DOWNLOAD PDF STANDARD PATTERNS FROM GOF BOOK

Chapter 1 : design patterns - Is the GoF book still the one to read? - Software Engineering Stack Exchange

Design Patterns: Elements of Reusable Object-Oriented Software () is a software engineering book describing software design patterns. The book's authors are Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides with a foreword by Grady Booch.

They were later joined by Ralph Johnson and John Vlissides. The authors refer to inheritance as white-box reuse, with white-box referring to visibility, because the internals of parent classes are often visible to subclasses. In contrast, the authors refer to object composition in which objects with well-defined interfaces are used dynamically at runtime by objects obtaining references to other objects as black-box reuse because no internal details of composed objects need be visible in the code using them. The authors discuss the tension between inheritance and encapsulation at length and state that in their experience, designers overuse inheritance Gang of Four The danger is stated as follows: Gang of Four Furthermore, they claim that a way to avoid this is to inherit only from abstract classesâ€”but then, they point out that there is minimal code reuse. Using inheritance is recommended mainly when adding to the functionality of existing components, reusing most of the old code and adding relatively small amounts of new code. Delegation involves two objects: Thus the link between two parts of a system are established only at runtime, not at compile-time. The Callback article has more information about delegation. The authors admit that delegation and parameterization are very powerful but add a warning: Acquaintance is a weaker relationship than aggregation and suggests much looser coupling between objects, which can often be desirable for maximum maintainability in a design. They state that applications are hard to design, toolkits are harder, and frameworks are the hardest to design. Each problem is analyzed in depth, and solutions are proposed. Each solution is explained in full, including pseudo-code and a slightly modified version of Object Modeling Technique where appropriate. Finally, each solution is associated directly with one or more design patterns. It is shown how the solution is a direct implementation of that design pattern. The seven problems including their constraints and their solutions including the pattern s referenced , are as follows: Document Structure[edit] The document is "an arrangement of basic graphical elements" such as characters, lines, other shapes, etc. The structure of the document contains a collection of these elements, and each element can in turn be a substructure of other elements. It should not have to know the difference between the two. Specific derivatives of abstract elements should have specialized analytical elements. Solution and Pattern A recursive composition is a hierarchical structure of elements, that builds "increasingly complex elements out of simpler ones" pp Each node in the structure knows of its own children and its parent. If an operation is to be performed on the whole structure, each node calls the operation on its children recursively. This is an implementation of the composite pattern , which is a collection of nodes. The node is an abstract base class , and derivatives can either be leaves singular , or collections of other nodes which in turn can contain leaves or collection-nodes. When an operation is performed on the parent, that operation is recursively passed down the hierarchy. Formatting[edit] Formatting differs from structure. This includes breaking text into lines, using hyphens, adjusting for margin widths, etc. Problems and Constraints Balance between formatting quality, speed and storage space Keep formatting independent uncoupled from the document structure. Solution and Pattern A Compositor class will encapsulate the algorithm used to format a composition. A Compositor has an associated instance of a Composition object. When a Compositor runs its Compose , it iterates through each element of its associated Composition, and rearranges the structure by inserting Row and Column objects as needed. The Compositor itself is an abstract class, allowing for derivative classes to use different formatting algorithms such as double-spacing, wider margins, etc. The Strategy Pattern is used to accomplish this goal. A Strategy is a method of encapsulating multiple algorithms to be used based on a changing context. In this case, formatting should be different, depending on whether text, graphics, simple elements, etc. Embellishing the User Interface[edit] The ability to change the graphical interface that the user uses to interact with the document. Problems and Constraints Demarcate a page of text with a border around the editing area Scroll bars that let the user view different parts of the page User interface objects should not know about the embellishments

