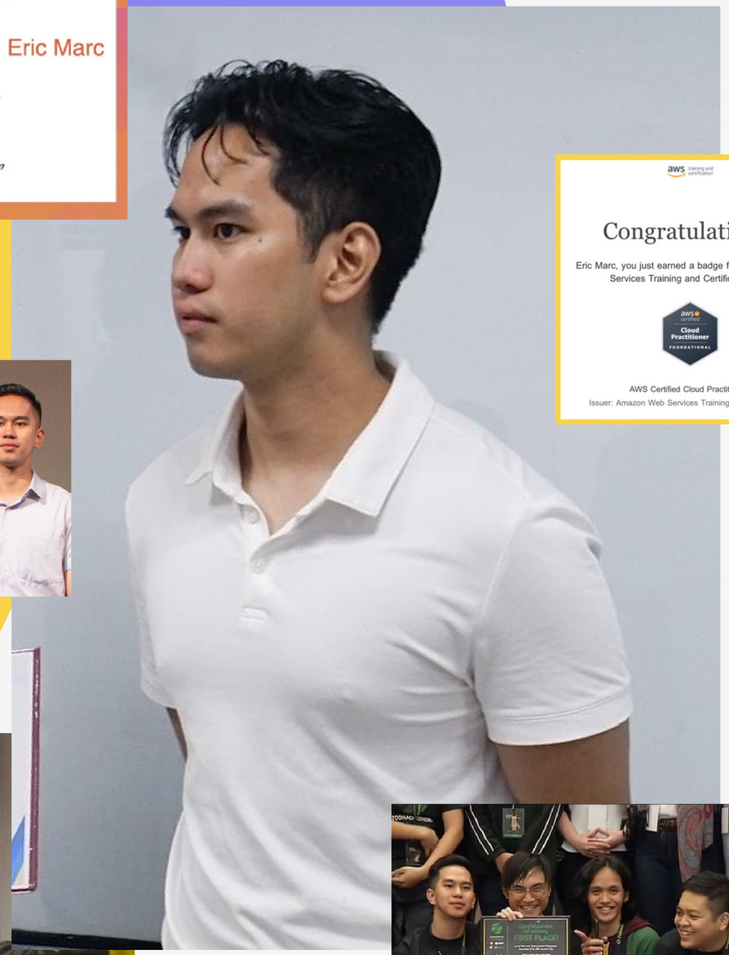
Clean Coding in JAVA

by Eric Marc Martin
Application Developer, IBM



Eric Marc Martin

OVERVIEW

Application Developer – Java & Web Technologies (2022-Present)
IBM

Software Engineer (2020 - 2022)
OOCL (Philippines) Inc.

Bachelor of Science in Information Technology (2020)
Technological Institute of the Philippines - Quezon City

Winner (2019)
Zathon Philippines

TRAINING AND CERTIFICATES

Amazon Web Services - Certified Cloud Practitioner
Amazon Web Services Training and Certification

Agile Full Stack Developer Bootcamp
ThoughtWorks

ABOUT THE SPEAKER

Eric Marc Martin

LEADERSHIP BACKGROUND

Java User Group - Philippines
Member (Present)

