# Clean code summary

## مقدمة صاحب الملخص :

### اخلاء مسئولية

- هذا الملخص يعبر بالضرورة عن *فهمي الشخصي* والذي قد يخطء في بعض المسائل
- هذا الملخص **لا يغني اطلاقا عن قراءة الكتاب نفسه**, خصوصا مع توافر الامثلة المطروحة فيه
- الهدف الوحيد من هذا الملخص هو تسهيل الامر لدى المبرمجين الذين قد لا يملكون الوقت الكافي لقراءة الكتاب
- الأمثلة المذكورة في هذا الملخص أغلبها هي من صنعي الشخصي لتسهيل المسائل للقارئ, وهي مستوحاة من محتوى الكتاب
- الاكواد المذكورة في الملخص قد تكون مش بت compile اصلا وفيها syntax errors بس دا مش فارق في حاجة طول ما انتا فاهم ال concept
- وهذا يعني انه يعتبر كبسولة سريعة الهضم او مكمل غذائي, *وليس وجبة مشبعة من المعلومات*
- لا يسمح بنشر او تداول هذا الملخص قبل *الصلاة على النبي* ثم **الدعاء لاهل غزة**, أسال الله ان يرفع عنهم البلاء وأن ينصرهم نصرا عزيزا مؤزرا, وأن يرزقهم الصبر والثبات وأن يربط على قلوبهم ويثبت أقدامهم ويسدد رميهم ويجعلهم اية لنصره, وأن يهلك عدوهم,
  اللهم منزل الكتاب ومجري السحاب هازم الاحزاب اهزم اليهود وانصرنا عليهم

### خذ نية صالحة تؤجر بها لتعلمك البرمجة عموما ولقراءتك هذا الملخص خصوصا (مثل اعفاف النفس عن السؤال ونفع المسلمين وتأدية فرض كفاية وبر الوالدين والأخذ بأسباب القوة...الخ)

### لا تنسونا من صالح دعائكم

### يرجى تنبيهي عند اكتشاف أخطاء علي ال email الاتي :

- alshaier@proton.me

# Clean Code

## Good Code matters

- الوحدة الوحيدة لقياس جودة الكود هي **حوقلة/دقيقة** (لا حول ولا قوة الا بالله, اي يبني الكود دا!)
- with bad code, team's productivity decreases
- And *due to that* team tries hard to increase their productivity
- Which leads to even WORSE code, because team are just focusing on "getting it done"
- When developers are on the pressure on delivering code before deadlines, code becomes sloppy, to the point it becomes hopeless, and they realize they need to redo the whole project
- and then a team of the best developers is gathered

- the *tiger team* must build a new system, parallel to the old team
- tiger team is now requested *not only to redo the whole project, but to catch up with the changes happening to the original team*
- time passes, and tiger team realizes their code has become with time no better than the old one!!!
- those races may continue for 10 long years, for absolutely nothing !
- **Clean code** helps in :
  - maintenance
  - modifications
  - readability
  - minimizing number of errors and bugs
  - saving time

## Attitude

- من مسئوليتك العلمية والمهنية انك تكون *صادق* مع المدير
- لو الشغل *هياخد شهر*, متقولش 3 اسابيع, *قول شهر*
- الموضوع عامل زي الجراح اللي المريض قاله "بلاش تعقيم قبل العملية, تضييع وقت على الفاضي"
  لو الدكتور سمع كلامه, يبقا أذاه, هنا المريض جاهل *فمن مسئولية الطبيب* انه يوعيه ويصارحه
- clean code in fact make you FASTER not slower, in the long term

# Chapter 1 : What is clean code

## Grady Booch

> **Quote**
>
> Clean code is *simple and direct*. Clean code reads like well-written prose. Clean code never obscures the designer's intent but rather is full of crisp abtractions and straight forward line of control.
> *clean code should contain only what is neccessary*

## Dave Thomas

> **Quote**
>
> Clean code can be read and enhanced by a developer other than its original author. It has unit and acceptance tests. It has meaningful names. *It provides one way rather than many ways for doing the same thing*. It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API. Code should be literate

since depending on the language, not all necessary information can be expressed clearly in code alone.

### Ron Jiffries

> Quote
>
> In priority of order, simple code :
> - runs all tests
> - contains no duplications
> - Expresses all the design ideas that are in the system
> - Minimizes the number of entities such as classes, methods and functions.

### Ward Cunningham

> Quote
>
> You know you are working on clean code when *each routine you read turns out to be pretty much what you expected*. You can call it beautiful code when the code also makes it "look like the language was made for the problem

### The Boy Scout Rule

> Quote
>
> **Leave the camp cleaner than you found it**

# Chapter 2: Meaningful names

- it's extremely important

## Use intention revealing names

- use the unit itself inside variable's name, for example:

```
int elapsedTimeInDays;
int daysSinceCreation;
int fileAgeInDays;
```

- if a name requires a comment then the name does not reveal its intent, for example:

```
    private int d; //number of books available
```

```
for(int i = 0; i < 2; i++){
    if(arr[i][i] ≠ arr[i + 1][i + 1]){
        y = false;
    }
}
```

the problem isn't the simplicity of the code but the *implicity* of the code.

