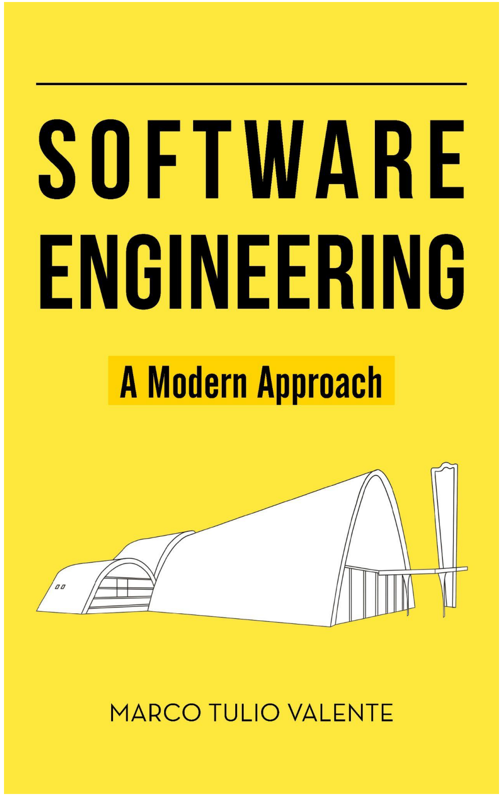


Chapter 9 - Refactoring

Prof. Marco Tulio Valente

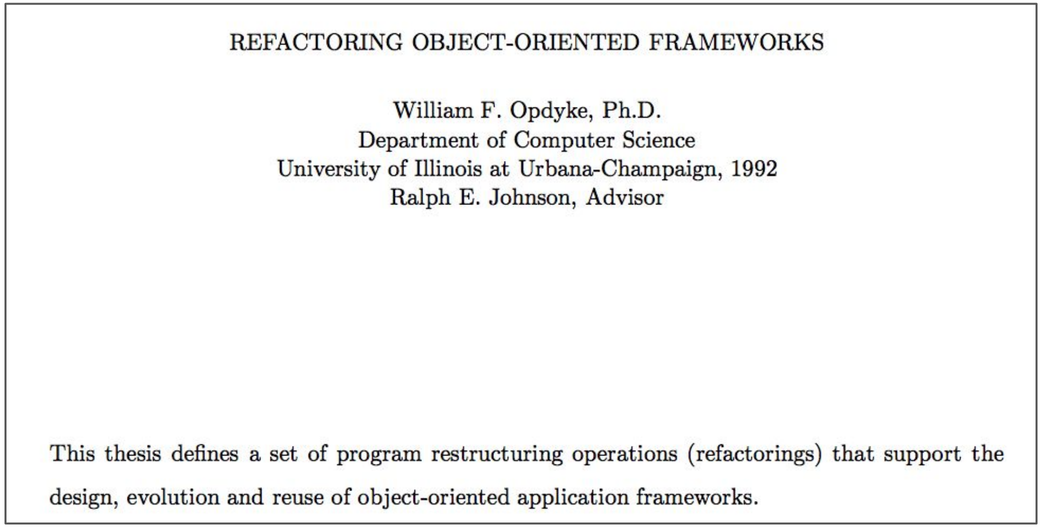
https://softengbook.org

Software Maintenance

- Preventive: bugs not yet reported
- Corrective: bugs reported by users
- Adaptive: customizations, new PL/OS versions, etc
- Evolutionary: new features
- Refactoring: code or design improvements

Refactoring

- Code transformations that improve maintainability without affecting external behavior

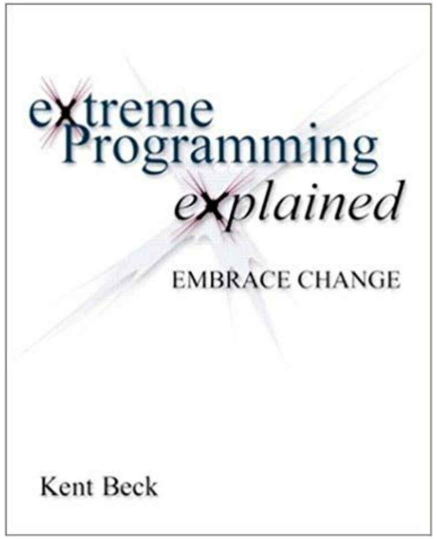


REFACTORING OBJECT-ORIENTED FRAMEWORKS

William F. Opdyke, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1992
Ralph E. Johnson, Advisor

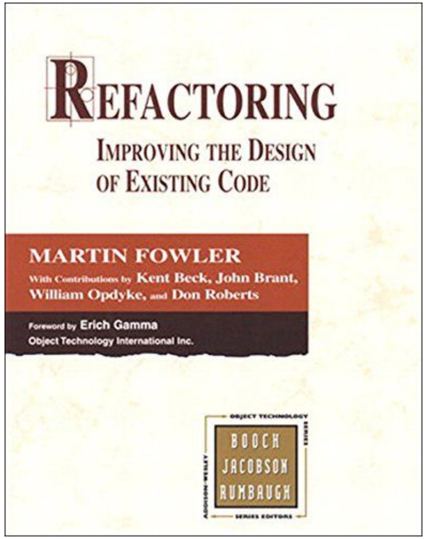
This thesis defines a set of program restructuring operations (refactorings) that support the design, evolution and reuse of object-oriented application frameworks.

Refactoring has become quite popular ...


extreme Programming explained

EMBRACE CHANGE

Kent Beck


REFACTORING

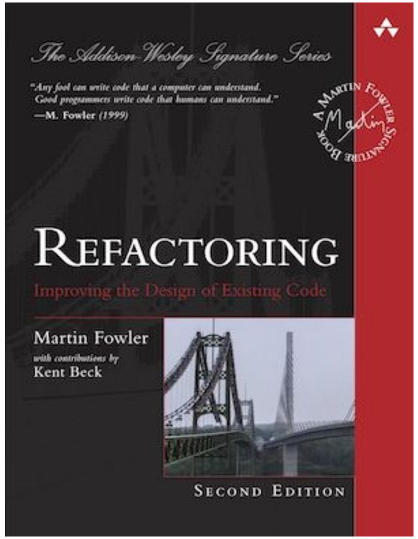
IMPROVING THE DESIGN OF EXISTING CODE

MARTIN FOWLER

With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts

Foreword by Erich Gamma
Object Technology International Inc.

OBJECT TECHNOLOGY SERIES
BOOCH JACOBSON RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY


The Addison-Wesley Signature Series

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
—M. Fowler (1999)

A MARTIN FOWLER SIGNATURE BOOK

REFACTORING

Improving the Design of Existing Code

Martin Fowler
with contributions by
Kent Beck

SECOND EDITION

1999

2000

2018

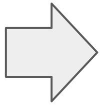
Catalog of Refactorings

- Extract Method
- Inline Method
- Move Method
- Extract Class
- Renaming
- etc

Method Extraction

Method Extraction

```
void f() {
  ... // A
  ... // B
  ... // C
}
```



```
 void g() {  // extracted method
  ... // B
}

void f () {
  ...  // A
  g();
  ...  // C
}
```

A real example ...

```
void onCreate(SQLiteDatabase database) {// before extraction
 // creates table 1
 database.execSQL("CREATE TABLE " +
          CELL_SIGNAL_TABLE + " (" + COLUMN_ID +
          " INTEGER PRIMARY KEY AUTOINCREMENT, " + ...
 database.execSQL("CREATE INDEX cellID_index ON " + ...);
 database.execSQL("CREATE INDEX cellID_timestamp ON " +...);

 // creates table 2
 String SMS_DATABASE_CREATE = "CREATE TABLE " +
          SILENT_SMS_TABLE + " (" + COLUMN_ID +
          " INTEGER PRIMARY KEY AUTOINCREMENT, " + ...
 database.execSQL(SMS_DATABASE_CREATE);
 String ZeroSMS = "INSERT INTO " + SILENT_SMS_TABLE +
          " (Address,Display,Class,ServiceCtr,Message) " +
          "VALUES ('"+ ...
 database.execSQL(ZeroSMS);

 // creates table 3
 String LOC_DATABASE_CREATE = "CREATE TABLE " +
          LOCATION_TABLE + " (" + COLUMN_ID +
          " INTEGER PRIMARY KEY AUTOINCREMENT, " + ...
 database.execSQL(LOC_DATABASE_CREATE);
 // more 200 lines, creating other tables
}
```

```
public void onCreate(SQLiteDatabase database) {
 createCellSignalTable(database);
 createSilentSmsTable(database);
 createLocationTable(database);
 createCellTable(database);
 createOpenCellIDTable(database);
 createDefaultMCCTable(database);
 createEventLogTable(database);
}
```

