**Chapter 5-6 Clean Code**

**Chapter 5 Formatting**

* You should take care that your code is nicely formatted
* You should choose a set of simple rules that govern the format of your code, and then you should consistently apply those rules

**The Purpose of Formatting**

* Code formatting is about communication, and communication is the professional developer’s first order of business.
* It is too important to ignore and it is too important to treat religiously

**The Newspaper Metaphor**

* The name should be simple but explanatory.
* The name, by itself, should be sufficient to tell us whether we are in the right module or not.

**Vertical Openness Between Concepts**

* all code is read left to right and top to bottom.
* Each line represents an expression or a clause, and each group of lines represents a complete thought.

**Vertical Density**

* Vertical Density implies close association

**Variable Declarations**

* Should be declared as close to their usage as possible
* Control variables for loops should usually be declared within the loop statement, as in this cute little function from the same source.

**Instance Variable**

* should be declared at the top of the class
* should not increase the vertical distance of these variables

**Dependent Functions**

* If one function calls another, they should be vertically close, and the caller should be above the caller if at all possible.

**Conceptual Affinity**

* The stronger that affinity, the less vertical distance there should be between them
* . Certain bits of code want to be near other bits. They have a certain conceptual affinity.

**Horizontal Formatting**

* Should be a short lines

**Identation**

* It is very important to put an indentation in your codes.

For example:

public class FitNesseServer implements SocketServer { private FitNesseContext context; public FitNesseServer(FitNesseContext context) { this.context = context; } public void serve(Socket s) { serve(s, 10000); } public void serve(Socket s, long requestTimeout) { try { FitNesseExpediter sender = new FitNesseExpediter(s, context); sender.setRequestParsingTimeLimit(requestTimeout); sender.start(); } catch(Exception e) { e.printStackTrace(); } } } ----- public class FitNesseServer implements SocketServer { private FitNesseContext context;

This code is without Indentation

public FitNesseServer(FitNesseContext context) {

this.context = context;

}

public void serve(Socket s) {

serve(s, 10000);

}

public void serve(Socket s, long requestTimeout) {

try {

FitNesseExpediter sender = new FitNesseExpediter(s, context);

sender.setRequestParsingTimeLimit(requestTimeout);

sender.start();

}

catch (Exception e) {

e.printStackTrace();

}

}

}

And this example has an indentation and it is a readable code and it is easy to understand.

**Dummy Scopes**

* The body of a *while* or *for* statement is a dummy
* dummy body is properly indented and surrounded by braces

For Example:

while (dis.read(buf, 0, readBufferSize) != -1) ;

**Team Rules**

* A team of developers should agree upon a single formatting style, and then every member of that team should use that style.
* The software to have a consistent style

**Uncle Bob’s Formatting Rules**

* Uncle Bob’s Formatting Rules is a perfect example of a personal formatting refer to page 91 – 92.

**Chapter 6 Objects and Data Structures**

**Data Abstract**

* Hiding implementation is about abstractions

**Data/Object Anti-Symmetry**

* Objects hide their data behind abstractions and expose functions that operate on that data.
* Data structure expose their data and have no meaningful functions

**The Law of Demeter**

* A module should not know about the innards of the objects it manipulates.
* This means that an object should not expose its internal structure through accessors because to do so is to expose, rather than to hide, its internal structure.
* C
* An object created by f
* An object passed as an argument to f
* An object held in an instance variable of C

**Train Wrecks**

**Hybrids**

* Hybrids make it hard to add new functions but also make it hard to add new data structures.
* Hybrid structures that are half object and half data structure.

**Hiding Structure**

* Objects are supposed to hide their internal structure, we should not be able to navigate through them.

**Data Transfer Objects**

* The quintessential form of a data structure is a class with public variables and no functions
* Very useful structures, especially when communicating with databases or parsing messages from sockets, and so on.
* The first in a series of translation stages that convert raw data in a database into objects in the application code.

**Active Record**

* They are data structures with public (or bean-accessed) variables; but they typically have navigational methods like save and find.
* Special forms of Data Transfer Objects.
* Active Records are direct translations from database tables, or other data sources.