- compare the last code with this one :

```
for(int cell_index = 0; cell_index < diagonal_size - 1; cell_index++){
    int cell = board[cell_index][cell_index];
    int next_cell = board[cell_index + 1][cell_index + 1];
    if(cell ≠ next_cell)
        player_wins = false;
}
```

> **Quote**
>
> قال رسول الله - صلى الله عليه وسلم - : "من صلى عليَّ أو سأل لي الوسيلة حقت عليه شفاعتي يوم القيامة "
> (رواه مسلم)

## Avoid disinformation:

- do not refer to a grouping of books as "BooksArray" unless it's actually an array!
  - if it's a vector, that would make a great confusion to the reader
- خليك حذر في استخدام الاسماء المتشابهة :
  - XYZControllerForEfficientHandlingOfStrings
  - XYZControllerForEfficientStorageOfStrings
  - programmers use autocompletion, that would make a problem if names aren't *easy to be distinguished* from each other
- avoid using lower case 'L' or uppercase "i" or uppercase 'O' as variable names.
- consider following code :

```
int a = l;
if(O == l)
    a = O1;
```

```
else
    l = 01;
```

## Make meaningful distinctions

- number series naming (a1, a2...aN) is the *opposite of intentional naming*
- Noise words are meaningless distinctions:
  - what is the difference between : ProductInfo and ProductData ???
  - you made variable names different *without them meaning actually something different*

> Quote

قال رسول الله صلى الله عليه وسلم:
"اتق الله حيثما كنت ، وأتبع السيئة الحسنة تمحها ، وخالق الناس بخلق حسن"

## use pronouncable names

- name like :
  - BkDta for : BookData
  - plyr for : player
  - rgstr for : register
- those are non-pronounceable, and consequently we can't *discuss them* in a programming environment without sounding like idiots
- programming is something done between people, which mean it *needs to be communicated* properly in meetings and discussions

## use searchable names

- a length of a name should *correspond to the size of its scope*
- a name like "e" is a bad choice for a variable that is used in multiple places in code body, it's the most frequent letter in English
  - trying to search for it will cause you to go through jungles of strings and code lines to reach it

## Avoid mental mapping

- *don't show off* your mental ability by naming variables with one character and mapping that character in your head to its actual meaning
- شايفك يا بتاع ال problem solving ياللي عايز تسمي ال variables ب x و y

## Class name

- should be a noun, like Customer, WikiPage, AddressParser.

## Method names

- methods should have verb or verb phrase names like : deletePage or save.
- When constructors are overloaded, use static factory methods with names that describe the arguments, for example

```
Complex x = Complex.FromRealNumber(23.0);
```

> Quote
>
> عن أبي سعيد الخدري رضي الله عنه أن النبي صلى الله عليه وسلم قال: "ما من رجل يدعو الله بدعوة ليس فيها إثم، ولا قطيعة رحم، إلا أعطاه الله بها إحدى ثلاث خصال: إما أن يعجل له دعوته، أو يدخر له من الخير مثلها، أو يصرف عنه من الشر مثلها". قالوا: يا رسول الله، إذًا نكثر. قال: "الله أكثر".

## Don't be cute

- انك تسمي اسماء زي nine11 عشان توصف ال destructor بتاع الكلاس tower مثلا حاجة هتخلي الكود غير مقروء وغير منطقي للمبرمج اللي هيشتغل بعدك

## Pick one word per concept

- be consistent with your naming style
  - يعني مثلا ميبقاش *fetch* و *retrieve* و *get* بيعملو نفس الحاجة في classes مختلفة
  - اختار اسم واحد فيهم لمبدأ ما ووحد الاستخدام دا في كل حتة في الكود
  - استخدم add مثلا لجمع قيمتين, وعليه **متستخدمش** اسم add عشان تضيف عنصر ل set of elements مثلا

## Use Solution domain names

- استخدم مصطلحات ال CS للمشاكل المعروفة والمشهورة.
- استخدم *مصطلحات* الحاجة اللي شغال فيها domain specific, يعني لو بتعمل برنامج لتنظيم تخزين المواد الكيميائية بين مجموعة من المعامل, استخدم المصطلحات الكيميائية اللي تعرفها وليها علاقة ببرنامجك, زي ان المادة ليها درجة حرارة ب "تتسامى" عندها او غيره

## Add meaningful context

- لو شفت اسماء زي board, player, score, choice, ممكن تتوقع ان احنا بنحاول نعمل لعبة
  - بس لو شفت كلمة choice لوحدها, هل كان هيجيلك نفس الانطباع ؟
  - وبالتالي بيكون كويس انك تضيف prefix للاسم عشان تدي سياق :
- game_board, player_choice, player_score
- take a look at this java code

```
private void printGuessStatistics(char candidate, int count) {
    String number;
    String verb;
    String pluralModifier;
    if (count == 0) {
      number = "no";
      verb = "are";
      pluralModifier = "s";
    } else if (count == 1) {
      number = "1";
      verb = "is";
      pluralModifier = "";
    } else {
      number = Integer.toString(count);
      verb = "are";
      pluralModifier = "s";
    }
    String guessMessage = String.format("There %s %s %s%s", verb,
number, candidate, pluralModifier);
    print(guessMessage);
}
```

