like a flashback from school or university â€" one developer is coding while another reads the written code and contemplates about potential issues. Developers coach each other in coding and can change roles during a session. The main benefit of pair programming is knowledge sharing and code quality improvement. It also improves communication inside of the team. Pair programming is not only about the code but it can be used for OS command lines, work with DB server, file management, work with cloud and many other cases. Coding Conventions Coding conventions are the set of guidelines that are developed by dev teams and include the recommendations for the programming style, practices, and methods for each aspect of a code that is written within the company or certain project. These conventions are usually specific for every programming language and cover file organization, indentation, comments, declarations, statements, white spaces, naming conventions, programming practices and principles, programming rules, architectural best practices, etc. The main advantage of defined standards is that every piece of code looks and feels familiar. It makes it more readable and helps programmers understand code written by another programmer. If the coding standards are followed by and applied consistently throughout the development process, in future, it will be easier to maintain and extend the code, refactor it, and resolve integration conflicts. The standard itself matters much less than adherence to it. Code conventions are important to programmers for a number of reasons: Hardly any software is maintained for its whole life by its author. Code conventions improve the readability of the software, allowing programmers to understand the new code more quickly. Frankly speaking, I am a big fan of coding standards. For me, it makes sense to spend time on debating and arguing over them, since it is a valuable contribution to save your time and efforts in the future. Start with arranging a meeting on what should be in the coding standard. It should be only a face to face meeting, but not over email or via the chat systems. Discuss the rules that you decided to include in your code convention. The code convention guidelines should be reviewed frequently. If some of them do not work as expected, then they need to be reworked or removed from the guidelines. Programmers are highly recommended to follow these guidelines during the coding and it should be used as part of the code review which is the next one in our list. Code Review The next best thing to pair programming is code review. If you do not practice pair programming then it is recommended to consider at least code review. It is a lightweight process that should be applied as soon as possible after the code is written. Some of the other subjects listed below are not so obvious but are worth to be considered. Unit Tests The next methodology we will talk about is unit testing. Depending on your experience you may or may not have heard something about unit tests, test-driven development or some other type of testing methodology. Usually, these methodologies are applied in the context of large software systems and sometimes in the context of simple web sites. In computer programming, unit testing is a software development process in which the smallest testable part of a source code, called unit, is tested individually and independently to examine whether they are working as expected. One of the benefits that you acquire from creating unit tests is that your team will be motivated to write testable code. Since unit testing requires the appropriate structure of your code it means that code must be divided into smaller more focused functions. Each of which is responsible for a single operation on a set of data rather than on large functions performing a number of different operations. The second benefit of writing a well-tested code is that you can prevent the future failure when the small changes to existing code break functionality. When a failure happens you will be informed that you have written something wrong. At first glance, spending time on writing the unit tests during development looks like extra expenses. However, it will save time you might spend on debugging. This should be a step by step process. Of course, you can not write unit tests for every part of your project but you need to make sure the core components behave as expected. When the project grows you can simply run the tests you have developed to ensure that existing functionality is not broken when new functionality is introduced. Test-Driven Development Test-driven development TDD , also called test-driven design, is a methodology of developing software that combines unit testing, programming, and refactoring of source code. Usually, developers tend to skip the architectural aspect of software development and jump straight into coding. They try to get a rough implementation and only after having a working prototype they think about writing the tests. While this approach can succeed it takes a lot of efforts to just make it work. TDD is one of those practices which contribute to better code quality and decreases bugs. It was introduced as part of a larger software design paradigm known as Scrum and Extreme Programming XP , which are types of the Agile software development methodology. TDD helps to produce applications of high quality in less time. Proper implementation of TDD requires the developers and testers to accurately anticipate how the application and its features will be used in the real world. When it is run as part of a continuous integration cycle with frequent automated builds and tests, the practice is Unit Testing on steroids. In order to apply this methodology, developers need specific training and coaching. In my understanding of TDD, before coding you need to know at least the starting point. First of all, you need to think of the objects and their behavior to solve the given problem. When doing TDD you start from a test and from your dream vision of how the test would look like. Continuous Integration Continuous Integration CI is a development practice that requires developers to integrate code into a shared repository several times a day SVN, Subversion, or Git. Each check-in is verified by the automated tests. Although automated tests are not strictly part of CI, they are usually anticipated. Such approach allows developers to detect problems earlier and, as a result, solve them faster. This is a valuable practice by itself. You should focus on setting up a simple Continuous Integration process as early as possible. There are many tools that can help you to set up this process and the most known are Jenkins , Bamboo , and Teamcity. They allow you to automate your software deployment and let you focus on building your product. Demo Session The demo review meeting usually takes place close to the end of the Sprint. The purpose of this meeting is to show the other team members, customers, and stakeholders the results of work team has accomplished over the Sprint. It may not be immediately visible why it leads to a better code but it will. By regularly showing the source code developers need to keep it close to the release state. With demo meetings on regular basis, you will have a well-organized process of receiving feedbacks. And this will give you a better understanding of what was done right and will indicate when something went in the wrong direction. Usually, the compiler catches the syntactic and arithmetic issues and lists out a stack trace. But there still might be some issues that compiler does not catch. These could be inappropriately implemented requirements, incorrect algorithm, bad code structure or some sort of potential issues that community knows from experience. The only way to catch such mistakes is to have some senior developer to review your code. Such approach is not a panacea and does not change much. All of them are usually used to analyze the quality and build some useful reports. Very often those reports are published by continuous integration servers, like Jenkins. Here is a checklist of Java static code analysis tools, that we use at RomexSoft in most of our projects. Checkstyle Code reviews are essential to code quality, but usually, no one in the team wants to review tens of thousands lines of code. But the challenges associated with manually code reviews can be automated by source code analyzers tool like Checkstyle. Checkstyle is a free and open source static code analysis tool used in software development for checking whether Java code conforms to the coding conventions you have established. It automates the crucial but boring task of checking Java code. It is one of the most popular tools used to automate the code review process. Checkstyle comes with predefined rules that help in maintaining the code standards. These rules are a good starting point but they do not account for project-specific requirements. The trick to gain a successful automated code review is to combine the built-in rules with custom ones as there is a variety of tutorials with how-tos. Checkstyle can be used as an Eclipse plugin or as the part of a built systems such as Ant , Maven or Gradle to validate code and create reports coding-standard violations. PMD PMD is a static code analysis tool that is capable to automatically detect a wide range of potential bugs and unsafe or non-optimized code. It examines Java source code and looks for potential problems such as possible bugs, dead code, suboptimal code, overcomplicated expressions, and duplicated code. Whereas other tools, such as Checkstyle, can verify whether coding conventions and standards are respected, PMD focuses more on preemptive defect detection.