Avoid an "explosion of classes" that would be caused by subclassing for "every possible combination of embellishments" and elements p44 Solution and Pattern The use of a transparent enclosure allows elements that augment the behaviour of composition to be added to a composition. These elements, such as Border and Scroller, are special subclasses of the singular element itself. This allows the composition to be augmented, effectively adding state-like elements. This means that the client does not need any special knowledge or interface with the structure in order to use the embellishments. This is a Decorator pattern , one that adds responsibilities to an object without modifying the object itself. These standards "define guidelines for how applications appear and react to the user" pp Problems and Constraints The editor must implement standards of multiple platforms so that it is portable Easily adapt to new and emergent standards Allow for run-time changing of look-and-feel i. No hard-coding Have a set of abstract elemental subclasses for each category of elements ScrollBar, Buttons, etc. Have a set of concrete subclasses for each abstract subclass that can have a different look-and-feel standard. ScrollBar having MotifScrollBar and PresentationScrollBar for Motif and Presentation look-and-feels Solution and Pattern Since object creation of different concrete objects cannot be done at runtime, the object creation process must be abstracted. This is done with an abstract guiFactory, which takes on the responsibility of creating UI elements. The abstract guiFactory has concrete implementations, such as MotifFactory, which creates concrete elements of the appropriate type MotifScrollBar. In this way, the program need only ask for a ScrollBar and, at run-time, it will be given the correct concrete element. This is an Abstract Factory. A regular factory creates concrete objects of one type. An abstract factory creates concrete objects of varying types, depending on the concrete implementation of the factory itself. Its ability to focus on not just concrete objects, but entire families of concrete objects "distinguishes it from other creational patterns, which involve only one kind of product object" pp Supporting Multiple Window Systems[edit] Just as look-and-feel is different across platforms, so is the method of handling windows. Each platform displays, lays out, handles input to and output from, and layers windows differently. Problems and Constraints The document editor must run on many of the "important and largely incompatible window systems" that exist p. Due to differing standards, there will not be a common abstract class for each type of widget. Do not create a new, nonstandard windowing system Solution and Pattern It is possible to develop "our own abstract and concrete product classes", because "all window systems do generally the same thing" p. An abstract base Window class can be derived to the different types of existing windows, such as application, iconified , dialog. These classes will contain operations that are associated with windows, such as reshaping, graphically refreshing, etc. In order to avoid having to create platform-specific Window subclasses for every possible platform, an interface will be used. The Window class will implement a Window implementation WindowImp abstract class. This class will then in turn be derived into multiple platform-specific implementations, each with platform-specific operations. Hence, only one set of Window classes are needed for each type of Window, and only one set of WindowImp classes are needed for each platform rather than the Cartesian product of all available types and platforms. In addition, adding a new window type does not require any modification of platform implementation, or vice versa. This is a Bridge pattern. Window and WindowImp are different, but related. Window deals with windowing in the program, and WindowImp deals with windowing on a platform. One of them can change without ever having to modify the other. The Bridge pattern allows these two "separate class hierarchies to work together even as they evolve independently" p. User Operations[edit] All actions the user can take with the document, ranging from entering text, changing formatting, quitting, saving, etc. Problems and Constraints Operations must be accessed through different inputs, such as a menu option and a keyboard shortcut for the same command Each option has an interface, which should be modifiable Operations are implemented in several different classes In order to avoid coupling, there must not be a lot of dependencies between implementation and user interface classes. Menus should be treated like hierarchical composite structures. Hence, a menu is a menu item that contains menu items which may contain other menu items, etc. Solution and Pattern Each menu item, rather than being instantiated with a list of parameters, is instead done with a Command object. Command is an abstract object that only has a single abstract Execute method. Derivative objects extend the Execute method appropriately i. These objects can be used by widgets or buttons just as easily as they can be used by menu