now compare with this one :

```
public class GuessStatisticsMessage {
  private String number;
  private String verb;
  private String pluralModifier;

  public String make(char candidate, int count) {
    createPluralDependentMessageParts(count);
    return String.format("There %s %s %s%s",verb, number, candidate,
pluralModifier);
  }

  private void createPluralDependentMessageParts(int count) {
    if (count == 0) {
      thereAreNoLetters();
    } else if (count == 1) {
      thereIsOneLetter();
    } else {
      thereAreManyLetters(count);
    }
  }

  private void thereAreManyLetters(int count) {
    number =  Integer.toString(count);
    verb = "are";
```

```
      pluralModifier = "s";
    }

    private void thereIsOneLetter() {
      number = "1";
      verb = "is";
      pluralModifier = "";
    }

    private void thereAreNoLetters() {
      number = "no";
      verb = "are";
      pluralModifier = "s";    }
  }
```

**بلاش مبدأ أبو بلاش كتر منه**

---

- in an application called "gas station deluxe" it's a bad idea to prefix every class with GSD
    - لما تيجي تبحث عن اسم هيبقا صداع
- shorter names are usually better than long ones
- **add no more context to a name than is necessary**

# chapter 3 : Functions

consider the following code :

```
public int static Login(String email, String password, boolean admin){
    Session.initialize();
    if(!admin){
        if(!emial.contains("@") || !password.matches(".*[a-zA-Z].*") ||
password.size() < 5 || password.matches(".*\\d.*") || )
            throw RunTimeException("please enter a valid  password");

        boolean f = false;
        for(String e : Database.getemail()){ //actually getUserEmailList
            if(e == email){
                f = true;
            }
        }
        if(f == false)
            throw RunTimeException("user doesn't exist");

        if(!Database.getPassword(email).equals(password))
            throw RunTimeException("password is not correct");

        Database.LogUser(email);
```

```
        System.out.println(Database.getUser(email).getName());
        System.out.println(Database.getUser(email).getBorrowed());
        System.out.println("number of read books :");
        System.out.println(Database.getUser(email).getNBooks());
        System.out.println("read : ");
        System.out.println(Database.getUser(email).getBorrowed());
        return 0;
    }
    else{
        //validate
        if(!emial.contains("@") || !password.matches(".*[a-zA-Z].*") ||
password.size() < 5 || password.matches(".*\\d.*") || )
            throw RunTimeException("please enter a valid  password");

        boolean f = false;
        for(String e : Database.getadmin()){ //actually getUserEmailList
            if(e.equals(email)){
                f = true;
            }
        }
        if(f == false)
            throw RunTimeException("user doesn't exist");

        if(!Database.getPassword(email).equals(password))
            throw RunTimeException("password is not correct");

        for(String e : Database.getemail()){
            String p = database.getPassword(e);
            System.out.println(email, p);
        }

        System.out.println("as admin you can : 1)remove user 2)block
user from borrowing 3)give a user a fine of certain amount");
        Scanner sc = new Scanner(System.in);
        return sc.nextInt();

    }

}
```

- هناك العديد من المصائب في هذا الكود, هل يمكنك أن تراها ؟

## القاعدة الأولي في ال functions : يجب أن تكون قصيرة

## القاعدة الثانية : يجب أن تكون أقصر !

- الكود اللي فات طويل جدا على function واحدة ويدل على اننها *مش بت do one thing*

- أقصرها ازاي ؟ اقسمها لعدة functions أصغر

## one level of abstraction per function :

- ال function هنا بتعمل validation لل input بدلا من انها تستخدم function اسمها validate مثلا, وبالتالي بتضطر تتعامل مع هراء كتير هي في غنى عنه

## Single responsibility principle :

- لو سمحت متقليش مهي بتعمل حاجة واحدة اهو بت login, لان فعليا دا مش صح والسبب :
- ال function اللي فاتت دي مش عارفين هيا بتعمل اي ولا اي ولا اي
  - بت validate inputs
  - بت print data
  - بت take inputs from an admin
  - بتتأكد ان ال password مطابق لمعايير البرنامج مع ان دا ملوش اي لازمة ومش مسئولية ال function دي اصلا !
- كل هذه المسئوليات تجعل من الافكار اللتي نريد التعبير عنها من خلال الكود:
  - مبعثرة
  - *غير قابله للاختبار* (هعرف منين اي بالظبط اللي بايظ)

## function arguments

- ال function اللي بتاخد arguments كتير (3 أو اكتر) هي مؤشر سيء جدا و شيء يجب تجنبه
- أفضل انواع ال functions اللي مبتاخدش arguments خالص**0**
- تاني أفضل نوع هوا اللي بياخد **1** argument
- وهكذا كل ما كان عدد ال arguments *أكتر* كل ما دا كان مؤشر *سيء* انك مش عامل OOP كويس غالبا, لانك مضطر تعرف كتير عن حجات كتير قبل ما تعمل شغلك, طب لي الحجات دي مش private variables في ال class مثلا ؟
- تحتاج الى سبب قوي جدا عشان تخلي function تاخد **4** arguments او اكتر (**ويستثنى** من هذا بعض ال functions طبعا زي :

```
comboBox.getItems().addAll("first option", "second", "third", "fourth",
"fifth")
```