Chapter 3 : Coderbyte | The #1 Website for Coding Challenges

10 Subtle Best Practices when Coding Java This is a list of 10 best practices that are more subtle than your average Josh Bloch Effective Java rule. While Josh Bloch's list is very easy to learn and concerns everyday situations, this list here contains less common situations involving API / SPI design that may have a big effect nontheless.

Modifications Authors who modified code with a description on why it was modified. The "description of the module" should be as brief as possible but without sacrificing clarity and comprehensiveness. However, the last two items have largely been obsoleted by the advent of revision control systems. Modifications and their authorship can be reliably tracked by using such tools rather than by using comments. Also, if complicated logic is being used, it is a good practice to leave a comment "block" near that part so that another programmer can understand what is exactly happening. Unit testing can be another way to show how code is intended to be used. Naming conventions[edit] Use of proper naming conventions is considered good practice. Sometimes programmers tend to use X1, Y1, etc. A variable for taking in weight as a parameter for a truck can be named TrkWeight or TruckWeightKilograms, with TruckWeightKilograms being the more preferable one, since it is instantly recognisable. See CamelCase naming of variables. Keep the code simple[edit] The code that a programmer writes should be simple. Complicated logic for achieving a simple thing should be kept to a minimum since the code might be modified by another programmer in the future. The logic one programmer implemented may not make perfect sense to another. So, always keep the code as simple as possible. In particular, it consumes 5 times more screen vertical space lines , and 97 characters versus 52 though editing tools may reduce the difference in actual typing. It is arguable, however, which is "simpler". The 2nd merely discards the braces, cutting the "vertical" size in half with little change in conceptual complexity. In most languages the "return" statements could also be appended to the prior lines, bringing the "vertical" size to only one more line that the 3rd form. The third form obviously minimizes the size, but may increase the complexity: It leaves the "true" and "false" values implicit, and intermixes the notions of "condition" and "return value". It is likely obvious to most programmers, but a novice might not immediately understand that the result of evaluating a condition is actually a value of type Boolean, or its equivalent in whatever language , and thus can be manipulated or returned. In more realistic examples, the 3rd form could have problems due to operator precedence , perhaps returning an unexpected type, where the prior forms would in some languages report an error. Thus, "simplicity" is not merely a matter of length, but of logical and conceptual structure; making code shorter may make it less or more complex. For large, long lived programs using verbose alternatives could contribute to bloat. Given how many times code might be viewed in the process of writing and maintaining, it might amount to a significant savings in programmer keystrokes in the life of the code. This might not seem significant to a student first learning to program. But when producing and maintaining large programs which often reach thousands or even millions of lines, it becomes apparent[citation needed] how much a minor code simplification might speed work[dubious â€" discuss], and lessen finger, wrist and eye strain, which are common medical issues suffered by production coders and information workers. Furthermore, the 3rd approach may allow similar lines of code to be more easily compared, particularly when many such constructs can appear on one screen at the same time. Finally, very terses layouts might better utilize modern wide-screen computer displays. Modern screens can easily display or more characters, allowing extremely long lines. Most modern coding styles and standards do not take up that entire width. Thus, if using one window as wide as the screen, a great deal of available space is wasted. On the other hand, with multiple windows, or using an IDE or other tool with various information in side panes, the available width for code is in the range familiar from earlier systems. It is also worth noting that the human visual system is greatly affected by line length; very long lines slightly increase reading speed, but reduce comprehension [1] and add to eye-tracking errors. Some studies suggest that longer lines fare better online than in print [2] , but this still only goes up to about 10 inches, and mainly for raw speed of reading prose. Otherwise the application will not run on a host that has a different design than anticipated. A careful programmer can parametrize such variables and configure them for the hosting environment outside of the application proper for example in property files, on an application server, or even in a database. Compare the mantra of a "single point of definition" [22] SPOD. As an extension, resources such as XML files should also contain variables rather than literal values, otherwise the application will not be portable to another environment without editing the XML files. For example, with J2EE applications running in an application server, such environmental parameters can be defined in the scope of the JVM and the application should get the values from there. Construction guidelines in brief[edit] A general overview of all of the above: Know what the code block must perform Maintain naming conventions which are uniform throughout. Indicate a brief description of what a variable is for reference to commenting Correct errors as they occur. Keep your code simple Code building[edit] A best practice for building code involves daily builds and testing, or better still continuous integration , or even continuous delivery. Software testing Testing is an integral part of software development that needs to be planned. It is also important that testing is done proactively; meaning that test cases are planned before coding starts, and test cases are developed while the application is being designed and coded. Debugging the code and correcting errors[edit] Programmers tend to write the complete code and then begin debugging and checking for errors. Though this approach can save time in smaller projects, bigger and complex ones tend to have too many variables and functions that need attention. Therefore, it is good to debug every module once you are done and not the entire program. This saves time in the long run so that one does not end up wasting a lot of time on figuring out what is wrong. Software deployment Deployment is the final stage of releasing an application for users. Some best practices are: Files and directories should be kept to a minimum. Keep only what is needed: The software configuration management activities must make sure this is enforced. Unused resources old or failed versions of files, source code, interfaces, etc. For delta-based deployments, make sure the versions of the resources that are already deployed are the latest before deploying the deltas. If not sure, perform a deployment from scratch delete everything first and then re-deploy. Adopt a multi-stage strategy: Depending on the size of the project, sometimes more deployments are needed. There must be a way to roll-back to a previous working version. Rely on automation for repeatable processes: Use a tool that is native to each operating system or, use a scripting language for cross-platform deployments. Consider everything routers, firewalls, web servers, web browsers, file systems, etc. Do not change deployment procedures and scripts on-the-fly and, document such changes: Wait for a new iteration and record such changes appropriately. Newer software products such as APIs, micro-services, etc. If other activities such as testing and configuration management are wrong, deployment surely will fail. Organizational, social, governmental considerations.

Chapter 4 : Java Coding Standards and Google Java Coding Standards

Stretching the limits of Java has its pitfalls, and it's also unavoidable for many advanced programming scenarios. jOOQ creator Lukas Eder offers 10 best practices for less common scenarios in API design and development.