items. To support undo and redo, Command is also given Unexecute and Reversible. In derivative classes, the former contains code that will undo that command, and the latter returns a boolean value that defines if the command is undoable. Reversible allows some commands to be non-undoable, such as a Save command. All executed Commands are kept in a list with a method of keeping a "present" marker directly after the most recently executed command. A request to undo will call the Command. Unexecute directly before "present", then move "present" back one command. Conversely, a Redo request will call Command. Execute after "present", and move "present" forward one. This Command approach is an implementation of the Command pattern. It encapsulates requests in objects, and uses a common interface to access those requests. Thus, the client can handle different requests, and commands can be scattered throughout the application. Although there are many analyses that can be performed, spell check and hyphenation-formatting are the focus. Problems and Constraints Allow for multiple ways to check spelling and identify places for hyphenation Allow for expansion for future analysis e. Solution and Pattern Removing the integer-based index from the basic element allows for a different iteration interface to be implemented. This will require extra methods for traversal and object retrieval. These methods are put into an abstract Iterator interface. Each element then implements a derivation of the Iterator, depending on how that element keeps its list ArrayIterator, LinkListIterator, etc.

Chapter 2 : Software design pattern - Wikipedia

As per the design pattern reference book Design Patterns - Elements of Reusable Object-Oriented Software, there are 23 design patterns which can be classified in three categories: Creational, Structural and Behavioral patterns. We'll also discuss another category of design pattern: J2EE design patterns.

British pastor and teacher John R. Do not follow their practices. You must obey my laws and be careful to follow my decrees. Also see Leviticus Also see 2 Corinthians 6: Jesus did not want his followers to be like the heathen, or to be like the religious leaders of his time. See, for example, Matthew 6: Also see verse The disciples were to be different. The rules of the kingdom of God are often the exact opposite of the rules of this world. But as we mature as Christians, we can come fairly close to living by them. Why is it so important for Christians to be different? It is the fact that we are different that challenges and draws unbelievers to God. If committed Christians are perceived as no different from anyone else, then what does Christianity have to offer to unbelievers? It is only as we are seen as having something different to offer, that people will be drawn to that something different and to God. If we continue to be conformed to the pattern of this world, then our allegiance is to this world. Or else we are double-minded, with one foot in the world and one foot in the kingdom of God. In either case, we will not have the desire or the power to stand up against the ungodliness that is so prevalent in our society today. Most Christians spend relatively few hours a week on the things of God. The rest of the time we are bombardedâ€”systematically, pervasively and insistentlyâ€”by the things of this world. If we are to stand up against that bombardment, we need to make a deliberate effort to be radically different from the world. As in most of his letters, he starts with a discussion of basic spiritual principles, and then discusses their practical application in our lives. In Romans that discussion of application begins with chapter First he tells us to give our lives to God, to commit ourselves totally to God Romans I deal with this in chapter I believe this is a key verse in Scripture. In this chapter, I deal with the first half of this sentence. In the next chapter, I shall deal with the second half. The two are closely related. They are opposite sides of the same coin. As we become transformed, we will reject the pattern of this world. But we cannot have it both ways. Rejecting the pattern of this world does not mean that we should live in isolation, as hermits. We are called to be salt and light to the world Matthew 5: Jesus moved about actively in the world of his day. Scripture repeats this theme over and over, in many different ways. He contrasts the wisdom of this world, or of this age, with the eternal wisdom of God, and warns against following the former 1 Corinthians 1: Also see James 3: Scripture warns us against it, also. Jesus spoke of satan as the ruler of this world John Also see John James tells us that pure religion is to keep oneself from being polluted by the world James 1: Also see James 4: Peter warns us to escape the corruption that is in the world 2 Peter 1: John warns us not to love the world or anything in it 1 John 2: He speaks of believers as overcoming the world 1 John 5: Scripture calls on us to be holy 1 Peter 1: Today the pattern of this world is thrust upon us to a greater degree than ever before. Most of these sounds and images are worldly. Some encourage sexual immorality. Others contain excessive and graphic violence. There is profanity and crude language. The family particularly fathers is portrayed in an unfavorable light. Christian values are mocked. Lying, cynicism, hardheartedness, and sexual promiscuity are portrayed as being normal. Selfishness and greed are encouraged. The world is not neutral. It is making a deliberate assault on Christian values and standards. It takes determination, and solid roots, to stand up against this assault. Why is the world such a danger? Let me mention some, among many, reasons: It contains much corruption and tempts us to become corrupt. Our desire for material things can become a form of idolatry. We are dealing here with two different world-views, a secular or worldly one, and a godly one. Our world views are pervasive and deeply entrenched. We do not change them easily or quickly. We are also dealing with mental strongholds. See 2 Corinthians There is spiritual conflict going on within us. The devil resists strenuously our efforts to demolish the stronghold of worldliness. Often strongholds do not come down easily or quickly. It takes persistent effort. One of the greatest temptations we face, as Christians, is the desire to please the world rather than God. Our goal should be to please God 2 Corinthians 5: Also see Ephesians 5: But the temptation is always to seek to please men, or to avoid their displeasure. This statement may seem