## function name should be a verb

- وخد بالك من الحجات ال double meaning زي مثلا

```
if(device.run()) //developer expected wheather device is actually run or
not


device.run(); //developer expected the device to actually run
```

في المثال اللي فات خليك حذر هوا اي معنى run ؟
طبعا بعض الكلمات بتبقا **عرفا** معروف هيا بتعمل اي فمش لازم تقلق, زي مثلا set و get و move ...الخ
بس في العموم خليك حذر من دا

## flag arguments

- **avoid flag** arguments
  - consider this :

```
int cost = getShortestPath(graph, true); //what do I understand from
this ???
// a better practice is to split the function into two :
int cost = getShortestPathDirected(graph);
int cost = getShortestPathUndirected(graph);
```

# No side effects

- تخيل لو في function اسمها checkRow في لعبة XO مثلا,
انتا مش متوقع من ال function دي انها تعلن عن الفائز لان مش دا شغلها,
فلو أعلنت عن الفائز فدا side effect غير مرغوب فيه, وكدا ال function مش بت do one thing

# Output arguments

- *don't return null*
- look at this :

```
appendFooter(s);
```

هوا انا هنا بديك footer وانتا تضيفه في مكان ما, ولا بديك حاجة وانتا بتضيها لل footer ؟
انتتا هنا مش عارف اصلا ال footer تبقا input ولا output
لكن دا بيتم علاجه قليلا كدا كدا باستخدام OO سليم :

```
report.appendFooter(s);
```

- ***Anything that forces you to check the function signature should be avoided***

# Command Query Separation

- يا اما تجاوبني على سؤال او تعمل حاجة ولكن مش الاتنين مع بعض
- consider this example

```
if(board.update_board(cell, 'X')){

}
```

```
else{
    throw update_failed_error();
}
```

في المثال السابق ال function دي بت update **و** ت check اذا كان ال update ينفع اصلا يتعمل ولا لا, خدت بالك من حرف العطف **"و"** دا معناه انها بتعمل حاجتين *مش حاجة واحدة*
واحنا اتفقنا انها لازم تعمل حاجة واحدة مش حاجتين
في الحالة دي افصلهم ل 2 functions واحدة بت check اذا كان ال cell دي مش ملعوب فيها قبل كدا وواحدة تانية بتغير محتوى ال cell

## Exceptions are better the error codes

طبعا ال error codes بتاعت ال web استثناء لكن في الطبيعي تجنب هذة الاكواد الصماء غير معلومة المعنى وارمي *exception* افضل

## Error handling is one thing

```
//this is bad
PrayerTimes getPrayerTimes(){
    try{
        String times = RemotePrayersGetter.getAPIPrayerTimes();
    }
    catch(Exception e){
        //Tell us what happened
        System.out.println("API is not availabe right now, times are
calculated locally throw astronomical calculations");
        Logger.logError("API not responding at time :
".concat(LocaleDateTime.now().toString()));

        //fix the issue
        LocalDateTime
Zuhr=PrayerTimesCalculator.getNextZuhr(LocaleDateTime.now());

        LocalDateTime
Asr=PrayerTimesCalculator.getNextAsr(LocaleDateTime.now());

        LocalDateTime
Maghrib=PrayerTimesCalculator.getNextMaghrib(LocaleDateTime.now());
        //etc for rest of the prayers
    }
}
```

```
//but this one is better
public PrayerTimes getPrayerTimes(){
    try{
        String times = RemotePrayersGetter.getAPIPrayerTimes();
    }
```

```
    catch(Exception e){
        return HandelAPINotRespdoning();
    }
}

private PrayerTimes HandelAPINotRespdoning(){
    LogAPIError();
    return getAstronomicalPrayerTimes();
}

private getAstronomicalPrayerTimes(){
    LocalDateTime
Zuhr=PrayerTimesCalculator.getNextZuhr(LocaleDateTime.now());
        LocalDateTime
Asr=PrayerTimesCalculator.getNextAsr(LocaleDateTime.now());
        LocalDateTime
Maghrib=PrayerTimesCalculator.getNextMaghrib(LocaleDateTime.now());
    //etc you got the idea
}
```

## DRY (Don't Repeat Yourself)

- ال *duplication* يا ولدي قد يكون *جذر وسبب كل الشرور* في عالم ال software , فأي وقت تحس نفسك بتكرر حاجة, انتا بنسبة ضخمة جدا بتعمل حاجة غلط, فوقف وهات ورد استغفار وفكر ممكن تعمل اي يصلح هذة المشكلة

### فاكر ال technical writing ؟

- فاكر لما كنا بنعمل rough draft و first draft و second و third ربما وهكذا ؟
- هوا الموضوع كدا بالظبط, *مش لازم تعمل code مثالي من اول كتابة*, لكن ارجع وعدل في اللي كتبته وضيف ونقص وحسن منه, مش لازم يطلع مثالي في اول مرة