By Lokesh Gupta Filed Under: Java Best practices Classes are the building blocks of any java application. If these blocks are not strong, the building i. This essentially means that not so well-written can lead to very difficult situations when the application scope goes up or application faces certain design issues either in production or maintenance. On the other hand, set of well designed and written classes can speed up the coding process by leaps and bounds, while reducing the number of bugs in comparison. In this tutorial, We will discuss SOLID principles in Java with examples which are 5 most recommended design principles, we should keep in mind while writing our classes. They also form the best practices to be followed for designing our application classes. Single Responsibility Principle The name of the principle says it all: This will give we the flexibility to make changes in future without worrying the impacts of changes for another entity. Not even utility global functions related to module. Better separate them in another globally accessible class file. This will help in maintaining the class for that particular purpose, and we can decide the visibility of class to specific module only. Single Responsibility Principle Example We can plenty of classes in all popular Java libraries which follow single responsibility principle. For example, in log4j, we have different classes with logging methods, different classes are logging levels and so on. In our application level code, we define model classes to represent real time entities such as person, employee, account etc. Most of these classes are examples of SRP principle because when we need to change the state of a person, only then we will modify a person class. In given example, we have two classes Person and Account. Both have single responsibility to store their specific information. If we want to change state of Person then we do not need to modify the class Account and vice-versa. Open Closed Principle This is second important rule which we should keep in mind while designing our application. Open closed principle states: If other developers are not able to design desired behavior due to constraints put by our class, then we should reconsider changing our class. Open Closed Principle Example If we take a look into any good framework like struts or spring, we will see that we can not change their core logic and request processing, but we modify the desired application flow just by extending some classes and plugin them in configuration files. For example, spring framework has class DispatcherServlet. This class acts as front controller for String based web applications. To use this class, we are not required to modify this class. Please note that apart from passing initialization parameters during application startup, we can override methods as well to modify the behavior of target class by extending the classes. For example, struts Action classes are extended to override the request processing logic. This requires the objects of your subclasses to behave in the same way as the objects of your superclass. This is mostly seen in places where we do run time type identification and then cast it to appropriate reference type. Spring provides property editors to represent properties in a different way than the object itself e. Spring can register one property editor for one data type and it is required to follow the constraint mandated by base class PropertyEditorSupport. So is any class extend PropertyEditorSupport class, then it can be substituted by everywhere base class is required. For example, every book has an ISBN number which is in always a fixed display format. For this requirement, we may write property editor in such a way â€” import java. Interface Segregation Principle This principle is my favorite one. It is applicable to interfaces as single responsibility principle holds to classes. Developer Alex created an interface Reportable and added two methods generateExcel and generatedPdf. Will he be able to use the functionality easily? He will have to implement both the methods, out of which one is extra burden put on him by designer of software. Either he will implement another method or leave it blank. This is not a good design. So what is the solution? Solution is to create two interfaces by breaking the existing one. They should be like PdfReportable and ExcelReportable. This will give the flexibility to user to use only required functionality only. It has different listener classes for each kind of event. We only need to write handlers for events, we wish to handle. Some of the listeners are â€” FocusListener.

Chapter 5 : 11 Best Practices and Tools to Improve the Java Code Quality â€“ Romexsoft

In every programming language, there is a list of best coding practices and Java is no different. While some practices are non-negotiable, others are more based on the personal style of the individual programmer.