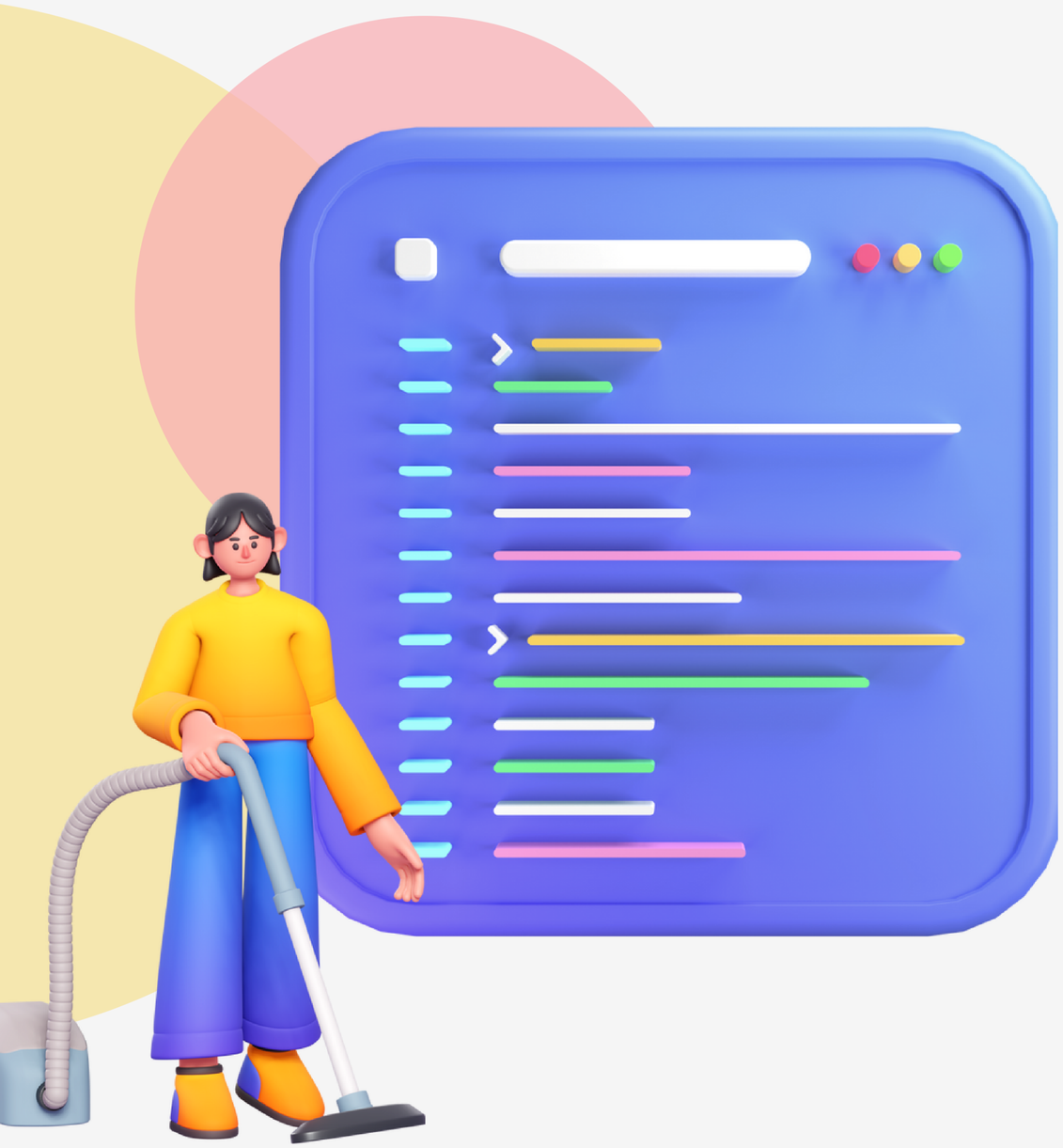
American Chamber of Commerce of the Philippines, Inc. - Business Leadership Program
Member (2020)

Supreme Student Council, TIP
Board Member (2019-2020)

IT Department Student Council, TIP
Vice President (2018-2019)