Reasons for Method Extraction

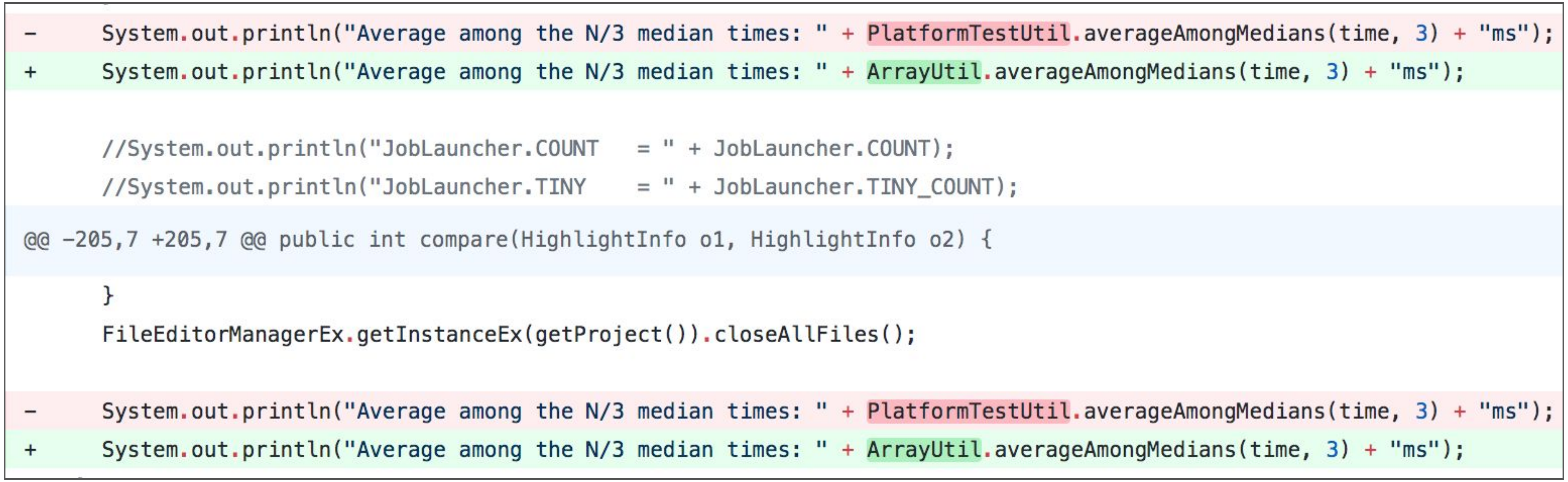
- Enable reuse of the extracted method
- Decompose large methods into smaller, focused ones
- Eliminate code duplication
- Improve testing by isolating functionality
- Support method overriding in subclasses
- Enable recursive implementations
- etc

Inline Method (opposite of extraction)

```
private void writeContentToFile(final byte[] revision) {
 getVirtualFile().setBinaryContent(revision);
}

private void write(byte[] revision) {
  VirtualFile virtualFile = getVirtualFile();
  ...
  if (document == null) {
     writeContentToFile(revision);
  }
   ...
 }
```

```
private void write(byte[] revision) {
  VirtualFile virtualFile = getVirtualFile();
  ...
  if (document == null) {
     virtualFile.setBinaryContent(revision); // after inline
  }
  ...
}
```

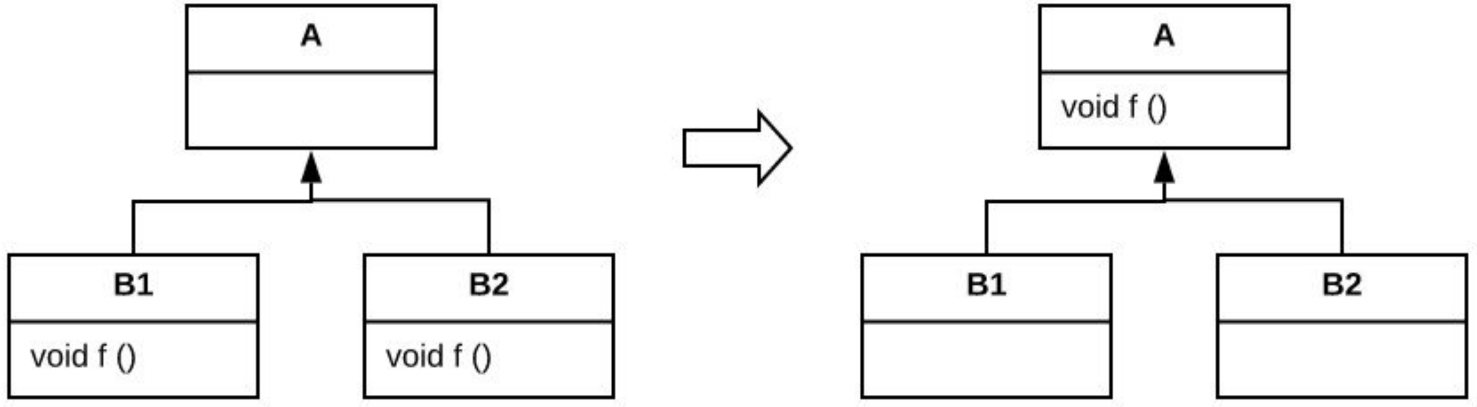
Move Method



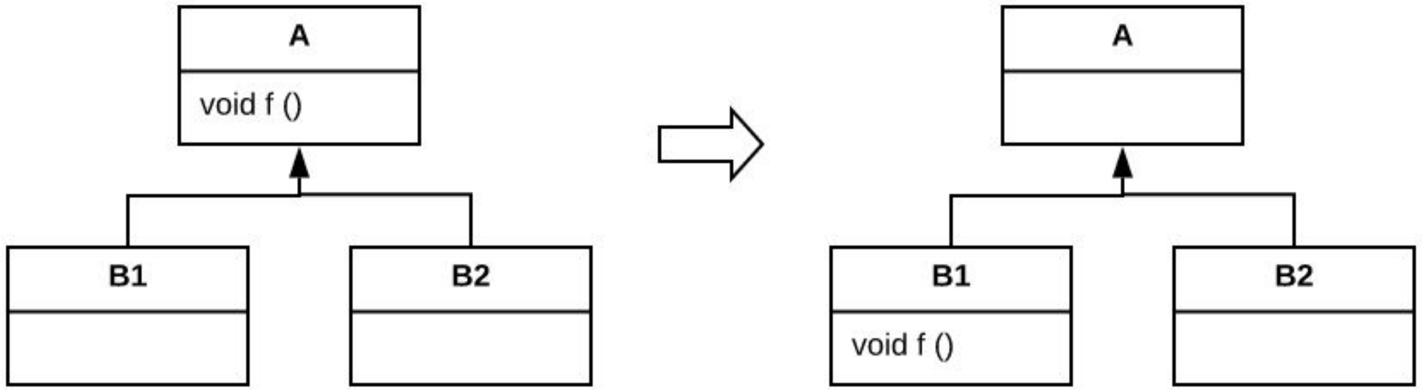
```diff
-      System.out.println("Average among the N/3 median times: " + PlatformTestUtil.averageAmongMedians(time, 3) + "ms");
+      System.out.println("Average among the N/3 median times: " + ArrayUtil.averageAmongMedians(time, 3) + "ms");

       //System.out.println("JobLauncher.COUNT   = " + JobLauncher.COUNT);
       //System.out.println("JobLauncher.TINY    = " + JobLauncher.TINY_COUNT);
@@ -205,7 +205,7 @@ public int compare(HighlightInfo o1, HighlightInfo o2) {
       }
       FileEditorManagerEx.getInstanceEx(getProject()).closeAllFiles();

-      System.out.println("Average among the N/3 median times: " + PlatformTestUtil.averageAmongMedians(time, 3) + "ms");
+      System.out.println("Average among the N/3 median times: " + ArrayUtil.averageAmongMedians(time, 3) + "ms");
```

Moving Methods Along Class Hierarchies

Pull Up Method



Push Down Method



Extract Class

```
class Person {
  String areaCode;
  String phone;
  String alternativeAreaCode;
  String alternativePhone;
  ...
}
```

```
class Phone { // extracted class
  String areaCode;
  String number;
}

class Person {
  Phone phone;
  Phone alternativePhone;
  ...
}
```

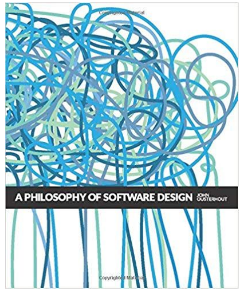
Renaming

(variable, parameter, method, class, exception, etc)

Choosing meaningful names for variables is one of the hardest problems in programming!

Refactoring Practice

Refactorings & Tests



Developers avoid refactoring without good test suites.

Instead, they try to minimize the number of code changes for each new feature or bug fix…

Which means that complexity accumulates and design mistakes don’t get corrected.

-- John Ousterhout

When should we refactor?

1. Opportunistic Refactorings
2. Planned Refactorings

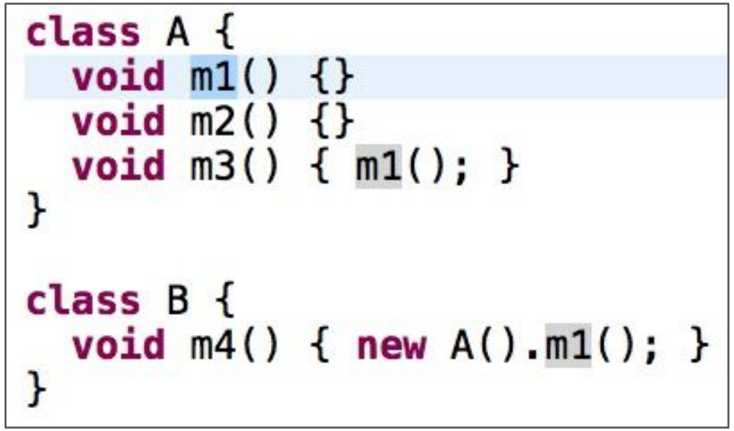
Opportunistic Refactorings

- Occurs in the midst of another development task
- The most common type of refactoring in practice