shocking, but I think we need to take it seriously. The social pressure to conform to the world and its standards is great, but we must resist that pressure. We have ignored Romans Believe in the Bible The first essential is that we must really believe in the Bible. We must consider it as authoritative and take it seriously. We must take Romans Then we must check everything we say or do against what the Bible says. Many of these are people who do not accept the Bible, the entire Bible, as authoritative. If we stand on the truth of Scripture, we have a solid basis from which to resist worldly pressures. If we do not stand on the truth of Scripture, then there is little reason not to go along with every popular movement and teaching. This may take courage. It is hard to resist peer pressure. When we resist, it may lead to confrontation, to mocking and ridicule, to rejection and other unpleasant consequences. The world hated Jesus and his disciples because they were not of this world. If we would follow godly standards, we need to be willing to accept disapproval and even hatred. Whom do we want to please, people or God? There are times when you cannot do both. I would carry this further and say that we need to take our thoughts, our words, and our actions captive to obey Jesus Christ. This takes continual alertness. We need to notice what we are thinking, saying and doing, and compare it with Scriptural teaching. We need to catch ourselves and repent. This is not easy, but it becomes easier with time and practice. It is when we are not consciously thinking of spiritual things that we are most apt to find ourselves conforming to the pattern of this world.

Chapter 3 : Do All Things According to the Pattern

The Gang of Four are the four authors of the book, "Design Patterns: Elements of Reusable Object-Oriented Software". In this article their twenty-three design patterns are described with links to UML diagrams, source code and real-world examples for each.

SingletonPattern] [PatternSingleton. StrategyPattern] [PatternStrategy. Also note that an efficient State implementation can use Singleton for the states. StatePattern] on the WikiWikiWeb. TemplateMethodPattern] This result is also called a Framework. Diferent types of element can use different operations in the interface if necesary. When in doubt leave it out! Keep It Simple Stupid! Only analyse, design, implement, and test a function that your client actually needs now. Notice that this can get clumsy and in Java foul up more important extensions. A marriage has two people: Treats SQL as an external entity with its own secrets. Have a single object that is responsible for recording bad things. Have a standard user interface for reporting and sorting out errors. So, for example a Controller will find the object that fits the ID. The are several places the controller can look to find id including an internal data structure listing all the possible objects. But using a data structure in the Controler means that the controller becomes responsible for updating this list -- which means lower cohesion. The Controler can not send the find ID to the object because the object has not been found yet! Of course, the Creator of an object has the information to store IDs and their objects. So a simple solution is to use the metaclass where the Constructor is. The diagram below shows the Controller using a static function to ask the Class that created the objects which one matches the id. The returned object is then sent on to other objects to get the work done: If the exact class of the object is not known, the Controller cannot turn to that class to find the id. For example, when I login to MyCoyote it does not know whether I am a student or a faculty member, until it looks up my ID. In this case we might well have decided to use a Factory object to construct the object we need. The top level one can keep track of all the IDs and the specific objects that are needed. Then we use Polymorphism: For example, it can return a Faculty object or a Student object as a special kind of Person class, by abstraction. Also known as Aim high -- allow your pointers to refer to abstractions and then they can point to many special kinds of object. Declare variable abstract and assign address of concrete: Polymorphism lets you add new functions to a class by putting them in a subclass. This can be separately compiled and linked into the running system without having to compile the base class. Put input, output, and logic in separate modules that communicate by function calls. Separate the physical from the logical. Hide external systems behind a Facade. Hide communications between parts inside a Mediator. Hide libraries behind a Facade , an Adapter , or a Bridge. Separate particular Applications from Business Logic and Domain classes. Separate Model, View, and Controller. This dates back to the the theory of abstract data types and was inherited by Meyer into objected oriented coding style. This principle leads to code that is slightly more verbose but very much clearer. Henry Ledgard proposed this for programming languages but it also applies to user interfaces, common appliances, and object-oriented classes. Apple had a tendency to extend so that the user experiences occasional pleasant surprises: This is also called DWIM:: This is a dangerous philosophy because it back fires. And use special icons for them: The code that describes user interfaces is independent of the code for the logic of an enterprise or application. The appearance of the user interface for a pay-roll should be separated from the rules for calculating your with-holdings. Neither should the choice of the user interface framework change the calculations. So the wise designer splits all the user interface code into specialized classes that communicate with the application specific classes. This is splitting the users "View" from the logical "Model". As a result the "Model" classes can reflect the Domain or Business model and so are easier to understand and get right Here is a paper [One program has a command line interface, one has a menu driven ASCII interface and one a GUI interface, but they share a common "model" of how to access the time server. This class never changes The diagrams in the paper are not correct UML, do not use them! In general, there are lots of classes defining the user interface and many classes in the Model. In the UML use packages to contain them. More strictly the user interface code is split into two sets of classes. The View and the Controler. Typically the user communicates with controlers that change the model. The