Rate your confidence in programming
0 - lowest and 9 - highest

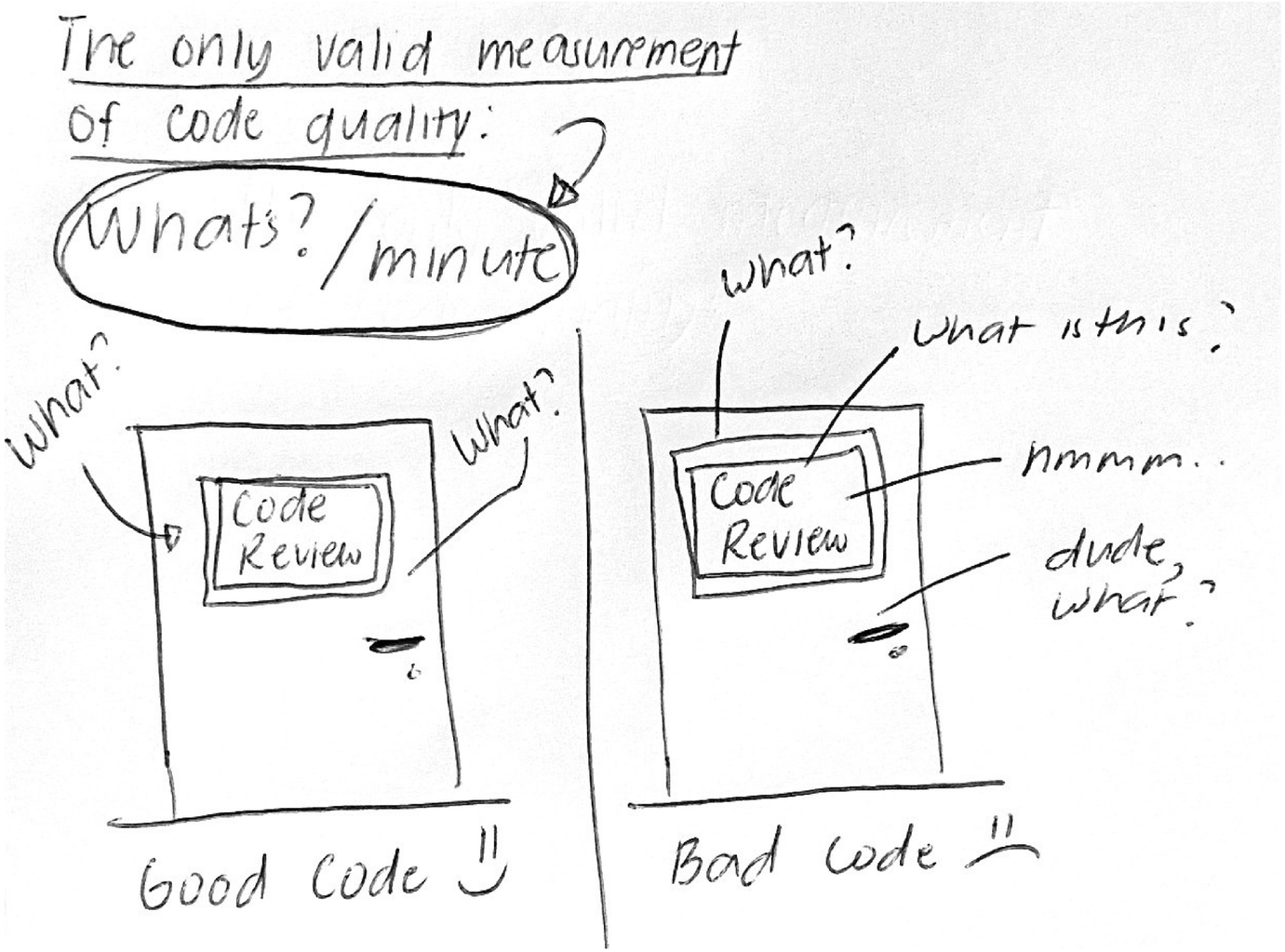
0 | 1 | 2 | 3 | 4
5 | 6 | 7 | 8 | 9



ABOUT THE TOPIC

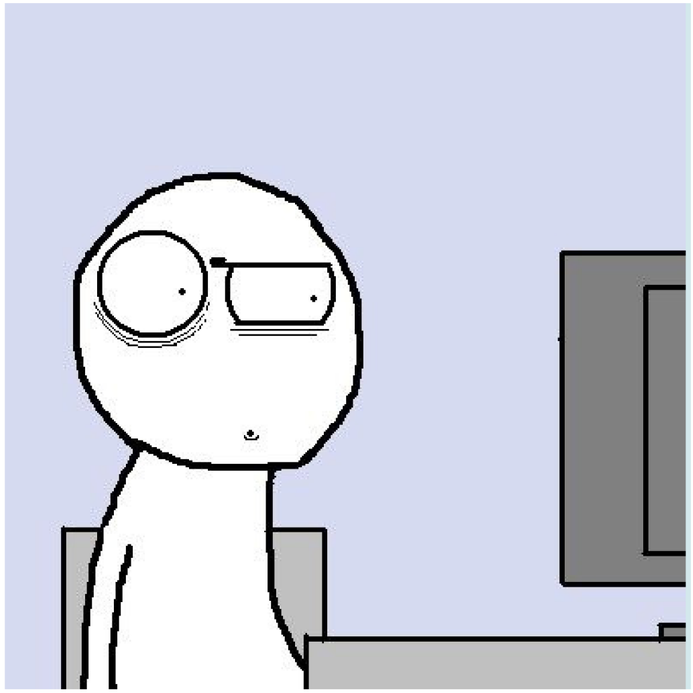
"this is my first time hearing about clean code"

WHICH DOOR REPRESENTS YOUR CODE?



Meme inspired from http://www.osnews.com/story/19266/WTFs_m

HOW ABOUT THIS CODE?