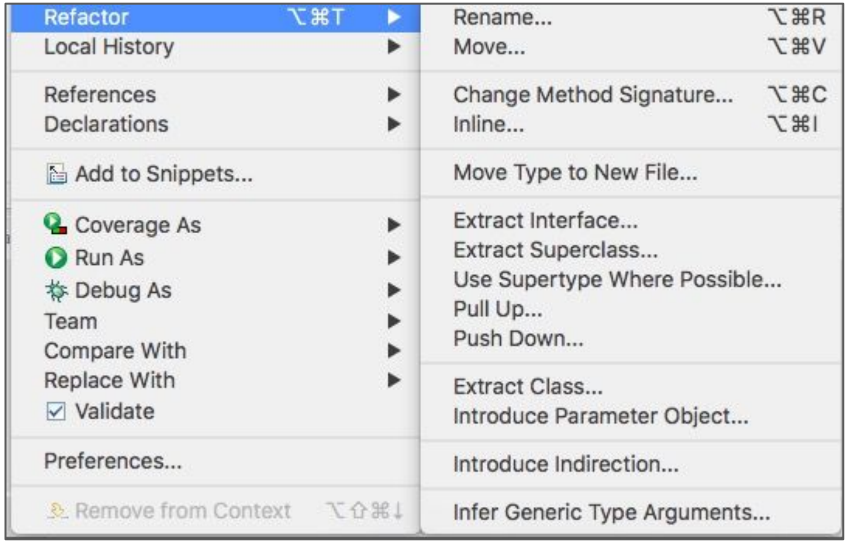
Planned (or Scheduled) Refactorings

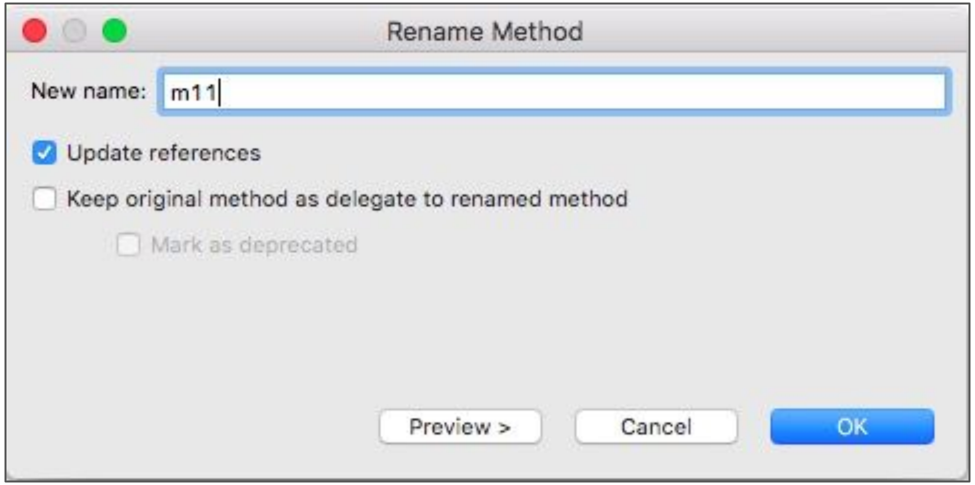
- Addresses correction of complex design problems
- Dedicated sessions focused solely on refactorings

IDE-Supported Refactoring



```
class A {
  void m1() {}
  void m2() {}
  void m3() { m1(); }
}

class B {
  void m4() { new A().m1(); }
}
```



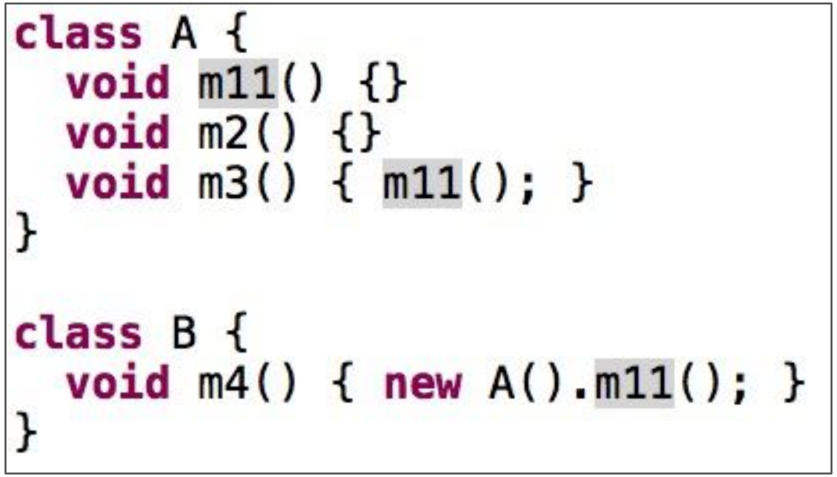


New name: m11

[✓] Update references

[ ] Keep original method as delegate to renamed method

[ ] Mark as deprecated



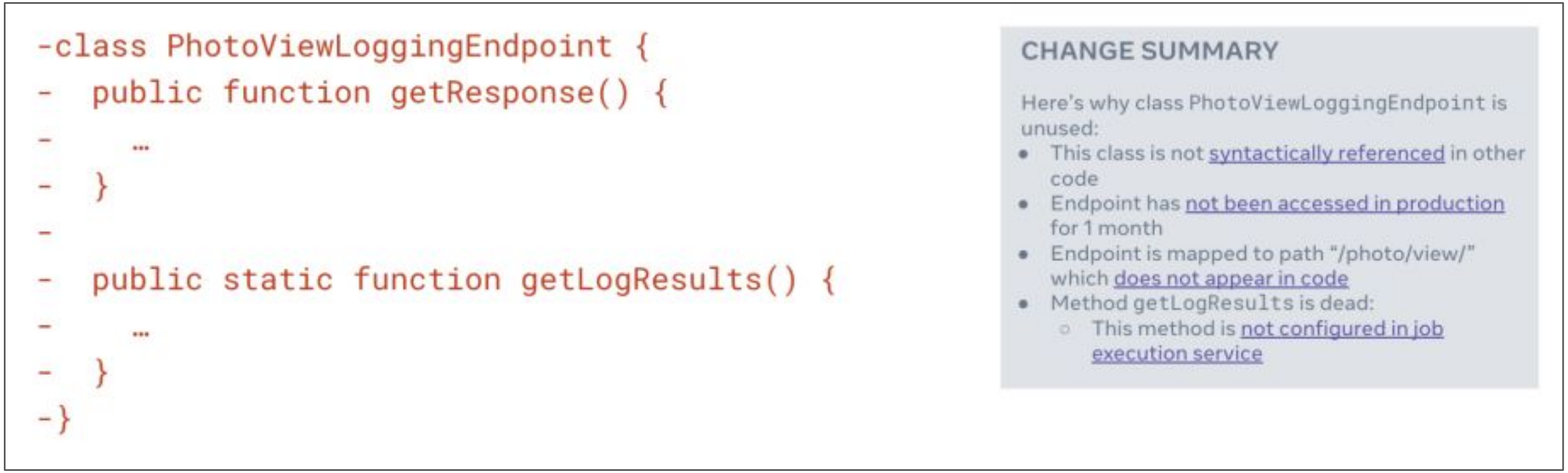
```
class A {
  void m11() {}
  void m2() {}
  void m3() { m11(); }
}

class B {
  void m4() { new A().m11(); }
}
```

Another Example: Remove Dead Code

- Code that is no longer being used is more common than we think…

Case Study: Meta/Facebook

- Meta has an internal tool to remove dead code



```
-class PhotoViewLoggingEndpoint {
-   public function getResponse() {
-       …
-   }
-
-   public static function getLogResults() {
-       …
-   }
-}
```

CHANGE SUMMARY

Here's why class PhotoViewLoggingEndpoint is unused:

- This class is not syntactically referenced in other code
- Endpoint has not been accessed in production for 1 month
- Endpoint is mapped to path "/photo/view/" which does not appear in code
- Method getLogResults is dead:
  - This method is not configured in job execution service

https://engineering.fb.com/2023/10/24/data-infrastructure/automating-dead-code-cleanup/

Usage Stats

- Tool was used to analyze hundreds of MLOC
- In 5 years, it helped to delete more than 100 MLOC, via 370K PR

"SCARF has grown to analyze hundreds of millions of lines of code; and five years on, it has automatically deleted more than 100 million lines of code in over 370,000 change requests."

Exercises

1. What is the relationship between the following sentence and the practice of refactoring?

"For each desired change, make the change easy (warning: this may be hard), then make the easy change."

-- Kent Beck

2. Give the names of refactorings A and B that, if executed in sequence, would not change the system's code.

These refactorings should be chosen so that refactoring B undoes the transformations made by A.

3. Normally, the application of a refactoring depends on certain preconditions. For example:

(a) When is it not possible to rename a local variable "a" to "b"?

(b) When is it not possible to move a method "f" from class A to class B?

4. (a) What code transformation was performed in this Java program? (b) Is it a refactoring? Justify.

```
class A {
  void f(){ print("hi");}
}

class B extends A {
  ...
}

class C {
  void f(){ print("hello");}
}

main() {
  B b = new B();
  b.f();
}
```

```
class A {
  void f(){ print("hi");}
}

class B extends A {
  void f(){ print("hello");}
}

class C {
  ...
}

main() {
  B b = new B();
  b.f();
}
```

5. (a) What code transformation was performed in this Java program? (b) Is it a refactoring? Justify.

package1/A.java

```
package package1;
public class A {
  void n() { (new B()).m("abc");
}
```

package1/B.java

```
package package1;
public class B {
  public void m(Object o) {…}
  void m(String s) {…}
}
```

⇨

package1/A.java

```
package package1;
public class A {
  void n() { (new B()).m("abc");
}
```

package2/B.java

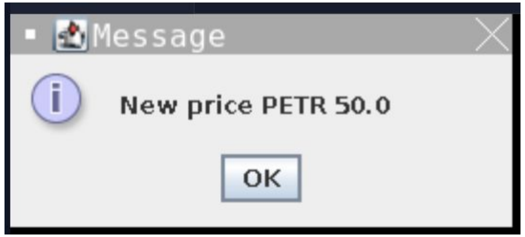
```
package package2;
public class B {
  public void m(Object o) {…}
  void m(String s) {…}
}
```

6. Is a change made to improve the performance of a system a refactoring?

Exercise: Software Design, Testability, and Refactoring

First, study the following code

```java
import static javax.swing.JOptionPane.showMessageDialog;

class Dashboard {
  private Stock stock;

  public Dashboard(Stock stock) {
    this.stock = stock;
  }

  public void alert() {
    showMessageDialog(null, "New price for " + stock.getName() + " " +
                      stock.getPrice());
  }
}
```

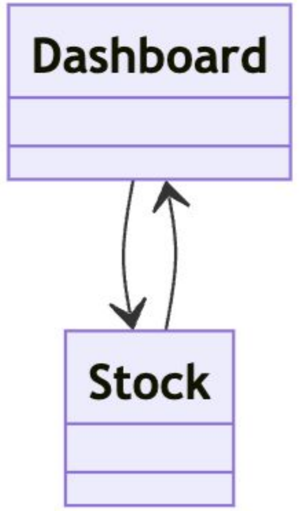


New price PETR 50.0

```
class Stock {
  private String name;
  private double price;
  private Dashboard dashboard;

  public Stock(String name, double price) {
    this.name = name;
    this.price = price;
  }

  public void setDashboard(Dashboard dashboard) {
    this.dashboard = dashboard;
  }

  public String getName() {
    return this.name;
  }

  public double getPrice() {
    return this.price;
  }
```

```java
// Continuation of the Stock class

  public void updatePrice(double price) {
    this.price = price;
    dashboard.alert();
  }

}


public class Main {
  public static void main(String[] args) {
    Stock stock = new Stock("PETR", 40);
    Dashboard dashboard = new Dashboard(stock);
    stock.setDashboard(dashboard);
    stock.updatePrice(50);
  }
}
```

- In the previous code, there is a circular dependency between Dashboard and Stock
- Circular dependencies are indicators of design and testability issues
- For example, why is it difficult to write a unit test for the Stock.updatePrice() method?