Simply finding the right way to express the rules for a business in the logical structure of software is hard enough. Many software projects end successfully if they manage to do this alone. But getting it right is often not enough anymore. As software becomes more and more essential to doing business, the more important it is for the software to generate and deliver the right answer, immediately. Some gate agents were even writing out tickets by hand, in ink, just like they did in the s before the age of computers. Java performance tools and teams survey: Who is doing what? While the tools for building websites and web services are better than ever, using them to produce something quick and accurate requires more skill and attention than ever before. This article will look at some of the most useful ways to deliver working solutions. Testing in the Agile Era: Some programmers see their work as a craft and feel that creating the best performing code is an art. They study the problem and then channel their experience through their intuition and end up building something that works well. Getting it right can be a bit of a game, a trick delivered by a master with effortless grace. They begin testing their code early and often, using the results to steer the development on track and toward the goal. Actual best practices are often a mixture of both these approaches. A good Java team will spend enough time planning the architecture so the code will seem like a work of art. Is there an ideal mix? Each project requires different skills. The old design patterns can be the best. They will only work with a mixture of creative architectural planning and thorough experimentation. These will usually require heavy testing to ensure that there are no hidden issues or unanticipated complications that throttle performance. Your job is to figure out how much time to devote to each of them. Plan the app and its test The first step is to plan the application and the performance testing. You must identify where the data comes from, where it must end up, and how the software needs to transform that data along the way. Identify potential delays Just as every chain has a weakest link, every piece of code has a slowest section. Sometimes the delay is obvious, such as in code that does a great deal of background processingâ€”for example, a photo sharing website that immediately constructs thumbnails of every new photo. Sometimes finding a bottleneck is difficult because it changes with time. Code that relies upon outside resources like databases or web services may run smoothly most of the time until the resource fails. Before adding logging functions or installing profiling code, start by sketching out the flow of data through the system. Chart where it enters, where it goes, and where it ends up. Mark the sections that you can test directly and note the sections which are out of your control. This flowchart will not offer guaranteed answers but it will be a road map for exploring. Most of your troubles will be caused by the slowest part of your code. Identify it, plan for the delay, and try to minimize it. Identify shared resources While every developer dreams that their code might run in a pristine environment, one devoted to the execution of the code alone, this is rarely the case. There are often different tasks competing for same resources: Sometimes what has to be shared can be as simple as the background work of the operating system; other times it can be another web application that runs on the same server. Finding the shared resources for competing tasks can be harder than ever thanks to virtualization. Scott Oaks, author of Java Performance: The Definitive Guide , reminds us, "A systemic, system-wide approach to performance is required in Java environments where the performance of databases and other backend systems is at least as important as the performance of the JVM. Taking a system-wide approach to performance means understanding and mastering shared resources, including virtual resources. Watch complexity Does your boss want the software to solve different problems all at once? Do the users expect that the software will do as many things as a Swiss Army Knife? Many performance problems begin when we ask the software and the machine to juggle too many things and produce too many different types of answers. This makes planning simpler and analysis more effective. Alas, this is often easier said than done. Some applications seem inherently integrated and impossible to separate. But often a bit of analysis can identify parts of the application that can work independently. Separating them will produce smaller, less complex sections. The entire application may be just as complex, but the programmers can concentrate on the smaller sections with their part of the complex mechanism. Some are beginning to use the term "microservices" to describe an architecture made up of a number of small programs that answer questions independently. When possible, separate the application into smaller apps that are as independent as possible. Watch database calls When RebelLabs asked developers about their biggest performance headaches, slow databases were at the top