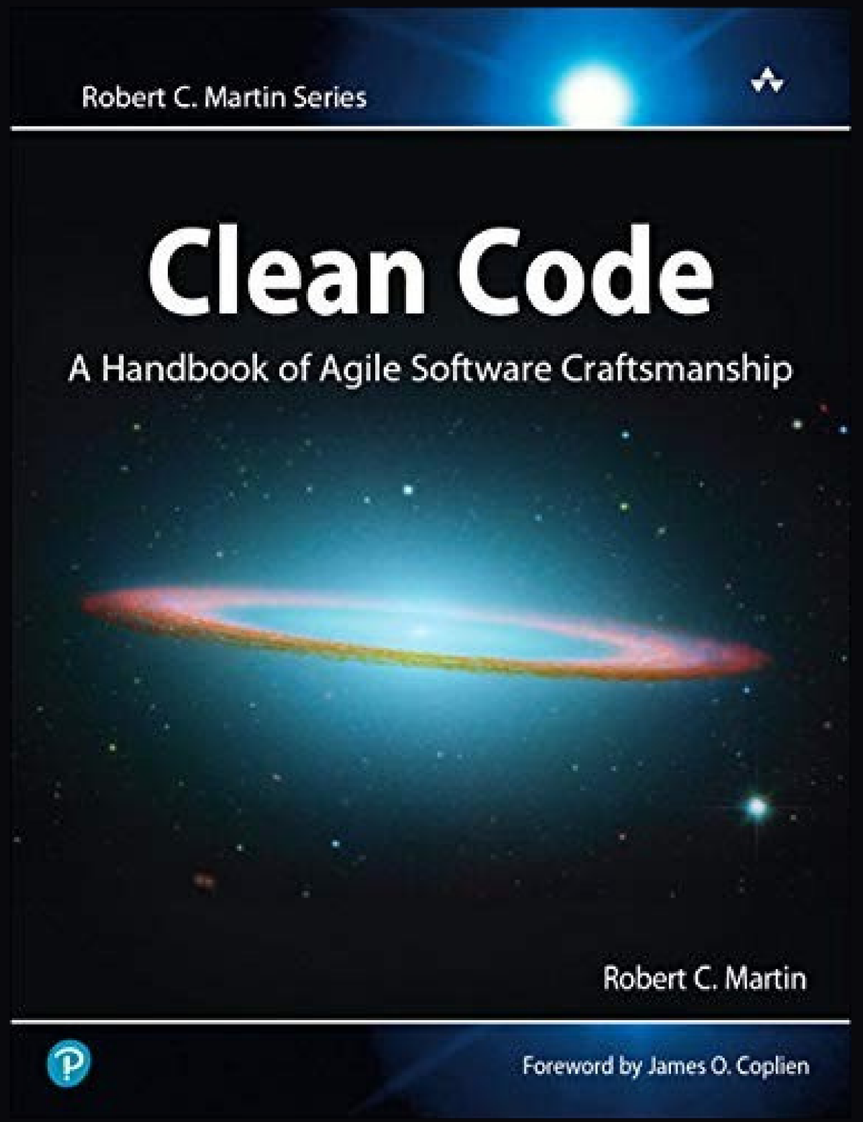
```
public boolean isValidRating() {
    if (this.getRating() != null) {
        if (this.getRating().substring(0, 1).equalsIgnoreCase("B")
                && this.getRating().length() == 2) {
            if (StringUtils.isNumeric(this.getRating().substring(1, 2))
                    && Integer.parseInt(this.getRating().substring(1, 2)) > 0
                    && Integer.parseInt(this.getRating().substring(1, 2)) < 5)
                return true;

        } else if (this.getRating().substring(0, 1).equalsIgnoreCase("A")
                && this.getRating().length() == 3
                && StringUtils.isNumeric(this.getRating().substring(1, 3)))
            return true;

    }
    return false;
}
```

- **does not show its intention right away**
- **time consuming**
- **repetitive (looping, method calls)**
- **lack of meaningful variables**