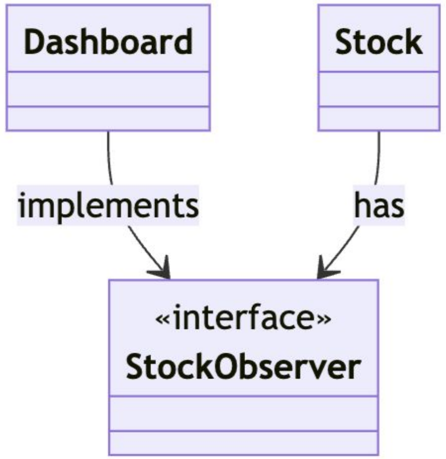


Exercises

1. Refactor the code to remove the circular dependency between the classes.
2. Why is it now easier to write a unit test for updatePrice()?

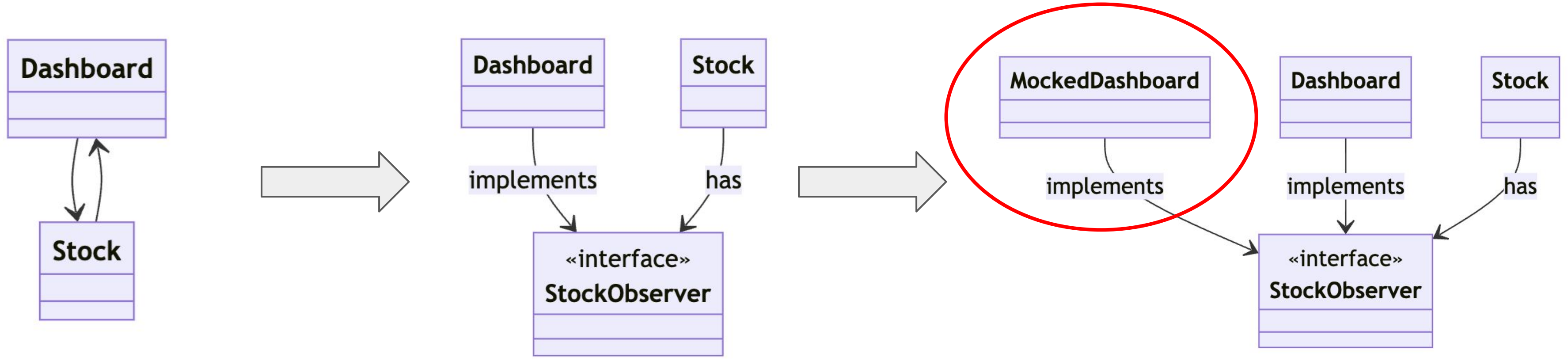
Answer

```
interface StockObserver {
  public void alert();
}

class Dashboard implements StockObserver {
  ...
  public void alert() {
    showMessageDialog(null, ...);
  }
}

class Stock { ...
  private StockObserver observer;

  public void setObserver(StockObserver observer)
{
    this.observer = observer;
  }
  ...
  public void updatePrice(double price) {
    this.price = price;
    observer.alert();
  }
```



```
public class Main {
  public static void main(String[] args) {
    Stock stock = new Stock("PETR", 40);
    Dashboard dashboard =  new Dashboard(stock);
    stock.setObserver(dashboard);
    stock.updatePrice(50);
  }
}
```

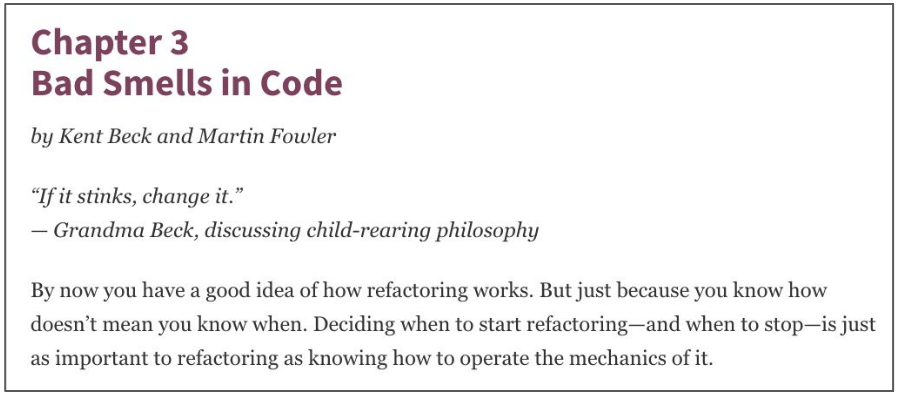
Summarizing



Code Smells

Code (or Bad) Smells

- Indicators of low-quality code
- Code that is hard to maintain, understand, modify or test
- Therefore, it is a candidate for refactoring



Chapter 3
Bad Smells in Code

by Kent Beck and Martin Fowler

"If it stinks, change it."
— Grandma Beck, discussing child-rearing philosophy

By now you have a good idea of how refactoring works. But just because you know how doesn't mean you know when. Deciding when to start refactoring—and when to stop—is just as important to refactoring as knowing how to operate the mechanics of it.

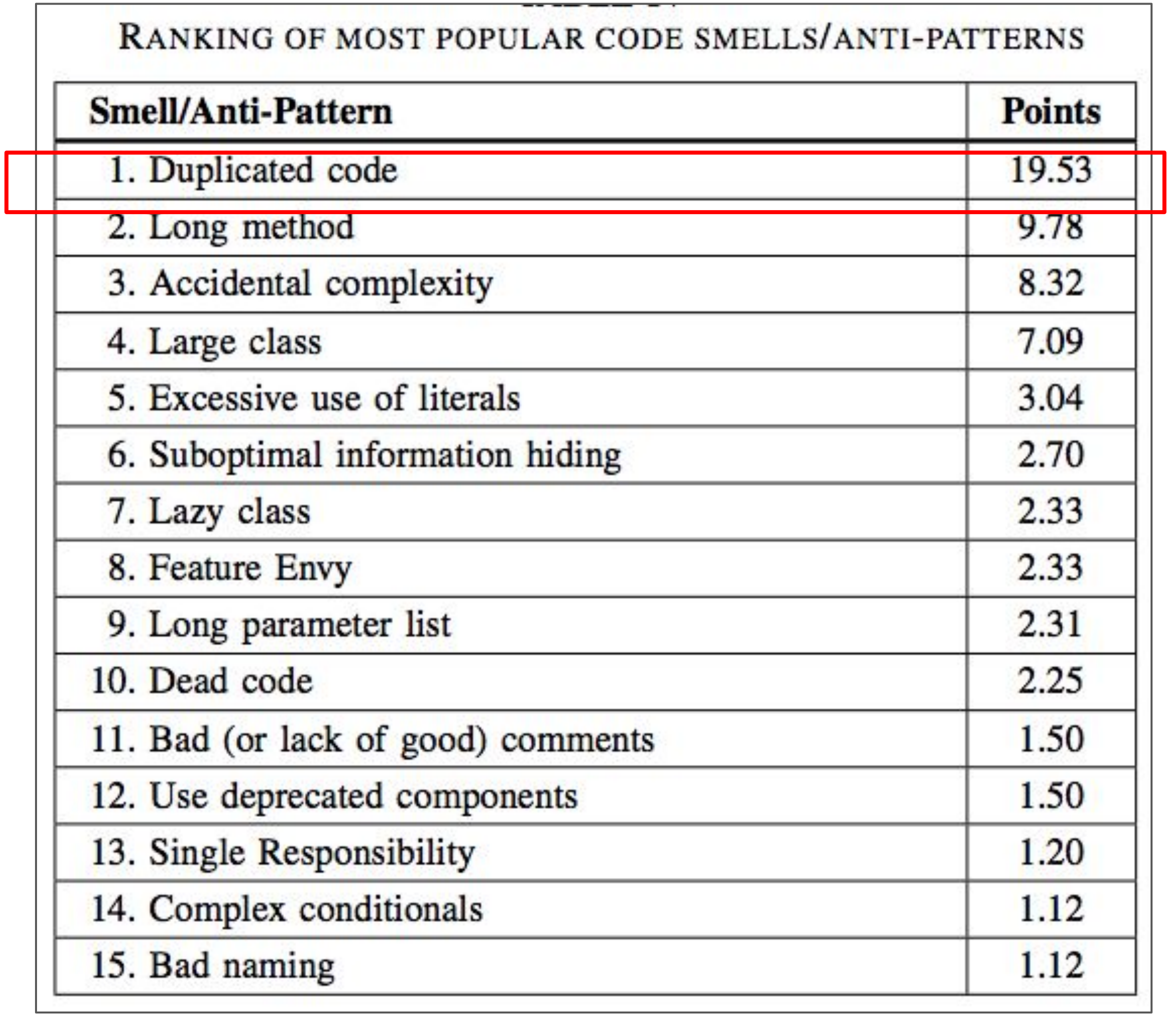
Catalog of Code Smells

- Duplicated Code
- Long Methods
- Large Classes
- Feature Envy
- Long Parameter List
- Global Variables
- Primitive Obsession
- Mutable Objects
- Data Classes
- Comments

Duplicated Code

Duplicated Code

- Makes maintenance more difficult
- Therefore, it is a candidate for refactoring



RANKING OF MOST POPULAR CODE SMELLS/ANTI-PATTERNS

Smell/Anti-Pattern	Points
1. Duplicated code	19.53
2. Long method	9.78
3. Accidental complexity	8.32
4. Large class	7.09
5. Excessive use of literals	3.04
6. Suboptimal information hiding	2.70
7. Lazy class	2.33
8. Feature Envy	2.33
9. Long parameter list	2.31
10. Dead code	2.25
11. Bad (or lack of good) comments	1.50
12. Use deprecated components	1.50
13. Single Responsibility	1.20
14. Complex conditionals	1.12
15. Bad naming	1.12

Aiko Yamashita, Leon Moonen. Do developers care about code smells? An exploratory survey. WCRE 2013.