view classes get data from the model and are responsible for rendering it to the user. Net Purposes include functions and use cases and define why someone wants the software. To help us sell items to customers. Qualities are the needed properties of the software like security and performance criteria. Realities are domain models. We have the problem of integrating different views of the the real world Systems are the existing systems that we are replacing, using, modifying, and even competing with. Techie include technologies, techniques, etc. Instead of the program calling the operating system when it needs to something, it waits to be called by the environment when events happen. The classic GoF State pattern uses it to change the state. You should read and study this an excellent, short, and readable blog entry [Good Links on Patterns A good list, overview and, discussion of patterns can be found at [wiki? CategoryPattern] on the WikiWikiWeb. Another source of pattern information is [Patterns. Here is a nice resource [http: Paul Wagner [ ] Has listed a dozen commain model patterns:

Chapter 4 : Gang of Four Design Patterns - Spring Framework Guru

The Gang of Four book - Design Patterns: Elements of Reusable Object-Oriented Software is probably the closest thing we have to an industry standard on design patterns. For a more accessible introduction, Head First: Design Patterns is good too.

This web site uses cookies. By using the site you accept the cookie policy. Elements of Reusable Object-Oriented Software". In this article their twenty-three design patterns are described with links to UML diagrams, source code and real-world examples for each. What are Design Patterns? Design patterns provide solutions to common software design problems. In the case of object-oriented programming , design patterns are generally aimed at solving the problems of object generation and interaction, rather than the larger scale problems of overall software architecture. They give generalised solutions in the form of templates that may be applied to real-world problems. Design patterns are a powerful tool for software developers. However, they should not be seen as prescriptive specifications for software. It is more important to understand the concepts that design patterns describe, rather than memorising their exact classes , methods and properties. It is also important to apply patterns appropriately. Using the incorrect pattern for a situation or applying a design pattern to a trivial solution can overcomplicate your code and lead to maintainability issues. Who are the Gang of Four? The Gang of Four are the authors of the book, " Design Patterns: Elements of Reusable Object-Oriented Software ". This important book describes various development techniques and pitfalls in addition to providing twenty-three object-oriented programming design patterns. Gang of Four Design Patterns This section gives a high-level description of the twenty-three design patterns described by the Gang of Four. Each pattern description includes a link to a more detailed article describing the design pattern and including a UML diagram, template source code and a real-world example programmed using C. Creational Patterns The first type of design pattern is the creational pattern. Creational patterns provide ways to instantiate single objects or groups of related objects. There are five such patterns: The abstract factory pattern is used to provide a client with a set of related or dependant objects. The "family" of objects created by the factory are determined at run-time. The builder pattern is used to create complex objects with constituent parts that must be created in the same order or using a specific algorithm. An external class controls the construction algorithm. The factory pattern is used to replace class constructors , abstracting the process of object generation so that the type of the object instantiated can be determined at run-time. The prototype pattern is used to instantiate a new object by copying all of the properties of an existing object, creating an independent clone. This practise is particularly