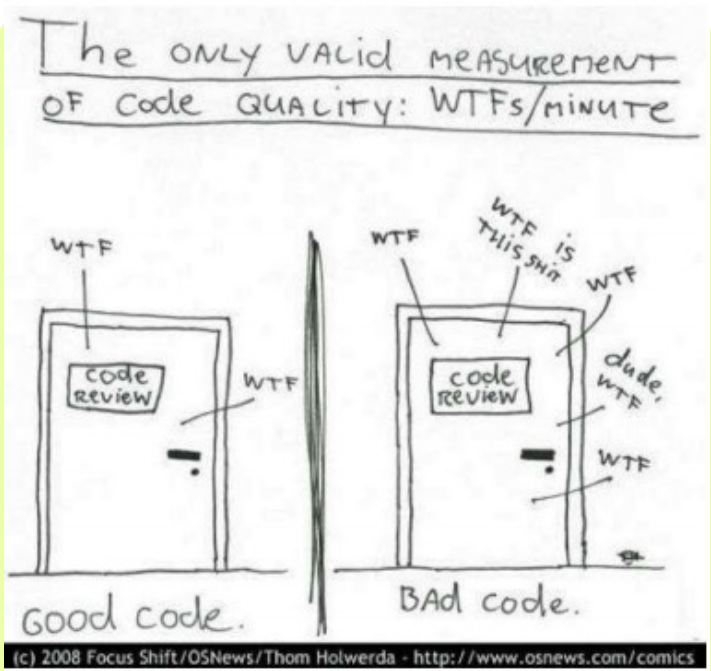
Clean Coding!

Krishnakripa Jayakumar



Which door represents your code?

Meaningful Names

1. Use Intention Revealing Names

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

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

2. Avoid Disinformation

Programmers must avoid leaving false clues that obscure the meaning of code.

- Do not refer to a grouping of accounts as an **accountList** unless it's actually a List
- If the container holding the accounts is not actually a List, it may lead to false conclusions.
- So **accountGroup** or **bunchOfAccounts** or just plain **accounts** would be better.

3. Make meaningful distinctions

- Number-series naming (a1, a2, .. aN) is the opposite of intentional naming.
- They provide no clue to the author's intention.

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

3. Make meaningful distinctions (Cont.)

- Having a **Product** class and having another class called **ProductInfo** or **ProductData**, brings a lot of confusion
- You have made the names different without making them mean anything different.

3. Make meaningful distinctions (Cont.)

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

How are the programmers in this project supposed to know which of these functions to call?

4. 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";
   /* ... */
};
```

What is happening here?

```
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;
}
```

5. Class Names

- Classes and objects should have noun or noun phrase names like **Customer**, **WikiPage**, **Account**, and **AddressParser**.
- A class name should not be a verb (eg: **Print**)

6. Method Names

- Methods should have verb or verb phrase names like **postPayment**, **deletePage**, or **save**.
- Accessors and mutators should be named for their value and prefixed with get, set.

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

6. Method Names (cont.)

- When constructors are overloaded, use static factory methods with names that describe the arguments. For example the following is better

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

than ...

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

7. Add meaningful context

- Imagine that you have variables named **firstName**, **lastName**, **street**, **houseNumber**, **city**, **state**, and **zipcode**.

Looking at all of these together, what do you think they mean?

7. Add meaningful context (cont.)

- You can add context by using prefixes: **addrFirstName**, **addrLastName**, **addrState**, and so on.
- However, a better solution is to create a class named **Address**.

Functions

1. Functions should be small

- Functions should not be 100 lines long.
- Functions should hardly ever be 20 lines long.

HtmlUtil.java (refactored)

```
public static String renderPageWithSetupsAndTeardowns(
  PageData pageData, boolean isSuite
) throws Exception {
  boolean isTestPage = pageData.hasAttribute("Test");
  if (isTestPage) {
    WikiPage testPage = pageData.getWikiPage();
    StringBuffer newPageContent = new StringBuffer();
    includeSetupPages(testPage, newPageContent, isSuite);
    newPageContent.append(pageData.getContent());
    includeTeardownPages(testPage, newPageContent, isSuite);
    pageData.setContent(newPageContent.toString());
  }

  return pageData.getHtml();
}
```

HtmlUtil.java (re-refactored)

```
public static String renderPageWithSetupsAndTeardowns(
  PageData pageData, boolean isSuite) throws Exception {
  if (isTestPage(pageData))
    includeSetupAndTeardownPages(pageData, isSuite);
  return pageData.getHtml();
}
```

2. Functions should do only 1 thing

- Functions should do one thing.
- They should do it well.
- They should do it only

- Let’s get back to the example…. (next slide)
- Is this function only doing one thing?

2. Functions should do only 1 thing (cont.)

HtmlUtil.java (re-refactored)