of the list. Over half the respondents Another subset said that the database was the root of the problem but that the software developers were partly to blameâ€” Modern databases provide complex solutions for keeping data organized. When users update records or interact with the application, they end up adding or updating rows of data and this will force the database to update indices that it uses to quickly locate particular rows. In many cases, the databases do the bulk of the work and execute the bulk of the machine instructions in the stack. Keeping the data safe, collated, and quickly accessible is a big challenge for your stack, which is why planning the interaction with the database is often the biggest job for anyone looking to maximize performance. More on this in the next tip. Watch the interaction with the database Databases can run slowly for a number of reasons. Sometimes they answer requests from multiple servers and they are pulled in too many directions. In many architectures, the database is the central clearinghouse that synchronizes the work of multiple servers. The load can skyrocket when all the servers start sending requests and transactions at the same time. In other cases, the queries are more complex than the database can handle quickly. If a query includes multiple join operations or it demands complex sorting, the answer may be slow to arrive because the database needs to juggle too much information. It must often pull data from multiple tables into memory to complete these processes and when the operations are large and the data overflows the cache, the operation becomes limited by the speed of the hard disks. In many cases, it makes sense for the developers and the database administrators to reexamine the queries and decide whether they need to be as complex as they are. The developers need to plan their requests and do their best to streamline them to prevent overloading. The database administrators must anticipate these needs and structure the database to be able to deliver the requests. There are many options for simplifying the structure and adapting it to the needs of the users. Sometimes the joins can be postponed or eliminated by presenting a smaller amount of data to the user who will then drill deeper if the extra information is needed. Paying attention to the complexity of the queries is one of the simplest ways to reduce the load on the database server. Choosing the right size for the database and the server is also important. The best practices are evolving quickly as more complex and elaborate databases enter the marketplace. Sometimes the correct solution is to replace the database server with a more powerful machine with more RAM. Many of the newer clustering databases make it possible to solve problems of heavy load by adding more servers. The interaction with the database is one of the greatest sources of delay. Nor does it mean that everything could be improved by throwing more money at the database layer. The best practice is to create a pool of connections that remain open with the application using a library such as Apache Commons, Apache Excalibur, or the ones from Oracle. When a thread needs to store data, it grabs an open Connection object, sends along the data, receives confirmation, then returns the Connection object to the pool where it can be reused. This process reduces the overhead spent on constant authentication and initialization. Setting up efficient connections with the database is surprisingly complex and often the source of many bottlenecks. It often makes sense to use some of the built-in persistence models. There are several included in the standard versions of Java and J2EE and a number of good open source and commercial packages. These offer highly optimized mechanisms for interacting with databases and you can benefit by relying upon their testing. Use good libraries for connecting with databases efficiently. Study the network architecture Many programmers often overlook the network hardware and topology. But in many cases, a slow network or a bad topology can do more to slow down an application than bad code. All the servers under your control might appear to have low loads, but the entire application could still be slowed. Reese spells out the best practice for design: Make sure the network can handle all traffic between components. Be willing to compromise on demands Everyone wants their data to be stored perfectly but that can be expensive when the data model is elaborate. If the database is going to juggle multiple tables and keep the data in them consistent, it must work much harder to ensure that the results are accurate even after a power failure or other catastrophe. Sometimes the best practice requires relaxing the expectations for storage in order to speed up response times. These decisions are as political as they are technical. Will the users be willing to accept very occasional errors in return for faster performance at a cheaper price? Look for opportunities to use less perfect transaction models to add speed. Many developers speak of splitting their application into microservices, which are smaller programs tuned to answer one particular question.