useful when the construction of a new object is inefficient. The singleton pattern ensures that only one object of a particular class is ever created. All further references to objects of the singleton class refer to the same underlying instance. Structural Patterns The second type of design pattern is the structural pattern. Structural patterns provide a manner to define relationships between classes or objects. The adapter pattern is used to provide a link between two otherwise incompatible types by wrapping the "adaptee" with a class that supports the interface required by the client. The bridge pattern is used to separate the abstract elements of a class from the implementation details, providing the means to replace the implementation details without modifying the abstraction. The composite pattern is used to create hierarchical, recursive tree structures of related objects where any element of the structure may be accessed and utilised in a standard manner. The decorator pattern is used to extend or alter the functionality of objects at run-time by wrapping them in an object of a decorator class. This provides a flexible alternative to using inheritance to modify behaviour. The facade pattern is used to define a simplified interface to a more complex subsystem. The flyweight pattern is used to reduce the memory and resource usage for complex models containing many hundreds, thousands or hundreds of thousands of similar objects. The proxy pattern is used to provide a surrogate or placeholder object, which references an underlying object. The proxy provides the same public interface as the underlying subject class, adding a level of indirection by accepting requests from a client object and passing these to the real subject object as necessary. Behavioural Patterns The final type of design pattern is the behavioural pattern. Behavioural patterns define manners of communication between classes and

objects. The chain of responsibility pattern is used to process varied requests, each of which may be dealt with by a different handler. The command pattern is used to express a request, including the call to be made and all of its required parameters , in a command object. The command may then be executed immediately or held for later use. The interpreter pattern is used to define the grammar for instructions that form part of a language or notation, whilst allowing the grammar to be easily extended. The iterator pattern is used to provide a standard interface for traversing a collection of items in an aggregate object without the need to understand its underlying structure. The mediator pattern is used to reduce coupling between classes that communicate with each other. Instead of classes communicating directly, and thus requiring knowledge of their implementation, the classes send messages via a mediator object. The memento pattern is used to capture the current state of an object and store it in such a manner that it can be restored at a later time without breaking the rules of encapsulation. The observer pattern is used to allow an object to publish changes to its state. Other objects subscribe to be immediately notified of any changes. The state pattern is used to alter the behaviour of an object as its internal state changes. The pattern allows the class for an object to apparently change at run-time. The strategy pattern is used to create an interchangeable family of algorithms from which the required process is chosen at run-time. The template method pattern is used to define the basic steps of an algorithm and allow the implementation of the individual steps to be changed. The visitor pattern is used to separate a relatively complex set of structured data classes from the functionality that may be performed upon the data that they hold. For a quick reference to the design patterns featured in this article, see the Gang of Four Design Patterns Reference Sheet.

Chapter 5 : CSE Patterns and Principles

Links on the Gang of Four patterns. THe Wikipedia has articles of varying quality on the GoF patterns. Here [Design_Patterns_(book)] is a link to a description of the book.