```
public static String renderPageWithSetupsAndTeardowns(
  PageData pageData, boolean isSuite) throws Exception {
  if (isTestPage(pageData))
    includeSetupAndTeardownPages(pageData, isSuite);
  return pageData.getHtml();
}
```

Looking at it, it is doing 3 things...

- 1. Determining whether the page is a test page.
- 2. If so, including setups and teardowns.
- 3. Rendering the page in HTML.

2. Functions should do only 1 thing (cont.)

- So is the function doing one thing or three things?
- The function is actually doing only 1 thing.
- Notice that the three steps of the function are one level of abstraction.

3. Use descriptive names

- Don’t be afraid to make a method name long. A long descriptive name is better than a short name.
- A long descriptive name is better than a long descriptive comment

4. Usage of function arguments

- When a function seems to need more than two or three arguments, it is likely that some of those arguments ought to be wrapped into a class of their own.

```
Circle makeCircle(double x, double y, double radius);
Circle makeCircle(Point center, double radius);
```

'checkPassword' method returns true if they match and false if anything goes wrong. But it is also doing more than 1 task. Can you spot it?

UserValidator.java

```
public class UserValidator {
  private Cryptographer cryptographer;

  public boolean checkPassword(String userName, String password) {
    User user = UserGateway.findByName(userName);
    if (user != User.NULL) {
      String codedPhrase = user.getPhraseEncodedByPassword();
      String phrase = cryptographer.decrypt(codedPhrase, password);
      if ("Valid Password".equals(phrase)) {
        Session.initialize();
        return true;
      }
    }
    return false;
  }
}
```

Error Handling

1. Prefer exceptions to returning error codes

This leads to deeply nested structures. When you return an error code, you create the problem that the caller must deal with the error immediately

```
public void process() {
    if(doSomething() == 0 )
    {
        if(doSomethingElse() == 100)
        {
            if(doSomethingElseAgain() == 200)
            {
                // etc.
            }
            else
            {
                // react to failure of doSomethingElseAgain
            }
        }
        else
        {
            // react to failure of doSomethingElse
        }
    }
    else
    {
        // react to failure of doSomething
    }
}
```

1. Prefer exceptions to returning error codes (c

On the other hand, if you use exceptions instead of returned error codes, then the error processing code can be separated from the happy path code and can be simplified.

```
public void process() {

    try
    {
        doSomething() ;
        doSomethingElse() ;
        doSomethingElseAgain() ;
    }
    catch(SomethingException e)
    {
        // react to failure of doSomething
    }
    catch(SomethingElseException e)
    {
        // react to failure of doSomethingElse
    }
    catch(SomethingElseAgainException e)
    {
        // react to failure of doSomethingElseAgain
    }

}
```

2. Don't return Null

- The problem with using null is the amount of error handling that needs to be put in place.

```
public void registerItem(Item item) {
  if (item != null) {
    ItemRegistry registry = peristentStore.getItemRegistry();
    if (registry != null) {
      Item existing = registry.getItem(item.getID());
      if (existing.getBillingPeriod().hasRetailOwner()) {
        existing.register(item);
      }
    }
  }
}
```

2. Don't return Null (cont.)

- In many cases, special case objects are an easy remedy. Imagine that you have code like this:

```
List<Employee> employees = getEmployees();
if (employees != null) {
  for(Employee e : employees) {
    totalPay += e.getPay();
  }
}
```

2. Don't return Null (cont.)

- If we change getEmployee so that it returns an empty list, we can clean up the code.
- Java has Collections.emptyList(), and it returns a predefined immutable list that we can use for this purpose:

```
public List<Employee> getEmployees() {
  if( .. there are no employees .. )
    return Collections.emptyList();
}
```

```
List<Employee> employees = getEmployees();
for(Employee e : employees) {
  totalPay += e.getPay();
}
```

3. Don't pass Null (cont.)

- What happens when you pass null in place of the first argument when you invoke the following function ?

```
public int calculatePoints(Point p1, Point p2) {
    return (p2.x - p1.x) / 2;
}
```

Comments

```
// Check to see if the employee is eligible for full benefits
if ((employee.flags & HOURLY_FLAG) &&
    (employee.age > 65))
```

```
if (employee.isEligibleForFullBenefits())
```

- It takes only a few seconds of thought to explain most of your intent in code. In many cases it’s simply a matter of creating a function that says the same thing as the comment you want to write.

1. Amplification

- A comment may be used to amplify the importance of something that may otherwise seem inconsequential.