### متنساش تاخد بصة في الكتاب على المثال اللي في اخر ال Chapter عشان مفيد

> **Check**
>
> شكرا ليك انك لسا مكمل قراية

# Chapter 4 : Comments

## Don't comment bad code--rewrite it

الكومنتات يا ولدي **necessary evil** are , شر لا بد منه
اوعا تفتكر ان الكومنتات الاكتر معناها كود أنضف

الكومنتات هيا الفازات اللي بيحطوها لما المحافظ يعدي عشان يداروا بيها البلاوي والشكل القبيح للمكان,
*لو كان كودك واضح وكويس مكنتش احتجت كومنت اصلا*

## Explain yourself in code

```
// if user can borrow book
if(!book.isBorrowed() && book.isBorrowable() && book.isNew()){
    //code's logic ...
}
```

this is better

```
if(book.CanBeBorrowed()){
    //code's logic ...
}
```

## انا من الجن الطيب

there are some comments that are ok,
for example :

```
//copyright (C) 2003,2004 by some inc. All rights reserved
//released under the terms of chocolate License of being sweet ... etc.
```

## Informative comments

```
//if matches U.S social Security Number SSN format NNN-NN-NNNN
if (myRegex.match(^\d{3}-\d{2}-\d{4}$
);){
//do some stuff ...
}
```

## Explanation of Intent

let this be a function in a symbol class in a board game that only uses character symbols

```
public int compareTo(Object object){
    if(object instanceof Character){
        //some stuff IDK ...
    }
    return 1; //we are bigger because we are the right type
}
```

the last comment tries to justify why the code behaves this way, so it's okay

## Clarification

```
if (LocalDate.now().getMonthValue() == 5 &&
LocalDate.now().getDayOfMonth() == 1){
    manager.giveVacationToAllEmployees(); //because this is عيد العمال
brother
}
```

## Watch out !

```
//don't run unless you have time, it tests with a very big set of data!
void theHugeTest(){
    //some stuff ...
}
```

## TODO comments

حلوة وبسمسم

## Amplification

لو في حتة مهمة من الكود ممكن عادي تنبه انها مهمة

## Avoid vague comments

انتا عامل الكومنت عشان توضضح الفكرة يعني متكتبليش كدا مثلا

```
//this solves the problem of n * r / t , using the previous algorithm
void somefunc(){
    //some code ...
}
```

What n ?
What r ?
what algorithm ?
what problem ?
bro what ???

اتق الله فينا يا اخي

## update comments when you update code

```
//this function gets Hijri Date from a free API Egypt-prayer-
times.api.gov.eg
String getPrayerTimes(){
    establishConnection("https://Makkah-prayer-times.api.gov.eg");
```

```
//bruh, Makkah or Egypt ?????
}
```

## Noise comments

- بجد عيب تبقا بالغ راشد عاقل وتعمل حاجة زي كدا :

```
//this function sets salary
void setSalary(int salary){
    this.salary = salary;
}

//this function checks validity of ID card number
boolean isIDCardValid(Card card){
    //some logic ...
}
```

هوا حضرتك معلق كورة مثلا منا شايف قدامي اهو

## Don't use comments where you can use a function or a variable

## position markers

```
//class//////////////////////////


//implementation//////////////////

//player stuff////////////////////

//board stuff/////////////////////

//validate ID/////////////////////
```

All of those are **ominous warnings** that we may fail to heed.
So refactor your code please

## Closing brace comments

```
                }//end if prime
            }//end if odd
        }//end if a number
}//end of try block
```

دا دليل برضو انك محتاج تقصر حجم الكود بتاعك جدا

## Don't comment out code and leave it

يعني قطعة من الكود مبقيتش محتاجها ومع ذلك انتا سايبها وحاططها كومنت
الناس كلها هتبقا خايفة تمسحه وهيفضل الكود ملعبك

### انا صانع المعجزات

```
//by the very great and smart and intelligent and beautiful Maryoumi ❤
```

مع كامل احترامي للاستاذة مريم, بس في version control system يسجل عملك العظيم, مش لازم تعرفينا انك عايزة تتجوزي او حاسة بالنجاح والرضا عن حالك

## HTML comments

يا أخي ارجوك اتعالج, لية تعمل comments ب HTML مين فاضي ياخدها يحطها في برنامج بيقراها, ما تكتب زي الناس

> هل تعلم أن الصدقة تطفئ غضب الرب، وتدفع ميتة السوء
> قال رسول الله - صَلَّى اللهُ عَلَيْهِ وَسَلَّمَ -: "صنائع المعروف تقي مصارع السوء، وصدقة السر تُطفئ غضبَ الربِّ، وصِلَةُ الرَّحِم تزيد في العمر". رواه الطبراني في "الكبير" *بإسناد حسن* كما قال الشيخ الألباني في صحيح الترغيب والترهيب.