Duplicated Code ⇒ Clones

Clone Type 1 (comments and spaces)

```
int factorial(int n) {
  fat = 1;
  for (i = 1; i <= n; i++)
    fat = fat * i;
  return fat;
}
```

Original code

```
int factorial(int n) {
  fat=1;
  for (i=1; i<=n; i++)
    fat=fat*i;
  return fat; // returns factorial
}
```

Clone Type 2 (type 1 + different names)

```
int factorial(int n) {
  fat = 1;
  for (i = 1; i <= n; i++)
    fat = fat * i;
  return fat;
}
```

Original code

```
int factorial(int n) {
  f = 1;
  for (j = 1; j <= n; j++)
    f = f * j;
  return f;
}
```

Clone Type 3 (type 2 + changes in commands)

```
int factorial(int n) {
  fat = 1;
  for (i = 1; i <= n; i++)
    fat = fat * i;
  return fat;
}
```

Original code

```
int factorial(int n) {
  fat = 1;
  for (j = 1; j <= n; j++)
    fat = fat * j;
  System.out.println(fat); // new command
  return fat;
}
```

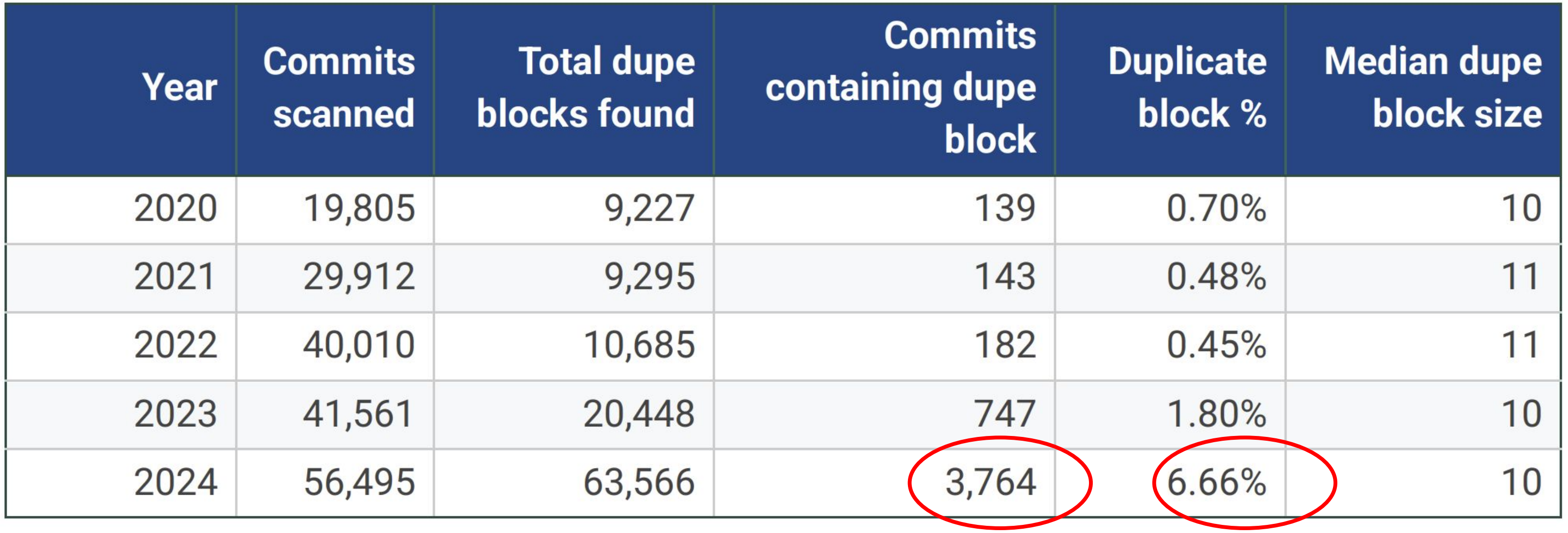
Clone Type 4 (equivalent algorithms)

```
int factorial(int n) {
  fat = 1;
  for (i = 1; i <= n; i++)
    fat = fat * i;
  return fat;
}
```

Original code

```
int factorial(int n) {
  if (n == 0)
     return 1;
  else return n*factorial(n-1);
}
```

https://www.gitclear.com/ai_assistant_code_quality_2025_research

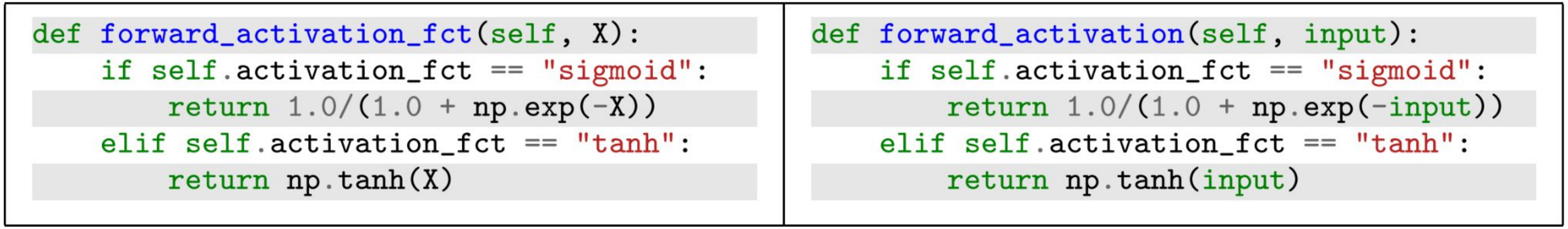


Year	Commits scanned	Total dupe blocks found	Commits containing dupe block	Duplicate block %	Median dupe block size
2020	19,805	9,227	139	0.70%	10
2021	29,912	9,295	143	0.48%	11
2022	40,010	10,685	182	0.45%	11
2023	41,561	20,448	747	1.80%	10
2024	56,495	63,566	3,764	6.66%	10

Why do you think the number of clones is increasing?

Exercise: What is the type of the following clones?

(a)



```
def forward_activation_fct(self, X):
    if self.activation_fct == "sigmoid":
        return 1.0/(1.0 + np.exp(-X))
    elif self.activation_fct == "tanh":
        return np.tanh(X)
```

```
def forward_activation(self, input):
    if self.activation_fct == "sigmoid":
        return 1.0/(1.0 + np.exp(-input))
    elif self.activation_fct == "tanh":
        return np.tanh(input)
```

Source: https://arxiv.org/abs/2107.13614 (including next slides)

(b)

```
def forward_activation(self, X):
    #compute post activation value of X
    if self.activation_fct == "sigmoid":
        return 1.0/(1.0 + np.exp(-X))
    elif self.activation_fct == "tanh":
        return np.tanh(X)
```

```
def forward_activation(self, X):
    if self.activation_fct == "sigmoid":
        return 1.0/(1.0 + np.exp(-X))
    elif self.activation_fct == "tanh":
        return np.tanh(X)
```

(c)

```python
def forward_activation(self, X):
    if self.activation_fct == "sigmoid":
        return 1.0/(1.0 + np.exp(-X))
    elif self.activation_fct == "tanh":
        return np.tanh(X)
```

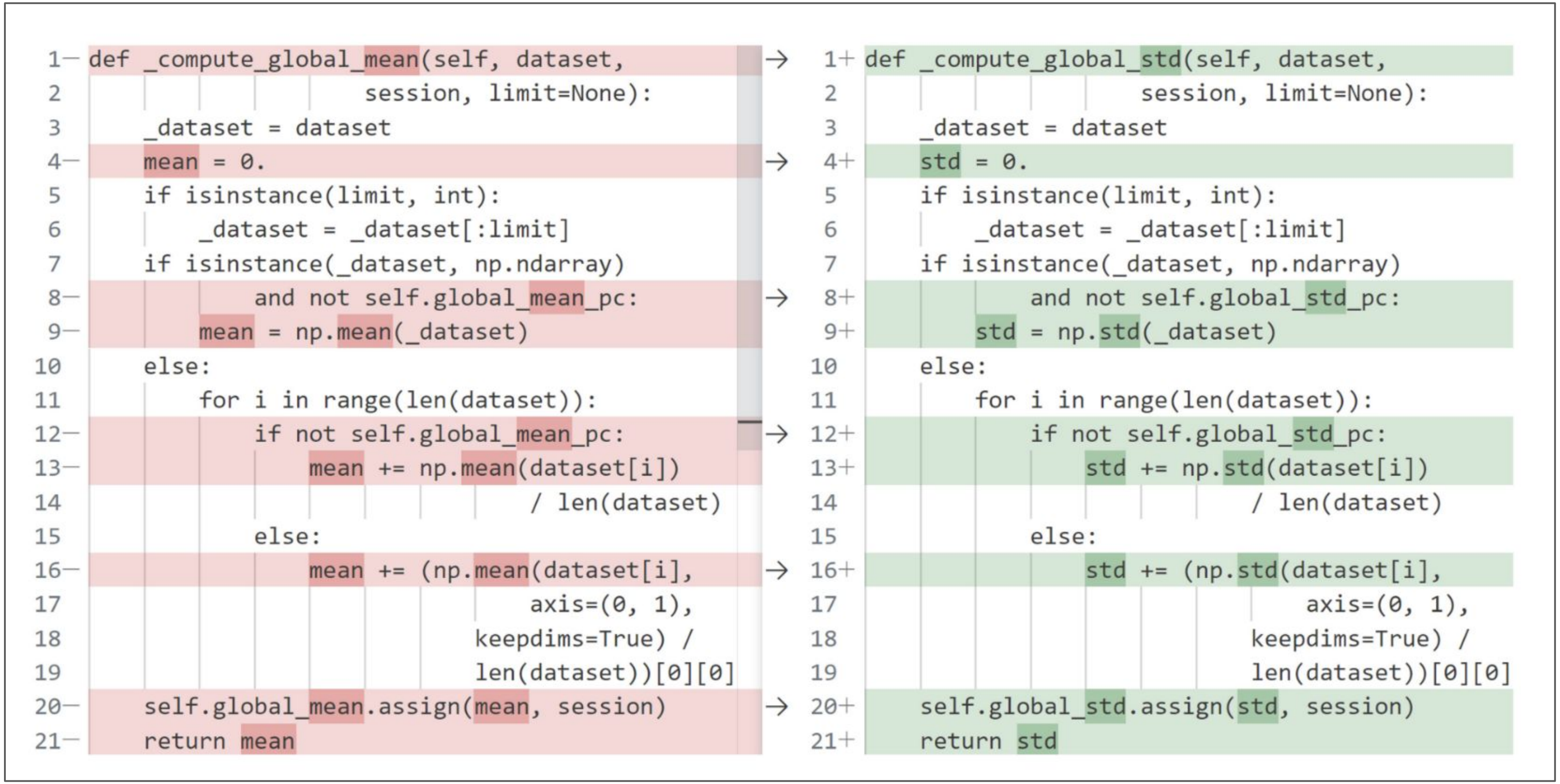
```python
def forward_activation(self, x):
    vals = { "sigmoid" : 1.0/(1.0+np.exp(-x)),
             "tanh" : np.tanh(x) }
    return vals[self.activation_fct]
```