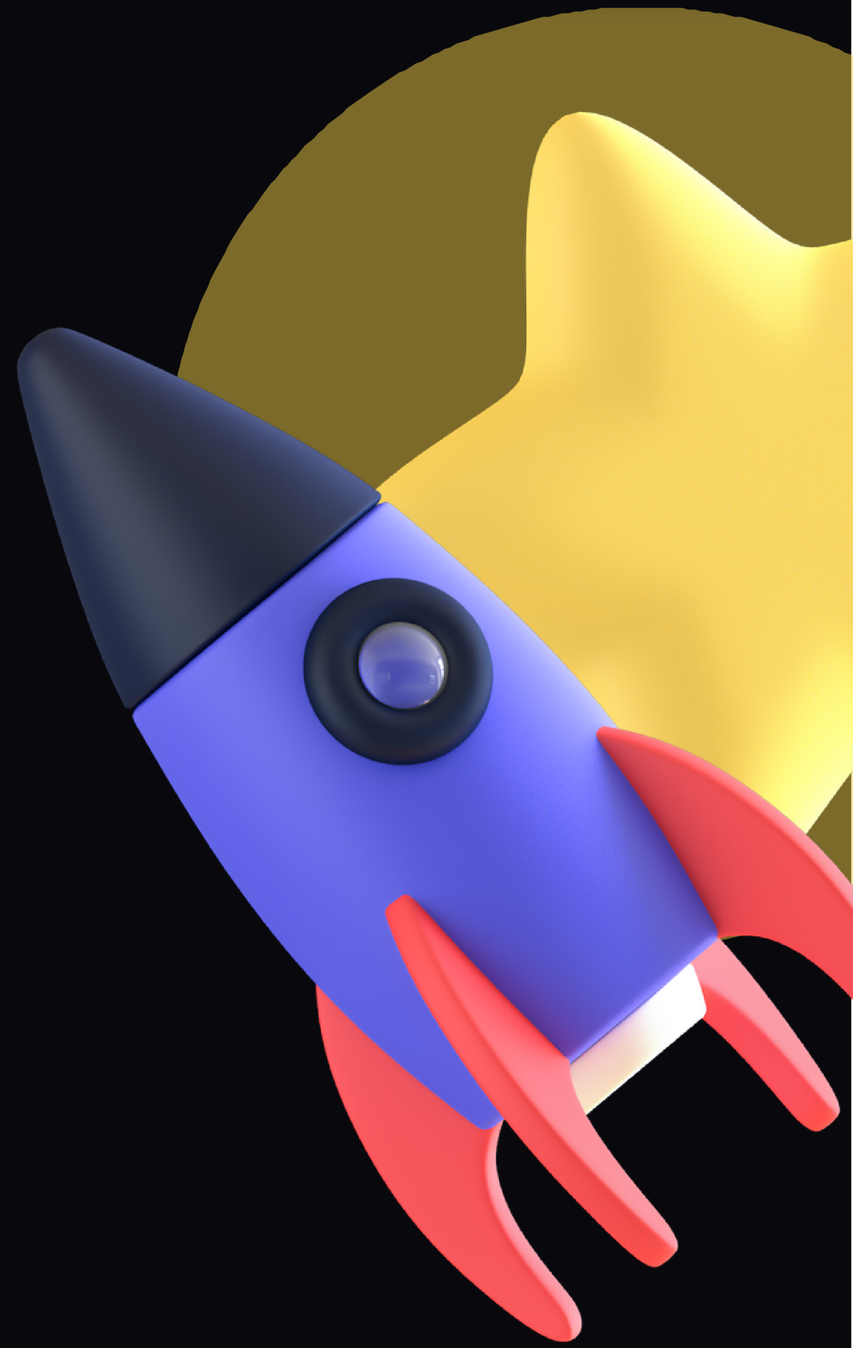
Clean Code



Clean Code

A Handbook of Agile Software Craftmanship

By Robert C. Martin



TESTIMONIAL

Clean Code

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..

“Big” Dave Thomas
Founder of OTI, godfather of the Eclipse strategy
Picture from www.davethomas.net/biography_index.html

TABLE OF CONTENTS

MEANINGFUL NAMES.

Use Intention-Revealing Names
Avoid Disinformation
Make meaningful distinctions
Don't be cute
Use pronounceable names
Use searchable name
Hungarian Notation
Avoid Mental Mapping
Class Names
Method Names
Pick one word per concept
Add Meaningful Context

FUNCTIONS.

Blocks and Indenting
Switch Statements Use descriptive names
Extract Try/Catch Blocks
Avoid repetition

GOOD COMMENTS.

Legal Comments
Explanation of Intent
Warning of Consequences
TODO Comments Javadocs in Public APIs

CODE REFACTORING.

```php
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_pasaword_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

ABOUT THE TOPIC

What makes a code not clean?

Meaningful Names

Use Intention-Revealing Names

```java
public class UseIntentionRevealingNames {
    static int n = 7; //number of days
    int day;
    String[] arr; //students array

    static int numberOfDays = 7;
    int elapsedTimeInDays;
    int daysSinceRegistration;

    String[] students;
    String firstName;

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

    public static void main(String[] args){
        System.out.println("There are " + n + " days in a week");
        System.out.println("There are " + numberOfDays + " days in a week");
```

It should tell 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.

Meaningful Names

Avoid Disinformation

```
11  public class AvoidDisinformation {
12      String accountList;
13      String accounts;
14      String accountGroup;
15      String names;
16
17      String account;
18      String student;
19      String[] enrolledStudents;
20      List<String> courseList;
21  }
```

Developers must not leave false clues that obscure the meaning of code

Meaningful Names

Make Meaningful Distinctions

```
6   public class MakeMeaningfulDistinctions {
7   @       private List<String> getAllStudents() {
8               return new ArrayList<String>();
9           }
10
11  @       private List<String> getAllEnrolledStudents() {
12              return new ArrayList<String>();
13          }
14
15  @       private List<String> getEnrolledStudentsById(int studentNum) {
16              return new ArrayList<String>();
17          }
18  }
```

If names are different, then they should also mean something different.

Meaningful Names

Don’t be cute / funny

```
3   public class DontBeCute {
4       boolean massacre() {
5           //do something
6           return true;
7       }
8
9       boolean studentYern(){
10          return true;
11      }
12
13      boolean tamaBaYon(){
14          return true;
15      }
16  }
```

If names are too funny, 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.

Meaningful Names

Use Pronounceable Names

```
6   public class UsePronounceableNames {
7       ArrayList<String> StdntRcrd() {
8           //do something
9           return new ArrayList<String>();
10      }
11
12      Date genymdhms;
13      //(generation date, year, month, day,
14      // hour, minute, and second)
15  }
```

Programming is a social activity.

If you can't pronounce it, you can't discuss it without sounding funny and no sense.

"Well, over here on the g e n y m d h m s...?"

Meaningful Names

Use Searchable Names

```
public class UseSearchableNames {
    int n;
    int e;
    int numberOfStudents;
    String tempName;
```

The name 'e' or 'n' is a poor choice of any variable name.

Use of single character name should be avoided.

5 minutes break

Is this a good code?

