

Clean Code

Chapter 2: Meaningful Names

author: Hamed Damirchi

hameddamirchi32@gmail.com

github.com/hamed98

linkedin.com/in/hamed-damirchi-ba4085178/

Author: Hamed Damirchi

از نام‌هایی استفاده کنید که مقصود شما را مشخص کند

نام گذاری نباید احتیاج به کامنت داشته باشد

نام گذاری باید طوری باشد که بگه چرا این متغیر/تابع هست و چی کار میکنه

نمونه غلط:

```
int d; // elapsed time in days
```

متغیر d هیچ اطلاعاتی به ما نمیدهد

نمونه های صحیح:

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

معلوم نیست هدف این کد چیه:

```
public List<int[]> getThem() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
    return list1;  
}
```

این نام گذاری این موارد رو مبهم میزاره:

- درون theList چه چیز هایی قرار میگیرن؟
- x[0] چیه اصلا؟
- عدد ۴ چیه؟
- از لیست ریترن شده چطور باید استفاده بشه؟ فقط یه آرایه دو بعدی از اعداده صرفا

نمونه صحیح: فرض کنید کد مربوط به بازی مین یاب باشد

```
public List<int[]> getFlaggedCells() {  
    List<int[]> flaggedCells = new ArrayList<int[]>();  
    for (int[] cell : gameBoard)  
        if (cell[STATUS_VALUE] == FLAGGED)  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```

حالا خیلی چیزها مشخص شد! عدد ۴ یعنی فلگ و $x[0]$ وضعیت رو توش نگه میداره.

حالت صحیح تر:

```
public List<Cell> getFlaggedCells() {  
    List<Cell> flaggedCells = new ArrayList<Cell>();  
    for (Cell cell : gameBoard)  
        if (cell.isFlagged())  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```

از دادن اطلاعات گمراه کننده خودداری کنید

- از عبارات و کلماتی که منظور شما با مفهوم آن ها متفاوت است استفاده نکنید.
- مثلا عبارات hp,aix,sco عبارات پلتفرم یونیکس هستند و ترجیحا نباید از چنین چیز هایی استفاده کنید. چون به معنایی که در یونیکس دارند جا افتاده شده است.
- همچنین به یه گروه مثلا اکانت نگید accountList مگه این که واقعا لیست باشن. بهتره بگید accountGroup یا accounts یا bunchOfAccounts.
- از نام گذاری دو چیز به صورتی که نام ها شبیه هم باشن و گمراه کننده بشن اجتناب کنید مثلا XYZControllerForEfficientStorageOfStrings و XYZControllerForEfficientHandlingOfStrings
- استفاده نکردن از حرف بزرگ O و حرف کوچک l به عنوان متغیر (با صفر و یک اشتباه گرفته میشه)

```
int a = 1;
if ( 0 == 1 )
    a = 01;
else
    1 = 01;
```

تمایز های معنی دار ایجاد کنید

وقتی دو تا متغیر جدا دارین، اسامی متغیر ها هم باید تمایز رو برسونه اضافه کردن عدد یا حروف بی مفهوم صرفا برای تکراری نبودن اسامی اشتباه است از متغیر های a_1, a_2, a_3, \dots استفاده نکنید. این ها درسته گمراه کننده نیستن (مثل قبل) اما اطلاعی نمیرسونن.

```
public static void copyChars(char a1[], char a2[]) {  
    for (int i = 0; i < a1.length; i++) {  
        a2[i] = a1[i];  
    }  
}
```

بهتره تو این مثال از source و destination استفاده بشه.

استفاده از noise words:

مثلا به متغیر Product داریم، بعدش دو تا متغیر اضافه میکنیم: ProductInfo و ProductData. کلمه های Data و Info هم معنی هستن و باعث اشتباه میشن. بنابراین وقتی دو متغیر داریم، اسم هر کدوم هم باید به معنی متفاوتی بده

اسم های بی معنی: variable, NameString, Tabe (for table)
اگر به کلاس داریم به اسم Customer دیگه نباید کلاس CustomerObject داشته باشیم. چون تمایز رو نمیرسونه.