```
String listItemContent = match.group(3).trim();
// the trim is real important.  It removes the starting
// spaces that could cause the item to be recognized
// as another list.
new ListItemWidget(this, listItemContent, this.level + 1);
return buildList(text.substring(match.end()));
```

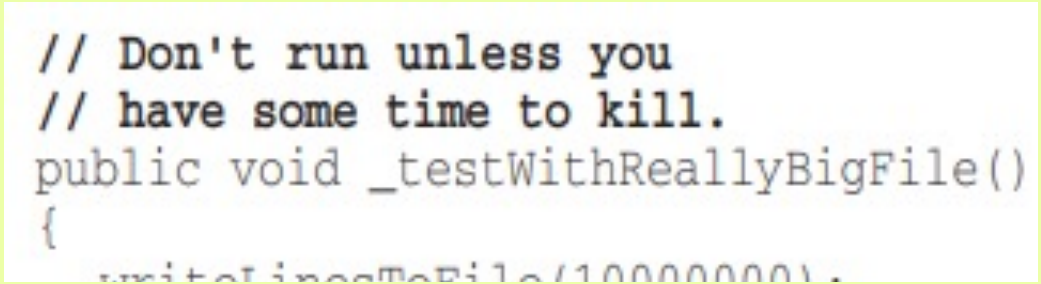
2. TODO Comments

- It is sometimes reasonable to leave “To do” notes in the form of //TODO comments. In the following case, the TODO comment explains why the function has a degenerate implementation.

```
//TODO-MdM these are not needed
// We expect this to go away when we do the checkout model
protected VersionInfo makeVersion() throws Exception
{
  return null;
}
```

3. Warning of consequences

- Sometimes it is useful to warn other programmers about certain consequences.



4. Bad Comments

```
/**
* @param title The title
* @param author The author
*/
```

```
/**
* Returns the day of the month
*
* @return the day of the month
*/
public int getDayOfMonth() {
  return dayOfMonth;
}
```

```
/**
* CHANGES
* ------------
* 02-Oct-2001   Reorganize process
* 15-Nov-2001  Added new Purchaser type
* 21-Nov-2001  Some customer objects arrive without city set
*/
```

4. Bad Comments

```
/* Added by Rick */
```

```
    InputStreamResponse response = new InputStreamResponse();
    response.setBody(formatter.getResultStream(), formatter.getByteCount());
//  InputStream resultsStream = formatter.getResultStream();
//  StreamReader reader = new StreamReader(resultsStream);
//  response.setContent(reader.read(formatter.getByteCount()));
```

Classes

1. Class Organization

- A class should begin with a list of variables (properties).
- Public static constants, if any, should come first.
- Then private static variables, followed by private instance variables.
- Public functions should follow the list of variables. We like to put the private utilities called by a public function right after the public function itself.

2. Single Responsibility Principle

- The Single Responsibility Principle (SRP) states that a class or module should have one, and only one, reason to change.

2. Single Responsibility Principle (cont.)

```
public class Customer
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public string CreditCardNumer { get; set; }
    public string CreditCardSecurityCode { get; set; }
    public DateTime CreditCardExpiryDate { get; set; }
    public bool CreditCardExpired
    {
        get { return CreditCardExpiryDate <
                     DateTime.Today; }
    }
}
```

2. Single Responsibility Principle (cont.)

```
public class Customer
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public CreditCard CreditCard { get; set; }
    public bool CreditCardExpired
    {
        get { return CreditCard.Expired; }
    }
}

public class CreditCard
{
    public string Number { get; set; }
    public string SecurityCode { get; set; }
    public DateTime ExpiryDate { get; set; }
    public CreditCard(string number, string securityCode, DateTime expiryDate)
    {
        //constructor code goes here...
    }
    public bool Expired
    {
        get { return ExpiryDate < DateTime.Today; }
    }
}
```

General Refactoring

1. Extract Method (Refactoring)

Before

```
public void Initialize()
{
    serviceUrl = ConfigurationManager.AppSettings["ServiceUrl"];
    int.TryParse(ConfigurationManager.AppSettings["PortNumber"], out portNumber);
    bool.TryParse(ConfigurationManager.AppSettings["ForceSSL"], out forceSSL);

    InitLogger();
}
```

1. Extract Method (Refactoring)

After

```
public void Initialize()
{
    LoadSettings();
    InitLogger();
}

private void LoadSettings()
{
    serviceUrl = ConfigurationManager.AppSettings["ServiceUrl"];
    int.TryParse(ConfigurationManager.AppSettings["PortNumber"], out portNumber);
    bool.TryParse(ConfigurationManager.AppSettings["ForceSSL"], out forceSSL);
}
```

2. Rename Methods (Refactoring)

Before

```
public DataSet GetDs(string connectionString, string sql)
{
      using (SqlConnection connection = new SqlConnection(connectionString))
      {
            connection.Open();
            SqlDataAdapter adapter = new SqlDataAdapter(new SqlCommand(sql, connection));
            DataSet dataSet = new DataSet();
            adapter.Fill(dataSet);
            return dataSet;
      }
}
```

2. Rename Methods (Refactoring)

After