```
int day;
String sName;
String stringCourses;
String subjects;
List<String> attendees;
List<Integer> blockedlistedIDList;
```

Is this a good code?

```
16      public int getThem(){
17          int s = 0;
18          int[] t = new int[50];
19          for (int j=0; j<34; j++){
20              s += (t[j]*4)/5;
21          }
22          return s;
23      }
24
25  @   public static void copyChars(char a[], char b[]) {
26          for (int i = 0; i < a.length; i++) {
27              b[i] = a[i];
28          }
29      }
```

Meaningful Names

Hungarian Notation

```
3   public class HungarianNotation {
4       int intDay;
5       String stringCourse;
6       char charInitial;
7
8       //vUsing adjHungarian nNotation vMakes
9       // nReading nCode adjDifficult.
```

The name 'e' or 'n' is a poor choice of any variable name.

Use of single character name should be avoided. Ex:

- wParam : word-size parameter
- lParam
- hwndFoo : handle to a window
- lpszBar : long pointer to a zero-terminated string

Meaningful Names

Avoid Mental Mapping

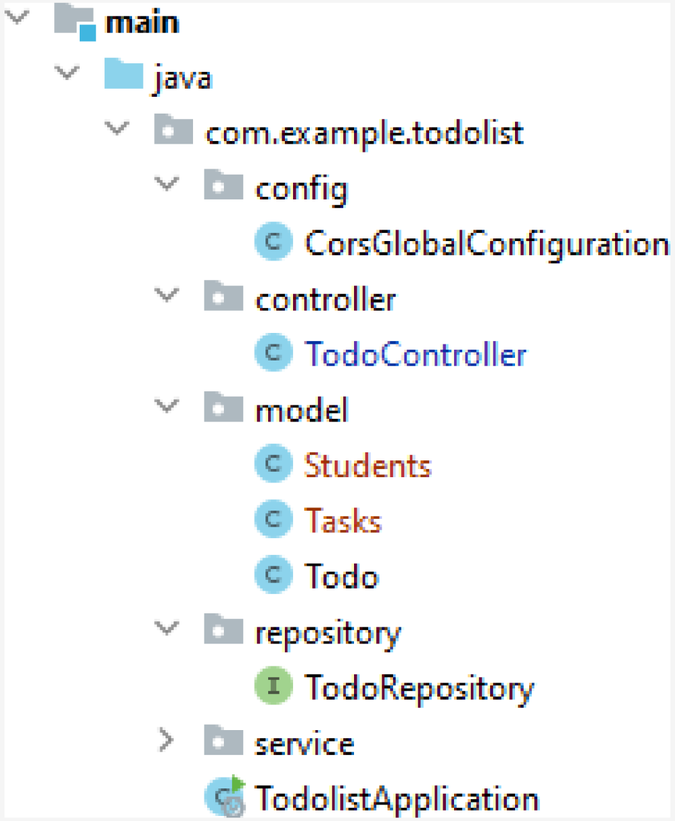
```
6   public class AvoidMentalMapping {
7       public List<String> getDetailsWOName(){
8           return new ArrayList<String>();
9       }
10
11      public List<String> getNameWithoutName(){
12          return new ArrayList<String>();
13      }
14  }
```

Readers shouldn't have to mentally translate your names into other names they already know.

As a professional programmer, we understand that *clarity is king*.

Meaningful Names

Class Names



Classes and objects should have noun or noun phrase.

A class name must not be a verb.

Meaningful Names

Method Names

```
public List<String> getDistinctGames() { return myFavouriteGames.stream().distinct().collect(Collectors.toList()); }

public List<String> getMappedGames(){
    return myFavouriteGames.stream().map(gameTitle -> "This is my favourite game: " + gameTitle)
            .collect(Collectors.toList());
}

public List<String> getGamesWithFiveCharacters(){
    return myFavouriteGames.stream().filter(gameTitle -> gameTitle.length() == 10).collect(Collectors.toList());
}

public Boolean getGameWithStartingLetterH(){
    return myFavouriteGames.stream().anyMatch(gameTitle -> gameTitle.startsWith("H"));
}
```

Methods should have verb or verb phrase names like postPayment, deletePage, or save.

Meaningful Names

Pick One Word per Concept