مثال اسامی غیر قابل تشخیص از هم در صورت نبود قرارداد:

- moneyAmount and money
- customerInfo and customer
- accountData and account
- message and theMessage

استفاده از اسامی قابل تلفظ

اگر قابل تلفظ نباشن اسم ها نمیتونی بخونی واسه خودت و این طوری میشه:

“Well, over here on the bee cee arr three cee enn tee we have a pee ess zee kyew int, see?”

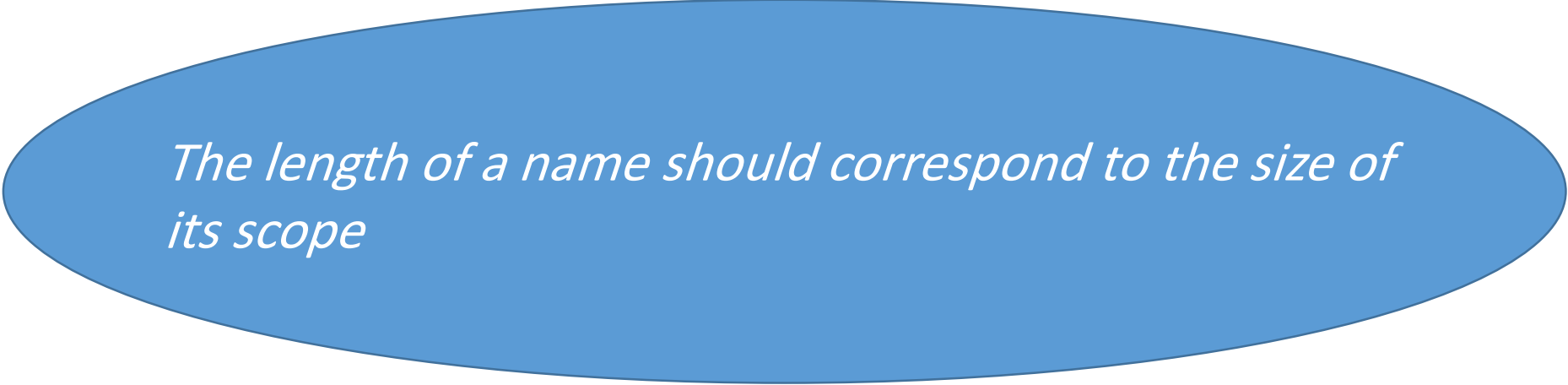
استفاده از genymdhms برای خلاصه کردن generation date, year, month, day, hour, minute, and second. خب به جاش از generationTimestamp استفاده کن یا اسمی بزار که بتونی تلفظ کنی

```
class DtaRcrd102 {  
    private Date genymdhms;  
    private Date modymdhms;  
    private final String pszqint = "102";  
    /* ... */  
};
```

```
class Customer {  
    private Date generationTimestamp;  
    private Date modificationTimestamp;;  
    private final String recordId = "102";  
    /* ... */  
};
```


اسامی قابل جستجو انتخاب کنید

مثلا متغیر MAX_CLASSES_PER_STUDENT خیلی قابل جستجو تره تا عدد ۷ !
اسم های خیلی کوچک (مثلا تک حرفی ها) رو خیلی سخت میشه سرچ کرد.
سعی کنید از اسامی تک حرفی تنها در تابع های خیلی کوچیک استفاده کنید



The length of a name should correspond to the size of its scope

اگر یه متغیر قراره تو جاهای مختلفی استفاده بشه (اسکوپ بزرگتری داره) حتما باید اسمی بدید بهش که راحت قابل جستجو باشه.

مقایسه دو کد یکسان:

```
for (int j=0; j<34; j++) {  
    s += (t[j]*4)/5;  
}
```

```
int realDaysPerIdealDay = 4;  
const int WORK_DAYS_PER_WEEK = 5;  
int sum = 0;  
for (int j=0; j < NUMBER_OF_TASKS; j++) {  
    int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;  
    int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);  
    sum += realTaskWeeks;  
}
```

کد گذاری نکنید

استفاده از کد گذاری زیاد بار ذهنی میاره و لازمه هر کسی که میخواد رو این پروژه کار کنه تمام اون ها رو قبلش یاد بگیره. همچنین اسامی کد گذاری شده معمولا سخت تلفظ میشن