# Chapter 5 : Formatting

## File size should be suitable

Don't go put everything in one file man

## Vertical openness between concepts

كل مجموعة من السطور بتعبر عن فكرة معينة المفروض تبقا قريبة من بعضها ومفصولة عن غيرها بسطور فاضية

## variable declarations should be close to their usages as much as possible

## instance variables should be declared at the top of the class

# if a function calls another, they should be close

## file structure :

```
public void somefunc(){ //first layer of abstraction
    func1();
    func2();
}

private void func1(){ //second layer of abstraction
    funcA();
}

private void func2(){ //second layer
    funcB();
}

private void funcA(){ //third layer

}
private void funcB(){ //third layer

}
//keep the low level details to the last as much as possible ***********
```

## java is not assembly

- it's generally a *bad idea* to do something like this :

```
private   Socket              socket;
private   InputStream         input;
protected long                reqquestParsingTimeLimit;
private   SomeLongClassName   someclass;
```

## indentation

يرحم والديك متعملش حاجة زي كدا :

```
int hello(){
for(int i = 0; i < 5; i++)
int x=0;
if(true == true){
if(false == false){
if(something.isCool())

}
```

```
else(how.areYou()){
//do something ...
}
}
}
```

ولكن بدلا من ذلك اعمل كدا :

```
int hello(){
for(int i = 0; i < 5; i++){
    int x = 0; //note the space between the x and the = and the 0
    if(true == true){
        if(false == false){
            if(something.isCool()){

            }
            else(how.areYou()){
                //do something ...
            }
        }
    }
}
```

**يرحم والديك متحطش المشروع كله في فايل واحد واكيد مش كلاس واحد واكيد اكيد مش فانكشان واحدة**

**بالله عليك اعمل تسميه كويسة للفايلات, مترفعش ملف اسمه index.html على الريبو بتاعتنا والفايل دا مش ال home page**

**اعمل directories متحطش كله في حتة واحدة**

**بوصولك الى هذة المرحلة فأنت عندك ما لا يسع المبرمج جهله من ال clean code for structured programming , الجزء الجاي هيكون عن object-oriented principles ولو قرأت الجزئية دي في *الكتاب* انتا ممكن متحتاجش تقرا كتاب تاني في OOP الا هم الا ال design patterns وبعض الكماليات الاخرى**

# Chapter 6 : Objects and Data Structures

- allow users to manipulate the essence of the data *without having to know its implementation*.
- the golden rule is : `don't access the class internals to do something, instead ask the class to provide it to you`

## Data/Object Anti Symmetry :

- while classes hide their data behind abstractions and expose functions the operate on the data, Data structures on the other side expose the data, and don't provide any useful functions

> **Note**
>
> *Procedural* code (ابو data structure) makes it easy to *add new functions* without changing the existing data strucures. *OO code* on the other hand, makes it eady to *add new classes* without changing existing functions

for example :
procedural :

```
int calculate_salary(string employee){
    if(employee == "hourly"){
        //some logic ...
    }
    else if (employee == "monthly"){
        //some logic ...
    }
    else if (employee ==  "jonior"){
        //some logic ...
    }
}
```

But what happens when I add a new type of employees?
the function itself changes !

On the other hand:

```
class Employee{
    //some attributes and stuff
    int calculate_salary() = 0;
};

class hourly_Employee : public Employee{
    //some stuff

    //override
    int calculate_salary(){
        //some logic ...
    }
};
```

```
class monthly_Employee{
    //some stuff

    //override
    int calculate_salary(){
        //some logic ...
    }
};


class jonior{
    //some stuff

    //override
    int calculate_salary(){
        //some logic ...
    }
};


//and when I want to use it, I just do :
int main(){
    Employee* employee = new monthly_Employee("Islam");
    cout << employee->calculate_salary();
}
```

## قاعدة متدخلنيش في تفاصيل

*a module should not know about the innards of the objects it manipulates*

## الطابور الصباحي :

```
int x = Library.getAuthor(bookID).getAddress().getStreet().getID();
```

ما هذا المنظر البشع يا معشر ال clean coders ?
لية المفروض انا كشخص بتعامل مع كتاب اعرف ان الكتاب ليه كاتب والكاتب ليه عنوان والعنوان ليه...الخ
ممكن ييجي في دماغك حل اننا نعمل كدا :

```
int x = Library.getAuthorStreetID(bookID);
```

بس انتا لو مشيت بالمبدأ دا هتعمل functions عددها لا نهائي

طب لحظة واحدة, لية انا اصلا مضطر اتعامل مع كل دا
يمكن مثلا بعد كل الهري دا انا مجرد محتاج اوتوجراف ؟!

طب اي رأيك نقول :

```
Library.GetAutographed(BookID);
```

دا أسهل بكتير وبيوفر عليا كتير جدا وبيوضح بالظبط كل حاجة مسئولية مين,

لتوضيح النقطة دي تخيل اننا عندنا class عربية, مش مضطر اقوله هاتلي الموتور وبعدين هاتلي الموبينة وبعدين هاتلي فرق الجهد,
ومش محتاج اعمل function خارجية بتجيبلي فرق الجهد وواحدة تانية بتجبلي ضغط الزيت وواحدة تالتة بتجيبلي مضخة البنزين...الخ
ولكن بدلا من ذلك, مش انتا يا عم عايز فرق جهد المبينة ؟
طب *عايزه لي ؟*
عشان اتأكد ان الموتور شغال كويس,
طب خلاص متقلقش, احنا هنعملك function اسمها checkup بتشيك على العربية كلها
او ممكن function اسمها checkMotorUp بتشيك على الموتور تحديدا اذا لزم الامر, متشغلش بالك انتا بالتفاصيل

*برضو الجملة اللي فاتت دي مش دقيقة علميًا اوي لان مش طبيعي في السياق العادي ان العربية بتشيك على نفسها, بس انا بفترض مثال عشان اوضح الفكرة مش اكتر, لكن في الطبيعي ال function بتاعت checkUp دي كان هيعملها class اسمه mechanic*

## avoid hybrid classes/data structures

- don't make a class that has public attributes, and at the same time has functions that do significant stuff, it just *takes the worst from both worlds*.