(d)

```python
def forward_activation_fct(self, X):
    if self.activation_fct == "sigmoid":
        return 1.0/(1.0 + np.exp(-X))
    elif self.activation_fct == "tanh":
        return np.tanh(X)
```

```python
def forward_activation(self, x):
    if self.activation_fct == "sigmoid":
        return 1.0/(1.0 + np.exp(-x))
    elif self.activation_fct == "tanh":
        return np.tanh(x)
    elif self.activation_fct == "relu":
        return np.maximum(0,x)
```

(e)



```python
def _compute_global_mean(self, dataset,
                    session, limit=None):
    _dataset = dataset
    mean = 0.
    if isinstance(limit, int):
        _dataset = _dataset[:limit]
    if isinstance(_dataset, np.ndarray)
            and not self.global_mean_pc:
        mean = np.mean(_dataset)
    else:
        for i in range(len(dataset)):
            if not self.global_mean_pc:
                mean += np.mean(dataset[i])
                                / len(dataset)
            else:
                mean += (np.mean(dataset[i],
                                axis=(0, 1),
                            keepdims=True) /
                            len(dataset))[0][0]
    self.global_mean.assign(mean, session)
    return mean
```

```python
def _compute_global_std(self, dataset,
                    session, limit=None):
    _dataset = dataset
    std = 0.
    if isinstance(limit, int):
        _dataset = _dataset[:limit]
    if isinstance(_dataset, np.ndarray)
            and not self.global_std_pc:
        std = np.std(_dataset)
    else:
        for i in range(len(dataset)):
            if not self.global_std_pc:
                std += np.std(dataset[i])
                                / len(dataset)
            else:
                std += (np.std(dataset[i],
                                axis=(0, 1),
                            keepdims=True) /
                            len(dataset))[0][0]
    self.global_std.assign(std, session)
    return std
```

Additional question: is it worth eliminating this clone?

Source: https://dl.acm.org/doi/10.1145/3607181

Don't DRY Your Code Prematurely

https://testing.googleblog.com/2024/05/dont-dry-your-code-prematurely.html

DRY = Don’t Repeat Yourself

While functions may look the same, they may also serve different requirements that evolve differently over time.

"Tolerable" duplication ⇒ different entities

```
# Repetitive but allows for clear, entity-specific
# logic and future changes.
```

✅
```
def set_task_deadline(task_deadline):
  if task_deadline <= datetime.now():
    raise ValueError(“Date must be in the future”)
```

✅
```
def set_payment_deadline(payment_deadline):
  if payment_deadline <= datetime.now():
    raise ValueError(“Date must be in the future”)
```

Feature Envy

Feature Envy

- Method that "envies" data and methods of another class
- Uses more methods and data from another class
- Therefore, it is a candidate for moving to that class

```
public class DrawingEditorProxy
                extends AbstractBean implements DrawingEditor {
  ...
  void fireAreaInvalidated2 (AbstractTool abt , Double r ){
    Point p1 = abt.getView().drawingToView (...);
    Point p2 = abt.getView().drawingToView (...);
    Rectangle r=new Rectangle(p1.x,p1.y,p2.x-p1.x p2.y-p1.y);
    abt.fireAreaInvalidated (r);
  }
  ...
}
```

Global Variables

Global Variables

- Poor coupling: global variables make understanding a method more difficult

```
void f(...) {
  // computes a certain value x
  return x + g; // where g is a global variable
}
```

To understand what f returns, we need to know the value of g
This value may vary between calls to f

Primitive Obsession

Primitive Obsession

- Using primitive types for zip code, currency, date, time, color, email, etc
- These values should have their own type with methods
- For example, methods for validation

Mutable Objects

Mutable vs Immutable Objects

- Mutable: state can change
- Immutable: an object whose state does not change after creation

Exercise: (a) What will be printed by the following Java program? Justify. (b) Are Strings in Java immutable or not?

```
class Main {
  public static void main(String[] args) {
    String s1 = "Hello";
    String s2 = s1.toUpperCase();
    System.out.println(s1);
    System.out.println(s2);
  }
}
```

Exercise: (a) What will be printed by the following C++ program? (b) Are Strings in C++ immutable or not?

```
#include <iostream>
#include <string>

int main() {
  std::string s = "ball";
  s[0] = 'c';
  std::cout << s;
}
```

Why are immutable objects "good"?

- They provide more "security" to the object creator
  - You can pass the object to other methods and be sure that they will not change its state
- They are not subject to race conditions or other concurrency issues
  - No need for synchronizations, locks, mutex, etc

How to interpret this code smell

- Whenever possible:
    - Create immutable objects
    - Especially, for simple objects (ZIP, Date, Time, etc)
- On the other hand, in imperative languages it is natural to have some mutable objects

Comments

Don't comment bad code, rewrite it

-- B. Kerninghan & P. J. Plauger

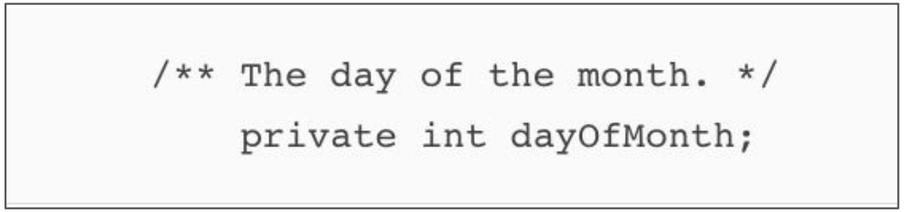
```
void f() {
  // task1
  ...
  // task2
  ...
  // taskn
  ...
}
```

```
void f() {
  // task1
  ...
  // task2
  ...
  // taskn
  ...
}
```

```
void task1 { ... }
void task2 { ... }
void taskn { ... }

void f {
  task1();
  task2();
  ...
  taskn();
}
```

"Noise" Comments (add nothing ...)

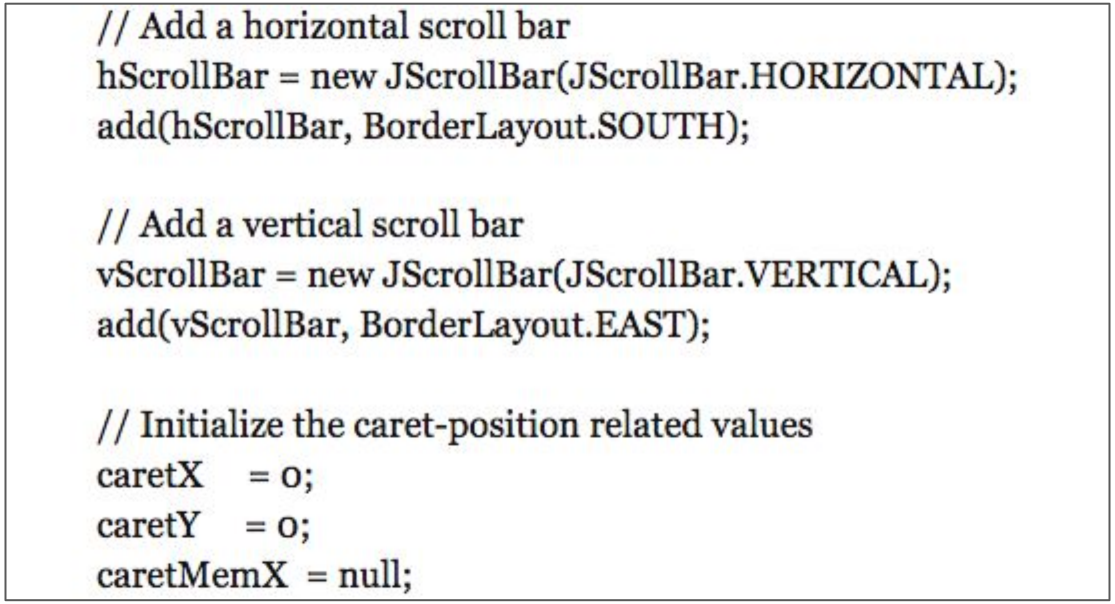
Example #1



```
/** The day of the month. */
    private int dayOfMonth;
```

```
// this function sends an email
void sendEmail() {
  ...
}

// this class holds data for an employee
public class Employee {
  ...
}
```