کد گذاری نکنید - Hungarian notation

نماد مجارستانی نوعی کد گذاریه که تایپ رو هم مشخص میکنه. مثال ها:

- lAccountNum -> l: long
- arru8NumberList -> arru8: array-unsigned-8bit
- strName: string
- fBusy: flag

این روش کد گذاری در گذشته بیشتر مرسوم بود مخصوصا در مایکروسافت
اما الان کامپایلر ها خودشون تایپ دارن و معمولا احتیاجی نیست
اگر بعدا بخوای یه تایپ دیگه بریزی تو متغیر چی؟

کد گذاری نکنید - Member Prefixes

بعضی ها متغیر های مربوط به یه کلاس یا تابع رو با `m_` شروع میکنند (نشان دهنده `member`). اما اشتباهه. باید کلاس ها و توابع انقدری کوچک باشه که احتیاج به این پیشوند نباشه. ضمناً، آدم ها معمولاً موقع خوندن معمولاً پیشوند رو ناخودآگاه نادیده میگیرن (البته پسوند رو نه!)

```
public class Part {  
    private String m_dsc; // The textual description  
    void setName(String name) {  
        m_dsc = name;  
    }  
}
```

```
public class Part {  
    String description;  
    void setDescription(String description) {  
        this.description = description;  
    }  
}
```

کد گذاری نکند- Interfaces and Implementations

- فرض کنید به اینترفیس `AbstractFactory` داریم. که قراره چند تا کلاس اونو پیاده سازی کنن.
- خیلی جاها این اینترفیس رو `IAbstractFactory` نام گذاری میکنن. اما نویسنده ترجیح میده اینترفیس ها هیچ چیز اضافه ای نداشته باشن و اگر قرار باشه پیشوند یا پسوند بزاریم، بهتره پیاده سازی ها پیشوند یا پسوند داشته باشن.
- مثلا `ShapeFactoryImp` یا `CShapeFactory` (C نشانه Concrete)

از نگاشت ذهنی پرهیز کنید

- طوری نام گذاری نکنید که خواننده مجبور شه تو ذهنش اونو به یه اسم دیگه نگاشت کنه
- این نام گذاری بیشتر مرتبط با اسامی تک حرفه. مثل ا درون حلقه که اگر حلقه طولانی شه، خواننده هر بار که ا رو میبینه اونو به یه اسم مرتبط دیگه تو ذهنش نگاشت میکنه
- خیلی ها فکر میکنن این که از mental mapping استفاده کنن یعنی باهوشن!
- در حالی که برنامه نویس حرفه ای میدونه که clarity is the king

نام گذاری کلاس ها

- اسم کلاس ها یا باید اسم باشد یا گروه اسمی (مثل کتاب، کتاب ریاضی)
- اسم یه کلاس به هیچ وجه نباید فعل باشد، تابع یه کار انجام میده نه کلاس
- اسامی خوب: Customer, WikiPage, Account, AddressParser

نام گذاری تابع ها

- تابع ها باید فعل یا گروه فعلی باشن. مثل `postPayment, deletePage, save`
- توابع تغییر دهنده مقدار باید با `set`، خواننده مقدار با `get`، و توابع خبری (predicate ها) با `is` شروع بشن.

```
string name = employee.getName();  
customer.setName("mike");  
if (paycheck.isPosted())...
```

*جالب: وقتی `constructor` قراره `overload` بشه، بهتره اصلا کلاس رو استاتیک کنی و از الگوری `factory` استفاده کنی و تابعی بنویسی که اون مفوم با `overload` مربوطه رو برسونه. تو ساخت تابع هم میتونی سازنده رو پرایوت کنی و ...

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

is generally better than

```
Complex fulcrumPoint = new Complex(23.0);
```

بانمک نباشید! Don't Be Cute

- اسم هایی که تو زبان محاوره استفاده میشه انتخاب نکنید

- HolyHandGrenade() vs DeleteItems()

- whack() vs kill()

- اصطلاحات مختص به یه فرهنگ استفاده نکنید

- eatMyShorts() vs abort()

- Say what you mean. Mean what you say.