```
6   public class PickOneWordPerConcept {
7       public String getName(){
8           return "";
9       }
10
11      public List<String> retrieveCourses(){
12          return new ArrayList<String>();
13      }
14
15      public List<String> fetchEnrolledStudents(){
16          return new ArrayList<String>();
17      }
18  }
```

Pick one word for one abstract concept and stick with it.

What is the essential difference between a DeviceManager and a DeviceController?

Meaningful Names

Add Meaningful Context

```
6   public class AddMeaningfulContext {
7       String street, houseNumber, city, zipCode, state;
8       String addrStreet, addrHouseNumber, addrCity, addrZipCode, addrState;
9   }
```

Sometimes context is necessary for your reader to understand your code.
State can be referred to many things - address, or a particular condition.

Meaningful Names

Do One Thing

```
public void updateFooStatusAndRepository(Foo foo) {
   if ( foo.hasFjord() ) {
      this.repository(foo.getIdentifier(), this.collaborator.calculate(foo));
   }

   if (importantBusinessLogic()) {
     foo.setStatus(FNAGLED);
     this.collaborator.collectFnagledState(foo);
   }
}
```

If you can extract another function from a function with a name that is not its restatement of its implementation - then it's doing more than one thing.

Meaningful Names

Blocks and Indention

```
6   public class BlocksAndIndention {
7
8       public List<String> getUserStatusByInfo1(String firstName, String lastName, String middleName, List<String> AddressInfo, String referenceNumber){
9           return new ArrayList<String>();
10      }
11
12      public List<String> getUserStatusByInfo2(String firstName, String lastName, String middleName,
13                                               List<String> AddressInfo, String referenceNumber){
14          return new ArrayList<String>();
15      }
16  }
```

```
public void codeBlockAndIndention() {
for (int i =1;i<=5; i++) {
for(int j=1; j<=i;j++) {
System.out.print("Hi!"); }
System.out.println("Hello!"); } }
```

What can you say about the codes?

Functions

Switch Statements

```
3       public class SwitchStatements {
4   @       public void printAddress(String accountNumber){
5               switch(accountNumber){
6                   case "1A":
7                       System.out.println("123 Don Antonio, QC");
8                       break;
9                   case "1B":
10                      System.out.println("123 Don Jose, QC");
11                      break;
12              }
13      }
14  }
```

Problems with this code:

1. Address changes from time to time
2. Account numbers increase

Functions

Use Descriptive Names

```
public List<String> getGameCharacters(){
    return myFavouriteGames.stream().filter(gameTitle -> gameTitle.length() == 5).collect(Collectors.toList());
}

public List<String> getGamesWithFiveCharacters(){
    return myFavouriteGames.stream().filter(gameTitle -> gameTitle.length() == 5).collect(Collectors.toList());
}
```

What do you think is the problem here?

Functions

Extract Try/Catch Blocks

```
3   public class ExtractTryCatchBlock {
4       public void delete(Course course) {
5           try {
6               deleteCourseAndReferences(page);
7           } catch (Exception e) {
8               logError(e);
9           }
10      }
11
12      private void deleteCourseAndReferences(Course course) throws Exception {
13          delete(course);
14
15          int courseId = course.getId();
16          studentInfo.deleteCourse(course.courseId);
17          studentInfo.deleteReference(course.courseId);
18      }
19
20  @   private void logError(Exception e) {
21          logger.log(e.getMessage());
22      }
23  }
```

Try/catch blocks are ugly in their own right. They confuse the structure of the code and mix error processing with normal processing.

Functions

Avoid Repetition

```
public boolean isDocumentMatch(RevisedDocumentDetails revisedDocumentDetails, List<CompiledArchives>
                               compiledArchives) {
    for (CompiledArchives compiledArchive : compiledArchives) {
        if (revisedDocumentDetails.getRoomId().equals(compiledArchives.getClassId()) &&
                revisedDocumentDetails.getTitle().equals(compiledArchive.getSubject())) {
            return true;
        }

        if (revisedDocumentDetails.getRoomId().equals(compiledArchives.getClassId()) &&
                revisedDocumentDetails.getAuthor().equals(compiledArchive.getAuthor())) {
            return true;
        }

        if (revisedDocumentDetails.getRoomId().equals(compiledArchives) &&
                revisedDocumentDetails.getPublishedDate().equals(compiledArchive.getCreateDate())) {
            return true;
        }
        return false;
    }
}
```

What do you think is the problem here?

Functions

Avoid Repetition