## Data Transfer Objects

- حجات كدا زي ال data structures بس بتخلي ال variables برايفت مش اكتر
- الزبدة انها *كويسة* استخدمها

# Chapter 7 : Error Handling

## Use exceptions rather than return codes

- عشان أحسن

### مفيش حلاوة من غير نار

- checked exceptions, for sorry, has a price
- and that price is *violation of the open/closed principle*
- تخيل معايا أخي المسلم انك عندك ال function دي :

```
//suppose that's a program to track your children Athkar
pulbic class AthkarDatabase{
    public static boolean HasUserSaidAthkar(Integer userID){
```

```
        return (
            sabah(userID) &&
            masa(userID) &&
            prayersAthkar(userID)
        );
    }

    private static Boolean sabah(Integer userID){
        getUserData();
        //do some logic to optain the boolean variable talking about
 sabah athkar
    }

    private static getUserData(Integer userID){
        //some logic
        if(DBnotResponding)
            throw new Exception("data base not responding");
    }


}
```

- ما الذي يحدث هنا يا عزيزي ؟
- اه فاهم انها حاجة غريبة اني ببعت ال ID بتاع ال user كل مرة, دي فعلا حاجة ممكن تتحسن بس دا مش قصدي

## بطاطا سخنة

في المثال اللي فات دا احنا حرفيا بنلعب بطاطا سخنة, كل واحد *بيرمي ال exception للي قبله*
لازم تضيف ال exception في ال signature بتاعت getUserData
وبعدين هتضيفها برضو لل signature بتاعت اذكار الصباح
وبعدين هتضيفها برضو لل function الاساسية بتاعت الاذكار كلها
وبعدين تمسك ال exception في try/catch block في ال function اللي استعدت ال function الاساسية دي في الاول خالص
شفت وجع القلب فين ؟
*بتغير functions كتير ملهاش علاقة بمشاكلك وحواراتك*

بكل صدق معرفش الموقف اللي فات دا ممكن يتصلح ازاي ولكن اللي اعرفه, **بقدر الامكان, لو تقدر تتعامل مع ال exception بنفسك من غير ما ت throw حاجة, اعمل كدا افضل كتيير**

او عادي استخدم unchecked exceptions دا احيانا بيكون افضل من العبث دا *ودي هي توصية مؤلف الكتاب اصلا*
(بس خلي بالك انه متطرف في بعض اراؤه في الcode فمتصدقش اي حاجة يقولهالك واحد صه..يوني (ايوة هوا صه..يوني بجد مش هزار بيدعم اللي ما يتسموا))

## مالك يا ولية ؟ => مفيش

- لما بتحط exceptions من غير سياق زي : UnExpectedBehaviourException
  والله بجد ؟؟ يبني مهو اكيد unexpected متقول حاجة مفيدة !

- حط اسم و رسالة exception معبرة عن نوايا العملية اللتي فشلت وأدت لظهور ال exception في الاساس
- اديني سياق متقوليش انا زعلان ولما اقلك مالك تقولي مفيش لو مهتم كنت عرفت

# Chapter 9 : Unit tests

## Keep tests clean

- having dirty tests is equivalent to, if not worse than, having not tests at all.

## Testing code is as important as production code

## Number of asserts per test should be minimized

## single concept per test

## Tests should be F.I.R.S.T :

### Fast :

- if it's slow you will be more lazy than to run it after changes

### Independent

- a test should not depend on another one

### Repeatable

- In any environment

### Self validating

- have a boolean output: pass or fail

### Timely

- Should be **before** production code

> Quote
>
> قَالَ ٱهْبِطَا مِنْهَا جَمِيعًا ۖ بَعْضُكُمْ لِبَعْضٍ عَدُوٌّ ۖ فَإِمَّا يَأْتِيَنَّكُم مِّنِّى هُدًى فَمَنِ ٱتَّبَعَ هُدَاىَ فَلَا يَضِلُّ وَلَا يَشْقَىٰ
> وَمَنْ أَعْرَضَ عَن ذِكْرِى فَإِنَّ لَهُۥ مَعِيشَةً ضَنكًا وَنَحْشُرُهُۥ يَوْمَ ٱلْقِيَـٰمَةِ أَعْمَىٰ

# Chapter 10: Classes

# class organization

- class should start with a list of variables: public static constants --> private static variables --> private instance variables
- and after that public functions, and after each public function, the private functions it uses.

