Meaningful Names

Well-written code

- …does one thing well – Bjarne Stroustroup(C++ inventor
- ...reads like well written prose – Grady Booch(UML inventor)
- Anyone can write code that a computer understands, but few programmers know how to write code that a human can understand – Martin Fowler

Why well-written code?

- True cost of software = its maintenance cost (80% of the total)
- Why? Because we get slower, as the code degrades over time

Why well-written code?

- We READ 10x or more times than we WRITE
  - So make the code more readable, even if it's harder to write, takes a lot of time, etc.
  - Boy scout rule – always check in cleaner code than you found
- Today, I stopped refactoring = Today, my application became Legacy.

“well written prose”

- Functions are verbs
  - product(), transaction()
  - searchProduct(), sendTransaction()
- Boolean names should answer yes/no
  - isGolden(), areStreamsValid()

“well written prose”

- Classes are nouns – Customer, Order
- Avoid meaningless names – Order~~Info~~, Order~~Data~~ vs Order
- Delete the Interfaces – ~~I~~CustomerService, OrderService~~Impl~~

How do you understand a function?

- You read the /* dusty old comment */ ?
- You hunt down the code calling it ?
- You decrypt its implementation ?

- You should NOT need to do that!
- **Make the name speak for itself !**

Meaningful Names

- Names are everywhere in software.
  - Variables
  - Functions
  - Arguments
  - Classes
  - Packages
  - Source files
  - The list goes on and on…

Names should be…

- Pronounceable
  - ~~**int** getinvcdtlmt()~~
  - **Int** getInvoiceableCreditLimit()
- Avoid abbreviations – unless it's a basic business concept, like VAT

Clean Code

- **Use Pronounceable Names**

```
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";
  /* ... */
};
```

Clean Code

- **Use Searchable Names**
  - One might easily grep for MAX_CLASSES_PER_STUDENT
  - If a variable or constant might be seen or used in multiple places in a body of code, it is imperative to give it a search-friendly name.

```
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;
}
```

Clean Code

- **Avoid Encodings**
  - IShapeFactory and ShapeFactory?
  - How do you name interfaces and their implementations ?

Meaningful Names

- Member Prefixes (Avoid encodings)

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

Meaningful Names

- Member Prefixes (Avoid encodings)

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

Meaningful Names

- Hungarian Notation (Avoid encodings)

PhoneNumber phoneString;

// name is not changing when the type is changing!

Meaningful Names

- Hungarian Notation (Avoid encodings)

PhoneNumber phone;

Meaningful Names

- **Use Intention-Revealing Names**
  - The name of a variable, function, or class, should answer all the big questions. It should tell you why it exists, what it does, and how it is used.
  - If a name requires a comment, then the name does not reveal its intent.

Clean Code

- **Class Names**
  - Classes and objects should have noun or noun phrase names.
  - Customer, WikiPage, Account, and AddressParser.
  - Avoid words like Manager, Processor, Data, or Info in the name of a class.
  - A class name should not be a verb.

Clean Code

- **Method Names**
  - Methods should have verb or verb phrase names
  - postPayment, deletePage, or save.
  - Accessors, mutators, and predicates should be named for their value and prefixed with get, set, and is according to the javabean standard.

(The last one is not common is some dynamic languages)

Meaningful Names

- Method Names

```
postPayment, deletePage, save
// methods should have verb or verb phrase names

String name = employee.getName();
customer.setName("mike");
if (paycheck.isPosted())…

Complex pivotPoint = Complex.fromRealNumber(23.0);
// is generally better than
Complex pivotPoint = new Complex(23.0);
```

Names should be…

- Consistent
  - .find..()
  - .fetch..()
  - .get..()
  - Stick to naming conventions
- Unique – synonyms confuse
  - Don’t use *buyer* or *client* to refer to *customer* – use this, if that’s the term most widely used in the bussiness
- Don’t create a gap between Business and IT

Names should have…

- Meaningful context
  - Order.customerFirstName
- But Don’t Repeat Yourself
  - Order.~~order~~CreationDate

Clean Code

- int d; // elapsed time in days
- int elapsedTimeInDays;

What is the purpose of this ?

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

1. What kinds of things are in theList?
2. What is the significance of the zeroth subscript of an item in theList?
3. What is the significance of the value 4?
4. How would I use the list being returned?