```
public DataSet GetDataSet(string connectionString, string sql)
{
      using (SqlConnection connection = new SqlConnection(connectionString))
      {
          connection.Open();
          SqlDataAdapter adapter = new SqlDataAdapter(new SqlCommand(sql, connection));
          DataSet dataSet = new DataSet();
          adapter.Fill(dataSet);
          return dataSet;
      }
   }
```

3. Substitute Algorithms

Before

```
public string ConcatArrayItems(string[] inputArray)
{
    string returnedValue = "";
    for (int i = 0; i < inputArray.Length; i++)
    {
        returnedValue += inputArray[i];
        if (i < inputArray.Length - 1)
            returnedValue += ",";
    }
    return returnedValue;
}
```

3. Substitute Algorithms

After

```
public string ConcatArrayItems(string[] inputArray)
{
    return string.Join(",", inputArray);
}
```

4. Decompose conditionals

Before

```
public double CalculateTotalPayment(string code, float quantity, float unitPrice)
{
    if (code.StartsWith("LCD") || code.StartsWith("KB") || code.StartsWith("HDD")
                                                        || code.StartsWith("CPU"))
        return quantity * unitPrice * (1 + VAT - DISCOUNT_PERCENTAGE);
    else
        return quantity * unitPrice * (1 + VAT);
}
```

After

```
public double CalculateTotalPayment(string code, float quantity, float unitPrice)
{
    if (IsDiscountedCategory(code))
        return CalculatePaymentWithDiscount(quantity, unitPrice);
    else
        return CalculatePayment(quantity, unitPrice);
}
```

5. Consolidate Conditional Fragments

Before

```
public bool CandidateRejected(TestResult result)
{
    if (result.IQScore < 100)
        return true;
    if (result.EnglishScore < 2)
        return true;
    if (result.TechnicalScore < 3)
        return true;
    if (result.IQScore > 120 && (result.EnglishScore*2 + result.TechnicalScore)/3 >= 5)
        return false;
    return true;
}
```

After

```
public bool CandidateRejected(TestResult result)
{
    if (UnderLowerBound(result))
        return true;
    if (MeetsExpectation(result))
        return false;
    return true;
}
```

6. Inline temporary variable

Before

```
public void ShowWelcomeMessage(string firstName, string lastName)
{
    string fullName = firstName + " " + lastName;
    MessageBox.Show("Welcome back, " + fullName);
}
```

After

```
public void ShowWelcomeMessage(string firstName, string lastName)
{
    MessageBox.Show(string.Format("Welcome back, {0} {1}", firstName,
lastName));
}
```

7. Replace magic number with symbolic constant

Before

```
public bool ExportToExcel(TestResult[] results)
{
    if (results.Length > 65536)
        return false;
    //export code goes here
    //...
    return true;
}
```

After

```
private readonly int EXCEL_MAX_NUMBER_OF_ROWS = 65536;
public bool ExportToExcel(TestResult[] results)
{
    if (results.Length > EXCEL_MAX_NUMBER_OF_ROWS)
        return false;
    //export code goes here
    //...
    return true;
}
```

8. Remove Duplicated Code

```
public static XmlElement CreateAddressElement(XmlDocument xmlDocument, Address address)
{
    XmlElement element = xmlDocument.CreateElement("Address");
    XmlAttribute tempAttribute = null;
    tempAttribute = xmlDocument.CreateAttribute("Number");
    tempAttribute.Value = address.Number;
    element.SetAttributeNode(tempAttribute);
    tempAttribute = xmlDocument.CreateAttribute("Street");
    tempAttribute.Value = address.Street;
    element.SetAttributeNode(tempAttribute);
    tempAttribute = xmlDocument.CreateAttribute("City");
    tempAttribute.Value = address.City;
    element.SetAttributeNode(tempAttribute);
    tempAttribute = xmlDocument.CreateAttribute("Country");
    tempAttribute.Value = address.Country;
    element.SetAttributeNode(tempAttribute);
    return element;
}
```

8. Remove Duplicated Code

```
public XmlElement CreateAddressElement(XmlDocument xmlDocument, Address address)
{
    XmlElement element = xmlDocument.CreateElement("Address");
    element.SetAttributeNode(CreateAttribute(xmlDocument, "Number", address.Number));
    element.SetAttributeNode(CreateAttribute(xmlDocument, "Street", address.Street));
    element.SetAttributeNode(CreateAttribute(xmlDocument, "City", address.City));
    element.SetAttributeNode(CreateAttribute(xmlDocument, "Country", address.Country));
    return element;
}

private XmlAttribute CreateAttribute(XmlDocument xmlDocument, string name, string value)
{
    XmlAttribute attribute = xmlDocument.CreateAttribute(name);
    attribute.Value = value;
    return attribute;
}
```

Best Book to Read