برای هر مفهوم یک کلمه انتخاب کنید

- مثلاً داشتن سه تابع `fetch`, `retrieve`, `get` با هم تو یه برنامه اشتباهه. چون معلوم نیست واسه دریافت اطلاعات از کدوم باید استفاده کرد.
- همچنین مثلاً `controller`, `manager`, `driver`

از جناس استفاده نکنید(یه کلمه برای چند کار)

- همیشه از قاعده "یک کلمه برای هر مفهوم" استفاده کنید
- مثلا فرض کنید چند تا کلاس داریم و همشون تابع `add` رو دارن. این تابع مثلا یه چیز رو به یه لیست اضافه میکنه و لیست جدید رو برمیگردونه
- حالا فرض کنید بخوایم یه تابع به یه کلاس دیگه اضافه کنیم که یه مقدار رو به یه لیست اضافه کنه فقط (ریترن نداره). این جا بهتره اسم این تابع رو `add` نذاریم چون نحوه کارش با `add` قبلی متفاوته.
- بهتره بزاریم `insert,append`

از اسامی مرسوم کامپیوتری ها استفاده کنید (solution domain names)

- اگر از یک الگوریتم خاصی استفاده میکنید، حتما اسم الگوریتم رو استفاده کنید.
- اگر از یک فرمول ریاضی مشهور استفاده میکنید، حتما اسم فرمول رو لحاظ کنید
- اگر از یک دیزاین پترن استفاده میکنید حتما از اسم اون دیزاین پترن استفاده کنید
- مثلا اسم AccountVisitor به اسم عالی هست، اگر از پترن VISITOR استفاده میکنید.
- یا مثلا اگر از ساختار Queue استفاده میکنید JobQueue اسم خیلی خوبیه.

از Problem Domain Names استفاده کنید

اگر هیچ اسم کامپیوتری مناسب پیدا نکردید از اسم های problem domain استفاده کنید
یعنی اسم هایی که با هدف پروژه به طور خاص مرتبط باشد.

زمینه معنادار اضافه کنید (add meaningful context)

- فقط چند تا اسم هستن که از روی اسم میتونی بفهمی محتوا راجع به چیه و چند تا معنی ازش برداشت نمیشه.
- اگه اسمی داری که چند تا معنی ازش برداشت میشه بهتره محتوای مربوطه رو هم به اسم بدی
- مثلا اگه چند تا متغیر واسه آدرس داری مثل:
`firstname,lastname,street,housenumber,city,state,zipcode`
میدونیم State لزوما به معنی استان نیست. بنابراین بهتره به این فیلد ها مثلا `addr` اضافه کنیم به عنوان پیشوند.
- اما راه بهتر اینه که همه رو به یه کلاس کنیم. با این کار کانتکست متغیر ها هم معلوم میشه

ادامه: مثال

تابع صفحه بعد نمونه ای از تابعیه که با نگاه کردن بهش همیشه به محتوا پی برد، باید کامل تابع رو بخونیم که متوجه بشیم. از اسامی و ساختار به راحتی همیشه متوجه کانتکست شد.

این تابع طولانی است. در حالی که میشه با تکه تکه کردن و تبدیل به یه کلاس اونو به یه ساختار خوشگل در آورد!

Listing 2-1**Variables with unclear context.**

```
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);
}
```

Listing 2-2

Variables have a context.

```
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";
    }
}
```

زمینه پرت و پرت اضافه نکنید!

- مثلاً اگر اسم اپلیکیشن شما Gas Station Delux عه، اضافه کردن GSD به اول همه کلاس‌های این برنامه کار اشتباهیه!
- تا جاییه که میتونید سعی کنید اسامی کوتاه باشن، اما با رعایت نکات گفته شده.
- مثلاً کلاس accountAddress یا customerAddress خوبه اما Address خالی بهتره.
- اگر آدرس‌های متفاوتی دارید مثل MACAddress و PortAddress و URLAddress بهتره به صورت MAC,URL,PostalAddress باشه.

حرف آخر

از تغییر نام متغیر ها نترسید.
نترسید از این که با تغییر نام و ریفکتور کردن به طور صحیح، سایر افراد تیم گیج بشن.