Yet, the questions we need to ask are, "How does God judge us? By our hearts, or by our deeds? If one is expecting the answer to be one or the other, then the answer given by the Bible may be surprising. The Pattern The title of this article refers to a quote found in three different places in the Bible. First, when God instructed Moses how to build the tabernacle in the Old Testament, He told Moses to "make all things according to the pattern" Exodus This reference is repeated in the book of Hebrews as a reminder of doing all things today "according to the pattern" Hebrews 8: A similar statement is made in Ezekiel, during the Jews captivity in Babylon. In the middle of a 9-chapter description of the dimensions of the new holy city, God tells Ezekiel: No more shall the house of Israel defile My holy name. Son of man, describe the temple to the house of Israel, that they may be ashamed of their iniquities; and let them measure the pattern. And if they are ashamed of all that they have done, make known to them the design of the temple and its arrangements, its exits and its entrances, its entire design and all its ordinances, all its forms, and all its laws. Write it down in their sight, so that they may keep its whole design and all its ordinances, and perform them. This is the law of the temple: The whole area surrounding the mountaintop is most holy. Behold, this is the law of the temple. This section became ultimately fulfilled in the New Testament church, and it continues to apply to us today. Yet, the solution is the same today as it was then: But, the Bible both condemns and condones these attitudes. It depends on what is implied by the charges. First, the Bible clearly condemns, or teaches that going through the motions is not what God desires. Please consider the following passages from the Old Testament: Shall I come before Him with burnt offerings, with calves a year old? Will the Lord be pleased with thousands of rams, ten thousand rivers of oil? Shall I give my firstborn for my transgression, the fruit of my body for the sin of my soul? The sacrifices of God are a broken spirit, and a broken and contrite heart - These, O God, You will not despise" Psalm As we look at more scriptures, we will learn that it is not the sacrifices or rituals alone that God desired. They were worthless if the heart is not right, but these statements do not necessitate that we conclude that God does not care about the correctness of our actions. These passages do not teach that God does not care about actions, but it does teach that actions alone are not enough. Love is emphasized by everyone today. It is one of the most powerful themes of the Bible. In the New Testament , Paul expressed the importance of love in the following passage: And though I have the gift of prophecy, and understand all mysteries and all knowledge, and though I have all faith, so that I could remove mountains, but have not love, I am nothing. And though I bestow all my goods to feed the poor, and though I give my body to be burned, but have not love, it profits me nothing. However, please note the fundamental error in logic. The above passage emphasizes and mandates love as our motivation, but it neither deemphasizes, releases us from, nor negates the commandments of God. Just because God requires love, does not imply that He does not require obedience. It is a leap in logic, and an unfounded assumption to go from these passages to this erroneous conclusion. But, how do we know this is so? We can find an answer by studying examples of how God dealt with such cases. In fact, he told them not to add anything, or do anything that he had not authorized. Please examine the following passages from both the New and Old Testaments: As we have said before, so now I say again, if anyone preaches any other gospel to you than what you have received, let him be accursed. If anyone ministers, let him do it as with the ability which God supplies, that in all things God may be glorified through Jesus Christ, to whom belong the glory and dominion forever and ever. Amen" I Peter 4: He who abides in the doctrine of Christ has both the Father and the Son. If anyone adds to these things, God will add to him the plagues that are written in this book; and if anyone takes away from the words of the book of this prophecy, God shall take away his part from the Book of Life, from the holy city, and from the things which are written in this book. It is sin for us to give permission for practices that were forbidden God, and it is sin to forbid the practices that were permitted by God. People were accursed by God for doing this. Finally, we are instructed to speak, or teach, where the Bible has spoken. Nadab and Abihu The following examples illustrate different