Clean Code

- Is this better ?

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

Name length

- Variables: long scope -> long name

```
public class OrderService {
  public final static int STATUS_ACTIVE=…;
  private OrderRepository orderRepository;

  public void copy(Order from, Order to) {
    boolean wasDelivered;
    for (int i = 0; i < 100; i++) { ... }
  }
}
```

Name length

- Private function should have long descriptive names

Clean Code

- **Avoid Disinformation**
  - Int hp = 10; **??? Do not use abbreviations.**
  - Beware of using names which vary in small ways.
    - XYZControllerForEfficientHandlingOfStrings
    - XYZControllerForEfficientStorageOfStrings

Clean Code

- Spelling similar concepts similarly is **information**. Using inconsistent spellings is **disinformation**.
  - account
  - accounts
  - acount

Clean Code

- **Make Meaningful Distinctions**
  - You can’t use the same name to refer to two different things in the same scope,
  - You might be tempted to change one name in an arbitrary way.

Clean Code

- Noise words are another meaningless distinction.
  - ProductInfo or ProductData, you have made the names different without making them mean anything different. Info and Data are indistinct noise words.

Noise words are redundant. The word variable should never appear in a variable name. The word table should never appear in a table name.

Clean Code

- Customer and CustomerObject.
  - Which one will represent the best path to a customer's payment history?

```
getActiveAccount();
getActiveAccounts();
getActiveAccountInfo();
```

  - **Distinguish names in such a way that the reader knows what the differences offer.**

Clean Code

- **Avoid Mental Mapping**
  - Readers shouldn't have to mentally translate your names into other names they already know.

```
for (a = 0; a < 10; a++)
    for (b = 0; b < 10; b++)


for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++)
```

Clean Code

- **Don’t Be Cute…**
  - If names are too clever, they will be memorable only to people who share the author’s sense of humor, and only as long as these people remember the joke.

Clean Code

- **Pick One Word per Concept**
  - It’s confusing to have **fetch**, **retrieve**, and **get** as equivalent methods of different classes.
  - How do you remember which method name goes with which class?
  - It’s confusing to have a **controller** and a **manager** and a **driver** in the same code base.
  - add/remove, open/close - Avoid using the same word for two purposes

Clean Code

- **Don’t Pun**
  - We have many classes where **add** will create a new value by adding or concatenating two existing values.
  - We are writing a new class that has a method that puts its single parameter into a collection.
  - Should we call this method add?

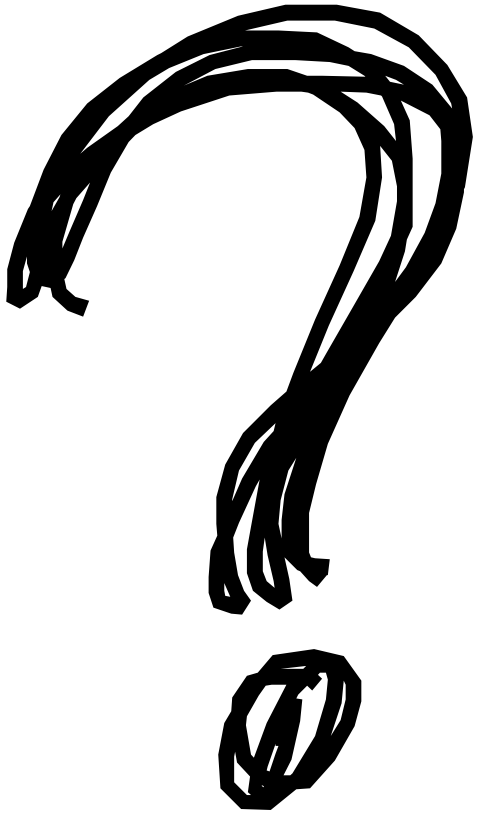
Clean Code

- **Use Solution Domain Names**
  - use computer science terms
  - It is not wise to draw every name from the problem domain because we don’t want our coworkers to have to run back and forth to the customer asking what every name means when they already know the concept by a different name.

Clean Code

- **Use Problem Domain Names**
  - When there is no “programmer-eese” for what you’re doing, use the name from the problem domain. At least the programmer who maintains your code can ask a domain expert what it means.


СВ. КЛИМЕНТ ОХРИДСКИ



THANK YOU