Chapter 6 : Java performance engineering best practices

The best way we learn anything is by practice and exercise questions. Here you have the opportunity to practice the Java programming language concepts by solving the exercises starting from basic to more complex exercises.

JavaDoc Google Java Coding Standards â€” Source File Basics The source file basics include standards that determine how to name a file, basic file encoding, Ascii character rules and whitespace rules. The most important style rules in terms of source file basics state that source files must be named using the. The name itself must contain the name of the top level class it contains. Source file names are also case sensitive and that should be kept in mind when naming source files. The only whitespace character that should appear anywhere in the file should be the Ascii horizontal space character. This means that whitespaces in characters in strings and literals are all escaped, and that tab characters should not be used for indentation purposes. Java Programming for Beginners course now. This course will teach you how to set up the Java development environment and how to troubleshoot any Java errors you may have. It will teach you how to work with the various variables within Java. The course will show you how Java strings work and how to use them in your applications. You will learn to work with simple and complex Java conditionals and the course will show you how to use the switch case break statement, how to use the while and do while loop and how to create the Java for loop. You will work with arrays and create simple and complex Java functions. The course will introduce you to the concept of Java inheritance and it will show you how to implement inheritance in your applications. Google Java Coding Standards â€” Source File Structures According to the Google Java coding standard, each source file may consist of license or copyright information, a package statement, import statements and then exactly one top level class, in that particular order. Developers will use one blank line only to separate each section within the file. This section also specifies that no wildcard imports are allowed. Import statements should not be line wrapped and the column limit of 80 or does not apply to these statements. This section also sets out the standards for grouping and sorting import statements. This course offers lectures and 10 hours of video content that will teach you all the Java skills required to program effectively in Java. It will teach you about programming using object-oriented concepts like classes, objects, loops, and more. The course is suitable for beginners and is aimed at those who thrive on learning by video and example. The training methods help you to learn by doing, therefor increasing retention and accelerating the learning process. Each step is broken down into simple to follow steps that will teach you to develop a Java program from start to finish. Google Java Coding Standards â€” Formatting in Java In terms of the google Java coding standards, programmers need to add braces where braces are optional. Braces should therefore be used with all if, else, for, while and do statements even when these structures contain only one statement. The brace should appear on the same line as the statement, so no line breaks should appear between the statement and the bracket. There should be a line break after the bracket and after the closing bracket. Block indentation should be set at two spaces. Statements should be confined to a new line per statement. Columns widths should be confined to 80 to characters except for long URL or import statements. Vertical line breaks or whitespaces should be limited to members of a class, to create logical groupings of statements. Multiple line breaks are discouraged. Horizontal whitespaces should be used to separate reserved words, separate reserved words from braces, before curly braces, on both sides of a binary operator, after closing parenthesis, between a type and variable declaration. Interior whitespaces should be avoided. Google Java Standards There are a number of different Java coding standards. The above is an introduction to those standards. It is important to apply the standards to your programs if you want to run them via Google in any form. These standards are not only necessary for programs running on Google, but they actually form part of best programming practices for Java in general, so learning programming standards will ultimately make you a better programmer. This course will teach you the basic Java concepts you need to understand to build fully functional Java programs. You will learn how Java is the base for all object-oriented programming languages and how you can improve your programming skills in any other language by learning Java. The course will teach you the basics of Java syntax. You will learn the complexities of object-oriented programming and what object-orientation means. The course will teach you about Java SE and it will take you step by step through the process of creating a Java program with user interfaces that run on Mac, PC and Linux machines. You will learn about the different variables in Java and how you can use them in your programs. Each chapter of the course ends with specially designed exercises to ensure that you can harness the power of what you have learned to create Java programs that you can be proud of.