Example #2



```
// Add a horizontal scroll bar
hScrollBar = new JScrollBar(JScrollBar.HORIZONTAL);
add(hScrollBar, BorderLayout.SOUTH);

// Add a vertical scroll bar
vScrollBar = new JScrollBar(JScrollBar.VERTICAL);
add(vScrollBar, BorderLayout.EAST);

// Initialize the caret-position related values
caretX    = 0;
caretY    = 0;
caretMemX  = null;
```

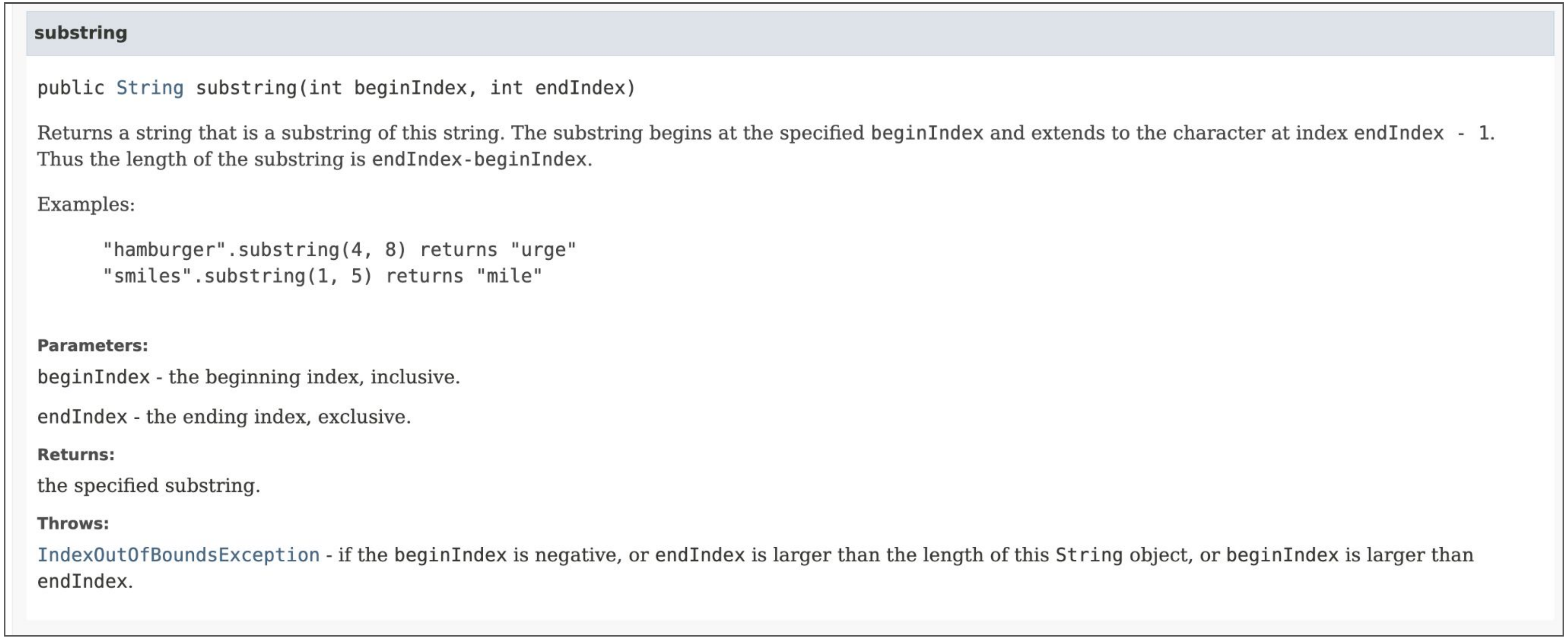
Comments that merely repeat what is already clear in the code

Source: A Philosophy of Software Design (chapter 13)

However, not every comment is a code smell

```
/**
* Returns a string that is a substring of this string. The
* substring begins at the specified {@code beginIndex} and
* extends to the character at index {@code endIndex - 1}.
* Thus the length of the substring is  {@code endIndex-beginIndex}.
* <p>
* Examples:
* <blockquote><pre>
* "hamburger".substring(4, 8) returns "urge"
* "smiles".substring(1, 5) returns "mile"
* </pre></blockquote>
*
* @param       beginIndex   the beginning index, inclusive.
* @param       endIndex     the ending index, exclusive.
* @return      the specified substring.
* @throws      IndexOutOfBoundsException  if the
*             {@code beginIndex} is negative, or
*             {@code endIndex} is larger than the length of
*             this {@code String} object, or
*             {@code beginIndex} is larger than
*             {@code endIndex}.
*/
public String substring(int beginIndex, int endIndex) { ... }
```

```
> javadoc -d docs String.java
```



substring

```
public String substring(int beginIndex, int endIndex)
```

Returns a string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex-beginIndex`.

Examples:

```
      "hamburger".substring(4, 8) returns "urge"
      "smiles".substring(1, 5) returns "mile"
```

Parameters:

`beginIndex` - the beginning index, inclusive.

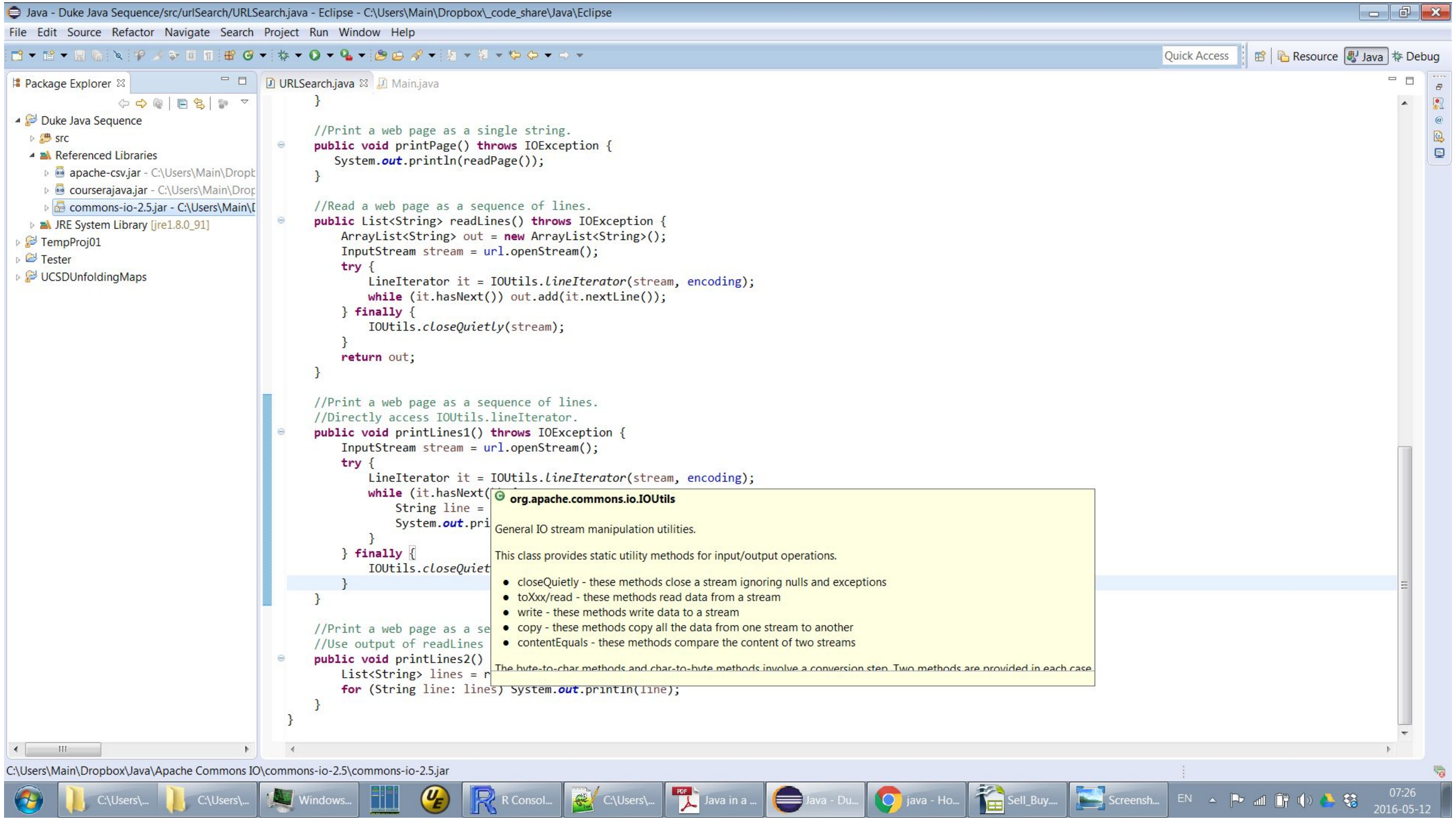
`endIndex` - the ending index, exclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.


Java - Duke Java Sequence/src/urlSearch/URLSearch.java - Eclipse - C:\Users\Main\Dropbox_code_share\Java\Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Resource
Java
Debug
Package Explorer
Duke Java Sequence
src
Referenced Libraries
apache-csv.jar - C:\Users\Main\Dropb
courserajava.jar - C:\Users\Main\Drop
commons-io-2.5.jar - C:\Users\Main\[
JRE System Library [jre1.8.0_91]
TempProj01
Tester
UCSDUnfoldingMaps
URLSearch.java
Main.java
EN
07:26
2016-05-12

```
    }

    //Print a web page as a single string.
    public void printPage() throws IOException {
       System.out.println(readPage());
    }

    //Read a web page as a sequence of lines.
    public List<String> readLines() throws IOException {
        ArrayList<String> out = new ArrayList<String>();
        InputStream stream = url.openStream();
        try {
            LineIterator it = IOUtils.lineIterator(stream, encoding);
            while (it.hasNext()) out.add(it.nextLine());
        } finally {
            IOUtils.closeQuietly(stream);
        }
        return out;
    }

    //Print a web page as a sequence of lines.
    //Directly access IOUtils.lineIterator.
    public void printLines1() throws IOException {
        InputStream stream = url.openStream();
        try {
            LineIterator it = IOUtils.lineIterator(stream, encoding);
            while (it.hasNext(
                String line =
                System.out.pri
            }
        } finally {
            IOUtils.closeQuiet
        }
    }

    //Print a web page as a se
    //Use output of readLines
    public void printLines2()
        List<String> lines = r
        for (String line: lines) System.out.println(line);
    }
}
```

org.apache.commons.io.IOUtils

General IO stream manipulation utilities.