```
public List<String> arrangeCustomerDetail(ToolApplicant toolApplicant) {
    List<String> customerDetail = new ArrayList<>();
    customerDetail.add(toolApplicant.getId());
    customerDetail.add(toolApplicant.getFullName());
    customerDetail.add(toolApplicant.getBirthDate());
    customerDetail.add(toolApplicant.getEmail());
    customerDetail.add(toolApplicant.getApplicantType());
    customerDetail.add(toolApplicant.getApplicationStatus());

    return customerDetail;
}
```

```
public List<String> arrangeInquirerDetail(ToolApplicant toolApplicant) {
    List<String> inquirerDetail = new ArrayList<>();
    inquirerDetail.add(toolApplicant.getId());
    inquirerDetail.add(toolApplicant.getFullName());
    inquirerDetail.add(toolApplicant.getBirthDate());
    inquirerDetail.add(toolApplicant.getEmail());
    inquirerDetail.add(toolApplicant.getApplicantType());
    inquirerDetail.add(toolApplicant.getApplicationStatus());

    return inquirerDetail;
}
```

What do you think is the problem here?

5 minutes break

Good Comments

Legal Comments

```
2   Legal Comments
3   // Copyright (C) 2015-2024 by ConsumerApp Services, Inc. All Rights Reserved.
```

Sometimes corporate coding standards require us to write certain comments for legal reasons.

Good Comments

Explanation of Intent

```
public class PackageDistributionHelper {
    private PackageDistributionHelper() {
        // Added a private constructor to hide the implicit one
    }
```

Sometimes a comment goes beyond just useful information about the implementation and provides the intention behind a decision or a code structure.

Good Comments

Warning of Consequences

```
//
// This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implementation, v2.2.8-b130911.180
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the source schema.
//
```

Good Comments

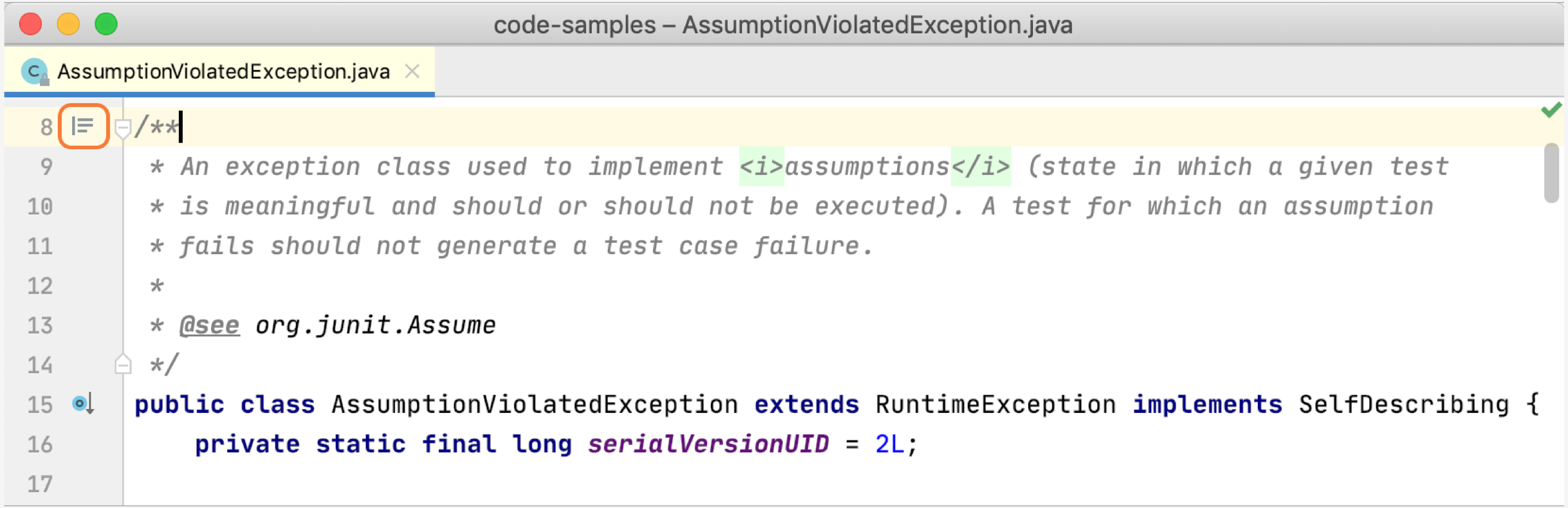
TODO Comments

```
6   public class TodoComments {
7   >     public List<String> getDetailsWOName() { return new ArrayList<String>(); }
10
11  >     public List<String> getNameWithoutName() { return new ArrayList<String>(); }
14        //TODO: Refactor to use update input objects
15  }
```

It is sometimes reasonable to leave “To do” notes in the form of //TODO comments.

Good Comments

Javadocs in Public APIs



Let's Refactor!

"Leave the campground cleaner than you found it."

ASK ME ANYTHING

THANK YOU!

Let’s connect

LinkedIn.com/in/EricMarcMartin

Facebook.com/EricMarcMartin