Chapter 7 : Unit Writing good Java code

Java Best Practices. I've been Programming, Designing and Architecting Java applications for 15 years. I would like this page to serve as a good starting point for programmers to understand what it takes to build good applications.

Do not make the member variables public or protected. The event handler should not contain the code to perform the required action. Rather call another method from the event handler. Do not programmatically click a button to execute the same action you have written in the button click event. Rather, call the same method which is called by the button click event handler. Never hardcode a path or drive name in code. Get the application path programmatically and use relative path. If a wrong value found in the configuration file, application should throw an error or give a message and also should tell the user what are the correct values. Error messages should help the user to solve the problem if it is of an expected origin. Please make sure the login id and password are correct. When displaying error messages, in addition to telling what is wrong, the message should also tell what the user should do to solve the problem. Show short and friendly message to the user. But log the actual error with all possible information. This will help a lot in diagnosing problems. Whenever possible, do not have more than one class in a single file. Avoid having very large files. If a single file has more than lines of code, it is a good candidate for refactoring. Split them logically into two or more classes. Avoid public methods and properties, unless they really need to be accessed from outside the class. Avoid passing too many parameters to a method. If you have a method returning a collection, return an empty collection instead of null, if you have no data to return. For example, if you have a method returning an ArrayList, always return a valid ArrayList. If you have no items to return, then return a valid ArrayList with 0 items. Logically organize all your files within appropriate folders. Use 2 level folder hierarchies. You can have up to 10 folders in the root folder and each folder can have up to 5 sub folders. If you have too many folders than cannot be accommodated with the above mentioned 2 level hierarchy, you may need re factoring into multiple packagesd. A good logging class needs to be create, which can be configured to log errors, warning or traces. If you configure to log errors, it should only log errors. But if you configure to log traces, it should record all errors, warnings and trace. The log class should be written such a way that in future you can change it easily to log to Windows Event Log, SQL Server, or Email to administrator or to a File etc without any change in any other part of the application. Use the log class extensively throughout the code to record errors, warning and even trace messages that can help you trouble shoot a problem. Declare variables as close as possible to where it is first used. Use one variable declaration per line. Do not declare all variables at the top of the function. Use StringBuilder class instead of String when you have to manipulate string objects in a loop. Consider the following example:

Chapter 8 : Java Practices->Home

Java Practices www.nxgvision.com offers concise presentations of Java practices, tasks, and designs, illustrated with syntax-highlighted code examples. Some general-purpose references are provided, along with some source code.

The popularity of Java programming language can be predicted from the scope of its use. While writing code, the developers work hard to write bug-free code with the least complexity and the most functionality. Whether you are a beginner source code writer or an expert, there are some rules that you should always follow to get the best code for your client. Below are the rules for bug-free code writing. No Need to Depend on Initialization In Java, developers always depend on the use of constructors to initialize an object. But this is a big myth that most developers follow. There are numerous ways to allocate an object without calling a constructor. To do this, you can follow any of the following steps: You can declare all the variables as private. For each object, write a new private boolean variable and initialize it. Write a non-constructor class that will ensure that each object is initialized before calling anywhere in the code. Secure Your Classes, Method, And Variables In your code, you will make some classes, methods, and variables as private and some as public. But the public methods, variables can be easily accessed and become a point to be attacked. So, try to make them with limited scope. Always remember to make the classes, methods, and variables public when you have the only option to do so. Always Predefine the Scope Most developers totally depend on the scope of the packages, but you should always predefine the scope of your code. JVM is not closed by default, allowing you to close your classes within the package. The inner classes are usually accessible to all the other classes in the same package, and it is already mentioned that you should predefine the scope of each and every class you are creating in the code. Ensure Noncloneable Classes Java has a feature to clone its own classes whenever required. But this feature can also be used adversely by the hackers. A hacker can easily use the java. Cloneable to make duplicate instances of the code and steal necessary information from your code. To get rid of this issue, all you have to do is to add the following code in each and every class of your code. But by keeping mindful of the concepts of java programming and your own experience, you can easily code without encountering any bug.