facets of the above statements. The consequence of their sin was immediate death. What was this heinous crime? So fire went out from the Lord and devoured them, and they died before the Lord. What is the lesson for us? Whenever God gives us specific instructions or commands, He expects us to keep it. We see both from the direct statements above and this example, that it is sin to go "outside" the boundaries that God has authorized. God instructed Saul to "utterly destroy" the nation of Amalek. But, Saul saved the king of Amalek, and he saved animals for sacrifices to the Lord. It would seem that sacrifices to God are good thing. Behold to obey is better than sacrifice, and to heed than the fat of rams. For rebellion is as the sin of witchcraft, and stubbornness is as iniquity and idolatry. Because you have rejected the word of the Lord, He also has rejected you from being king. However, this next example raises the standard even higher. It is the example of Uzza. However, he is not doing in the manner that God prescribed: Then the anger of the Lord was aroused against Uzza, and God struck him there for his error; and he died there by the ark of God. Probably like us, he could not understand why God would kill a man trying to do a good thing. Was not Uzza trying to prevent the ark from falling and being damaged? Later, David realized why God acted as He did. Once we have, then we stand condemned before God. While God may not kill us immediately today, the condemnation of sin warrants a postponed, but much more significant death - spiritual. The New Testament Teaching Often we assume that God has changed since the Old Testament, and since the coming of Christ, God has become a god of love, rather than wrath and judgment. But, is this really what the Bible says? Please examine the following quote from Jesus: Moreover, Jesus says that there will be surprised people on Judgment Day that will be condemned as sinners. What is the sobering lesson? The end never justifies the means, and neither does ignorance or purity of motive provide excuse! An Impossible Task These passages present what seems an impossible task. How can we hope of getting to heaven without making some accidental mistake, even with constant study and prayer? This troublesome question has a Bible answer, but it demands our faith. First, we see that God is not some god of fury that enjoys destroying people, nor does He watch from heaven, waiting for people to sin, so He can kill and condemn them. But, the Bible teaches another disposition: God does not want us to be destroyed and seeks every opportunity to be saved. As part of this desire, He has given us the following promise upon which we can rest: We must be sure that we have the proper heart - honest, loving and seeking truth - so that we will recognize the opportunities. God has promised both longsuffering and providential care. Our faith calls upon us to trust Him, and with this we can rest secure and confident in Him. Conclusion It is necessary to cover these scriptures, so that we may be firmly grounded in this truth. If we are not convinced that God cares about how we respond to His will, then we will not be ready to seek His will. Now, we will focus our search on the distinction between the New and Old Testaments. This subject will greatly affect our understanding and interpretation of the Bible. Failure to properly recognize this distinction may cause us to do the very thing that this study has warned against - "adding to", "taking away", straying to "the right", or to "the left".

Chapter 6 : java - what is Gang of Four design pattern - Stack Overflow

I find the GoF design patterns book very useful, especially the first section that cautions against common pitfalls such as overuse of inheritance.

Even though the GoF Design Patterns book was published over 20 years ago, it continues to be an Amazon best seller. As a Java developer using the Spring Framework to develop enterprise class applications, you will encounter the GoF Design Patterns on a daily basis. The GoF Design Patterns are broken into three categories: Allows for the creation of objects without specifying their concrete type. Used to create complex objects. Creates objects without specifying the exact class to create. Creates a new object from an existing object. Ensures only one instance of an object is created. Structural Design Patterns Adapter. Allows for two incompatible classes to work together by wrapping an interface around one of the existing classes. Decouples an abstraction so two classes can vary independently. Takes a group of objects into a single object. Provides a simple interface to a more complex underlying object. Reduces the cost of complex object models. Behavior Design Patterns Chain of Responsibility. Delegates commands to a chain of processing objects. Creates objects which encapsulate actions and parameters. Implements a specialized language. Accesses the elements of an object sequentially without exposing its underlying representation. Allows loose coupling between classes by being the only class that has detailed knowledge of their methods. Provides the ability to restore an object to its previous state. Allows an object to alter its behavior when its internal state changes. Allows one of a family of algorithms to be selected on-the-fly at run-time. Defines the skeleton of an algorithm as an abstract class, allowing its sub-classes to provide concrete behavior. Separates an algorithm from an object structure by moving the hierarchy of methods into one object.

Chapter 7 : GoF Design Patterns Reference

WATERLOO CHERITON SCHOOL OF COMPUTER SCIENCE Gang of Four (GoF) OO Design Patterns CS / ECE May 11th, IMPORTANT NOTICE TO STUDENTS These slides are NOT to be used as a replacement for student notes.

Chapter 8 : Design Patterns - Wikipedia

It's a book of design patterns that describes simple and elegant solutions With this book, the Gang of Four have made a seminalcontribution to software.

Chapter 9 : Is there a canonical book on design patterns? - Software Engineering Stack Exchange

quick-guide book to the basic GoF1 design patterns. A book that could be created an up-to-date view of the GoF design patterns in a structured and.