# Encapsulation

- use private *as much as possible* (instead of accessing something ask the class to give it to you)

# Classes should be small

- the first rule of classes is: they should be small
- the second rule is: *they should be smaller*
- but the word **small** here isn't about size, but rather *responsibilities*
  consider this example

```
public class Car{
    private int horsePower;
    private int CC;
    private int weight;
    private String serialNumber;
    private int kilomemters;
    private int maximumSpeed;
    private int numberOfPassengers;
    private String brand;
    ...

    public void washCar(){
        //some logic
    }

    public void fixCar(){
        //some logic
    }

    public void sellCar(){
        //some logic
    }
    ...
}
```

تراك زودتها يا الطيب كفاية

انتا كدا مش هتخلص ال class بيعمل كل حاجة في الدنيا

افترض معايا ان اللي فات دا class في معرض سيارات مثلا,
ساعتها الافضل ان الكلاس يبقا كدا مثلا :

```
public class car{
    private String serialNumber;
    private CarSpecs specs;
    private int price;
    //getters and setters
}
```

بهذه الطريقة انتا قصيت الكلاس لعشر حجمه على الاقل وشغلت بالك بالمهم فقط وأصبح للكلاس *مسئولية واحدة*
washCar is not the responsibility of the car itself

as a general rule: `if you can say class function itself` then it's its reposibility
for example: `a car washes itself` is not valid
`a car fixes itself` is not valid
`a car sells itself` is not valid
but rather we would have: cleaner, fixer and mechanic who does that for the car

## Single responsibility principle (SRP)

- you may fear having single responsibility will result in a lot of classes, but remember that a `a system with many small classes has no more moving parts than a system with a few large classes`

## A class should have a small number of instance variables

## Organizing for change

- *التغيير هو العدو الأول للمبرمج*, وهو بلا شك ملازم ليه عبر مسيرته المهنية, وبالتالي انتا بتكتب الكود بحيث يقدر يواكب التغييرات, عشان لما تحصل متبوظش اكواد كانت شغالة
- **Classes should be open for extension, but closed for modification**

## Dependency Inversion principle (DIP)

- Classes should depend upon abstractions, not on concrete details

```
public class laptop {
    private IntelProcessor processor;
    //rest of the class ...
}
```

what will happen if we use AMD instead of intel?
if requirements changes, this code will not be valid, and we will be forced to **modify** it.

instead we do:

```
public class laptop{
    private Processor processor;
    //rest of the class ...
}
```

# Chapter 11: Systems

## Seperate constructing a System from using it

## Seperation of Main

- move all aspects of construction to Main, or modules called by Main, and design the rest of the system assuming that all objects have been constructed and wired up appropriately

```
public class laptop{
    public laptop(Processor x){
        x = new IntelProcessor(7, 7500, 'H');
    }
}
```

This is Wrong, because it assumes the processor wasn't already created.

## Factories are good

- for more info: https://refactoring.guru/design-patterns/abstract-factory

## Dependency Injection

- it's an application of *Inversion of Control* (moving secondary responsibilities to other objects dedicated for it)
- `an object should not take repson. fro instantiating dependencies itself`

## Scaling UP

`it's a myth that we can get systems "right the first time" instead we should implement only today's story`

اعمل بس حاجة شغالة وبعدين ت scale up وتطبق ال OOP principles وال design patterns
بس اهم حاجة تعمل في الاول حاجة شغالة

## Test Drive the System architecture

- Big Design Up Front *(BDUF) is bad*
- An Optimal system architecture consists of modularized domains of concern
- the architecture can be test-driven just like the code

## Optimize decision making

- postpone decisions as long as you can to gather knowledge you need

## Use standards wisely

- because some standards are too old, or that your condition is different than usual

# Chapter 12: emergence

## Simple design Rule 1: Runs all tests

## Rule 2: Refactoring

`the fact that we have those tests eliminates the fear that cleaning code might break it up`

## Rule 3: No Duplication

## Rule 4: Expressive

- using design patterns names in the names of your classes is a clue for whoever reads the code that you use that pattern

## Rule 5: Minimal Classes and Methods

- دي هيا القاعدة الأقل أهمية, وبالتالي اتأكد من ان ال ٤ الاولانيين تمام قبل ما تطبق دي,
  المقصد انك *متبالغش بزيادة اوي* في عدد ال classes وال methods

> **Tip**
>
> صل على النبي

## توصيات صاحب الملخص

- استخدم vim key bindings او emacs لان بجد بجد لا غنى عنهم, مش لازم تستخدم vim text editor بس على الأقل ال key bindings لان بجد بتوفر كتير جدا
- استخدم اي tool او extension مفيدة تشوفها وهتكتشف ان في كتير جدا
- خلي ال AI يساعدك مش يكتبلك

- Think twice, code once
  - وانتا بكتب الكود فكر اي اللي ممكن يخلي الكود دا مش شغال
  - بص بعينك على *مشاريع مكتوبة* وادرسها وفصصها كويس لغاية ما تفهمها وتعرف اتكتبت ازاي وهتلاقي open source كتير لكل لغة
  - ال multi inheritance وحش ولكن لو هت Implement اكتر من interface يبقا كويس
- Search about: **how to solve diamond inheritance**
  - اتعلم ال factory pattern, observer pattern لكن لا تسهب في تعلم ال patterns عموما
- Learn the standards of any language you are using
  - ساءلتك بالله يا أخي تتعلم github عشان الواحد بيشوف حجات صعبة اوي