Chapter 9 : SOLID Principles in Java [with Examples] - HowToDoInJava

The best way to avoid this problem is to avoid the use of Java synchronization. One of the most common uses of synchronization is to implement pooling of serially reusable objects. Often, you can simply add a serially reusable object to an existing pooled object.

Beginners find it hard to understand and even experienced developers can spend hours discussing how and which Java exceptions should be thrown or handled. Nevertheless, there are several best practices that are used by most teams. Here are the 9 most important ones that help you get started or improve your exception handling. A common mistake in these situations is to close the resource at the end of the try block. All statements within the try block will get executed, and the resource gets closed. But you added the try block for a reason. You call one or more methods which might throw an exception, or maybe you throw the exception yourself. That means you might not reach the end of the try block. And as a result, you will not close the resources. You should, therefore, put all your clean up code into the finally block or use a try-with-resource statement. Use a Finally Block In contrast to the last few lines of your try block, the finally block gets always executed. That happens either after the successful execution of the try block or after you handled an exception in a catch block. Due to this, you can be sure that you clean up all the opened resources. You can use it if your resource implements the AutoCloseable interface. When you open the resource in the try clause, it will get automatically closed after the try block got executed, or an exception handled. Prefer Specific Exceptions The more specific the exception is that you throw, the better. Therefore make sure to provide them as many information as possible. That makes your API easier to understand. And as a result, the caller of your method will be able to handle the exception better or avoid it with an additional check. So, always try to find the class that fits best to your exceptional event, e. And avoid throwing an unspecific Exception. Document the Exceptions You Specify Whenever you specify an exception in your method signature, you should also document it in your Javadoc. That has the same goal as the previous best practice: Provide the caller as many information as possible so that he can avoid or handle the exception. So, make sure to add a throws declaration to your Javadoc and to describe the situations that can cause the exception. Throw Exceptions With Descriptive Messages The idea behind this best practice is similar to the two previous ones. It should, therefore, describe the problem as precisely as possible and provide the most relevant information to understand the exceptional event. But you should explain the reason for the exception in short sentences. That helps your operations team to understand the severity of the problem, and it also makes it easier for you to analyze any service incidents. If you throw a specific exception, its class name will most likely already describe the kind of error. A good example for that is the NumberFormatException. It gets thrown by the constructor of the class java. Long when you provide a String in a wrong format. Its message only needs to provide the input string that caused the problem. Catch the most specific exception first Most IDEs help you with this best practice. They report an unreachable code block when you try to catch the less specific exception first. The problem is that only the first catch block that matches the exception gets executed. Always catch the most specific exception class first and add the less specific catch blocks to the end of your list. You can see an example of such a try-catch statement in the following code snippet. You can use it in a catch clause, but you should never do it! If you use Throwable in a catch clause, it will not only catch all exceptions; it will also catch all errors. Errors are thrown by the JVM to indicate serious problems that are not intended to be handled by an application. So, please, never ignore an exception. Someone might remove the validation that prevented the exceptional event without recognizing that this creates a problem. You should at least write a log message telling everyone that the unthinkable just had happened and that someone needs to check it. You can find lots of code snippets and even libraries in which an exception gets caught, logged and rethrown. But it will write multiple error messages for the same exception. As explained in best practice 4, the exception message should describe the exceptional event. And the stack trace tells you in which class, method, and line the exception was thrown. If you need to add additional information, you should catch the exception and wrap it in a custom one. But make sure to follow best practice number 9. Otherwise, specify it in the method signature and let the caller take care of it. A typical example for such an exception is an application or framework specific business exception. That allows you to add additional information and you can also implement a special handling for your exception class. When you do that, make sure to set the original exception as the cause. The Exception class provides specific constructor methods that accept a Throwable as a parameter. Otherwise, you lose the stack trace and message of the original exception which will make it difficult to analyze the exceptional event that caused your exception. Most of them have the goal to improve the readability of your code or the usability of your API. Exceptions are most often an error handling mechanism and a communication medium at the same time. You should, therefore, make sure to discuss the Java exception handling best practices and rules you want to apply with your coworkers so that everyone understands the general concepts and uses them in the same way. When using Retrace APM with code profiling, you can collect exceptions directly from Java, without any code changes!