This class provides static utility methods for input/output operations.

- closeQuietly - these methods close a stream ignoring nulls and exceptions
- toXxx/read - these methods read data from a stream
- write - these methods write data to a stream
- copy - these methods copy all the data from one stream to another
- contentEquals - these methods compare the content of two streams

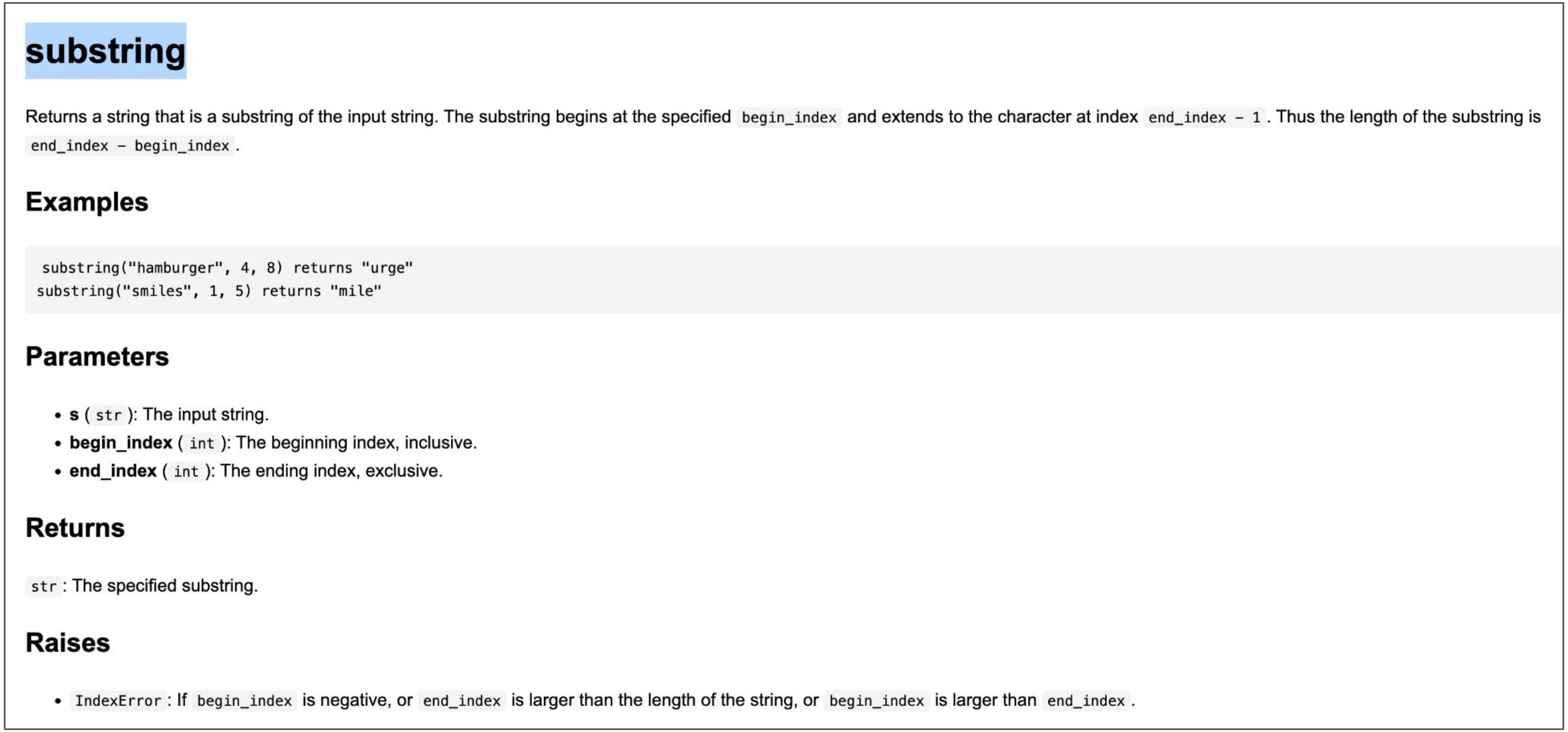
The byte-to-char methods and char-to-byte methods involve a conversion step. Two methods are provided in each case

C:\Users\Main\Dropbox\Java\Apache Commons IO\commons-io-2.5\commons-io-2.5.jar

Same example, now in Python (using docstrings)

```
def substring(s: str, begin_index: int, end_index: int) -> str:
    """
    Returns a string that is a substring of the input string. The
    substring begins at the specified `begin_index` and
    extends to the character at index `end_index - 1`.
    Thus the length of the substring is `end_index - begin_index`.

    Examples:
        substring("hamburger", 4, 8) returns "urge"
        substring("smiles", 1, 5) returns "mile"

    Parameters:
        s (str): The input string.
        begin_index (int): The beginning index, inclusive.
        end_index (int): The ending index, exclusive.

    Returns:
        str: The specified substring.

    Raises:
        IndexError: If `begin_index` is negative, or
                    `end_index` is larger than the length of the string, or
                    `begin_index` is larger than `end_index`.
   """
```

Docstrings: documentation strings after the definition of functions, classes, or modules. Thus, they are not comments.



substring

Returns a string that is a substring of the input string. The substring begins at the specified `begin_index` and extends to the character at index `end_index - 1`. Thus the length of the substring is `end_index - begin_index`.

Examples

```
 substring("hamburger", 4, 8) returns "urge"
substring("smiles", 1, 5) returns "mile"
```

Parameters

- **s** (`str`): The input string.
- **begin_index** (`int`): The beginning index, inclusive.
- **end_index** (`int`): The ending index, exclusive.

Returns

`str` : The specified substring.

Raises

- `IndexError` : If `begin_index` is negative, or `end_index` is larger than the length of the string, or `begin_index` is larger than `end_index`.

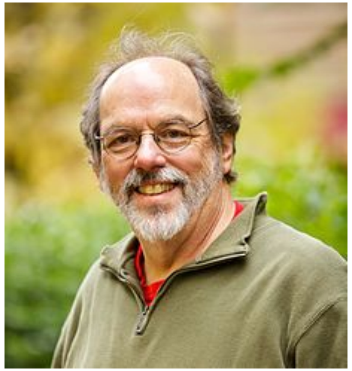
Another example: documenting inherently complex code

```
// format matched kk:mm:ss EEE, MMM dd, yyy
Pattern timePattern = Pattern.compile("\\d*:\\d*:\\d* \\w*, \\w*, \\d*, \\d*");
```

Final Topic: Technical Debt

Technical Debt

- A Metaphor to explain the importance of SE practices
- Proposed by Ward Cunningham (1992)
- Designates non-optimal design solutions that make maintenance and evolution difficult



Examples of Technical Debt

- Lack of tests
- Non-compliance with architectural patterns
- High coupling and low cohesion
- Code smells
- Lack of documentation
- Inconsistent code formatting
- etc

Exercises

1. Consider the following Price class. One advantage is that it may have a method (not shown) to convert the value to other currencies. (a) Why are objects of this class mutable? (b) Re-implement the class so that its objects are immutable.

```
class Price {
  ...
  private double value = 0.0;

  public void increment(double amount) {
    this.value += amount;
  }
  ...
}
```

Answer in Java

```
final class Price {                 // final: forbids subclasses
  ...
  private final double value;   // initialized once
                                // (usually, in the constructor)

  public Price(double value) {
    this.value = value;
  }

  public Price increment(double amount) {
    return new Price(this.value + amount);
  }
  ...
}
```

Answer in Java, using records

```
public record Price(double value) {

  public Price increment(double amount) {
    return new Price(value + amount);
  }

}
```

Records: simple and compact syntax for implementing immutable objects, available from Java 14

2. In the next three slides, we show the code of a function from the open-source system called FitNesse, which is discussed in Robert C. Martin's Clean Code book.

(a) What code smell exists in this function?

(b) What is the main refactoring that eliminates this smell?

Note: The complete function code is available at this link.

```
public static String testableHtml(
        PageData pageData,
        boolean includeSuiteSetup
) throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();
    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup = PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_SETUP_NAME, wikiPage
            );
            if (suiteSetup != null) {
                WikiPagePath pagePath =
                                 suiteSetup.getPageCrawler().getFullPath(suiteSetup);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -setup .")
                        .append(pagePathName)
                        .append("\n");
            }
        }
        WikiPage setup = PageCrawlerImpl.getInheritedPage("SetUp", wikiPage);
        if (setup != null) {
            WikiPagePath setupPath = wikiPage.getPageCrawler().getFullPath(setup);
            String setupPathName = PathParser.render(setupPath);
            buffer.append("!include -setup .")
                    .append(setupPathName)
                    .append("\n");
        }
    }
    buffer.append(pageData.getContent());
    if (pageData.hasAttribute("Test")) {
        WikiPage teardown = PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
        if (teardown != null) {
            WikiPagePath tearDownPath =
                                wikiPage.getPageCrawler().getFullPath(teardown);
            String tearDownPathName = PathParser.render(tearDownPath);
            buffer.append("!include -teardown .")
                    .append(tearDownPathName)
                    .append("\n");
        }
        if (includeSuiteSetup) {
            WikiPage suiteTeardown = PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_TEARDOWN_NAME,
                    wikiPage
            );
            if (suiteTeardown != null) {
                WikiPagePath pagePath =
                         suiteTeardown.getPageCrawler().getFullPath(suiteTeardown);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -teardown .")
                        .append(pagePathName)
                        .append("\n");
            }
        }
    }
    pageData.setContent(buffer.toString());
    return pageData.getHtml();

}
```

End