Guides for Writing Better Code

Common guideliines for writing better code.

Guidelines

1. Code should be easy to read.

2. Be consistent in naming and formatting.

3. Use comments where helpful. Explain & document code. Write Javadoc.

4. Write (simple) methods that do just one thing.

5. Avoid duplicate logic and duplicate code (DRY).

6. Localize variables.

7. No magic numbers.

8. Treat warnings like errors.

Why?

1. Help write correct code.

   Easy to Read -> *fewer bugs*

   Simple methods -> *verify by testing & code review*

   No duplicate code -> *no inconsistencies*

2. Easy to modify or update.

3. Code that others can understand.

   – If they don't understand it, they won't use it.

4. Reusable.

Code should be easy to read

1. Use a *coding standard (names and formatting).*

2. Use space to separate blocks of code.

3. Use space around operators, (...), and {...}

   Bad: `if(x<0||x>=board.width)return false;`

   Good: `if (x < 0 || x >= board.width) return`

4. Use descriptive variable & method names:

   Good: gameBoard, player, piece, moveTo(Piece, x, y)

   Bad: b, play, pc, mv(Piece, x, y)

5. Use simple expressions instead of complex ones.

Java Naming Convention

```java
public class BankAccount {

    private String accountId;

    private Person owner;


    public double getBalance() {

        ...

    }


    public void deposit(

            Money amount) {...

    }
```

class name is TitleCase. Usually a noun.

variable names are camelCase. Don't use abbreviations.

methods names are camelCase. Usually a verb phrase. Don't use abbreviations.

Class Names

Good Class Names

BankAccount

Person

Menu

MenuItem

OrderTaker

Inventory

RestaurantUI

avoid "___System", it is vague and overused.

Bad Class Names

bankaccount

BankAcct (no abbrev.)

main

TEST

Program

assignment02

OrderSystem

Class (Java has a core class named "Class")

Variable & Method Names

Good Names

```
bankAccount

quantity

totalPrice

# methods are behavior

# names should be verbs

printMenu()

getBalance()

addToOrder(item)

computeTotal()
```

Bad Names

```
bankacct

qnty

q1, q2, sum1, sum2

check (*)


menu()

bal()

order(item)
```

(*) In OOP2, points deducted for a variable named "check". Its usually a sign of BAD LOGIC.

Order of code in a class

```java
public class BankAccount {

    private static final String CURRENCY = "Baht";

    private static long nextAccountId;

    private String accountId;

    private Person owner;

    /** Constructor initializes a new object. */
    public BankAccount() {

        ...

    }


    public double getBalance() {

        ...

    }
```

1. constants

2. static variables

3. attributes of objects

4. constructor(s)

5. methods

What comes BEFORE a class?

```
package ske.food;

import java.util.Scanner;

/**

 * Console application to display

 * menu and record a food order.

 * @author Fatalai Jon

 */

public class OrderTaker
```

package (optional) - lowercase

imports

Class Javadoc comment - describe the class.

Valid tags:
```
    @author Your Name
    @version 2017.09.26
    @see URL
```
(you can repeat a tag)

Variable & Method Names

Good Names

bankAccount
quantity
totalPrice
 # methods are behavior
 # names should be verbs
printMenu()
getBalance()
addToOrder(item)
computeTotal()

Bad Names

bankacct
qnty
q1, q2, sum1, sum2
check (*)
menu()
bal()
order(item)

(*) In OOP2, points deducted for variable named "check". Its usually a sign of BAD LOGIC.

Use Comments to Explain "why"

Good comments help reader understand the code

```
public void paint(Graphic g) {
    // create a copy of g so we can modify it
    // this is recommended by Swing
    g = g.create();
```

Bad comments just state the obvious

```
    // if the piece is null then return
    if (piece == null) return;
    // add all the scores in list
    int total = scores.stream( ).sum( );
```

Leave Vertical Space (blank lines)

Blank lines help you divide a long code into components.
Compare this code with the next slide.

```java
class Banknote {
    private double value;
    private String currency;
    public Banknote(double value, String currency) {
    this.value = value;
    this.currency = currency;
    }
    public int compareTo(Banknote other) {
        if ( !other.currency.equals(this.currency))
            return currency.compareTo(other.currency);
        return Double.compare(this.value,other.value);
    }
    public String toString() {
        return String.format("%f %s note",value,currency);
    }
    ...
```

Code with Vertical Space

Which version is easier to understand?

```
class Banknote {
    private double value;
    private String currency;

    public Banknote(double value, String currency) {
        this.value = value;
        this.currency = currency;
    }

    public int compareTo(Banknote other) {
        if ( !other.currency.equals(this.currency) )
            return currency.compareTo(other.currency);
        return Double.compare(this.value,other.value);
    }

    public String toString() {
        return String.format("%f %s note",value,currency);
```

Indent Your Code using TAB

Use TAB, not spaces. Set 1 TAB = 4 spaces (for Java).
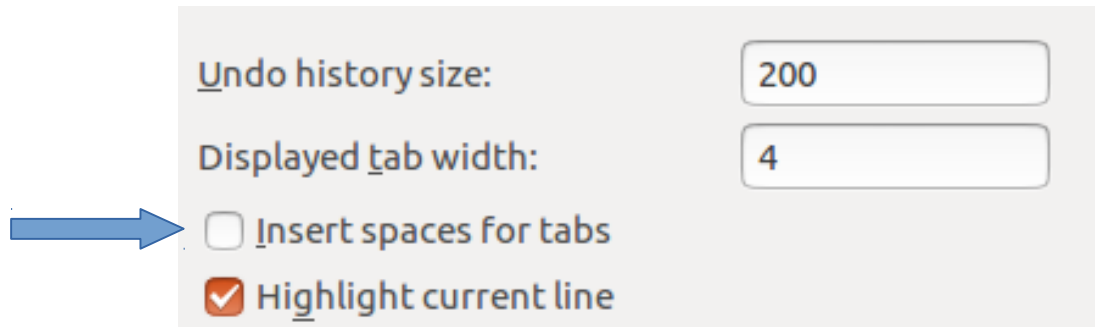
TAB TAB TAB

```java
class Greeter {
    private static Scanner console;

    public void greet(String name)
    {
        LocalTime now = LocalTime.now();
        if (now.getHour() < 12) {
            System.out.println("Good morning, "+name);
        }
        else {
            System.out.println("Good afternoon, "+name);
        }
    }
}
```
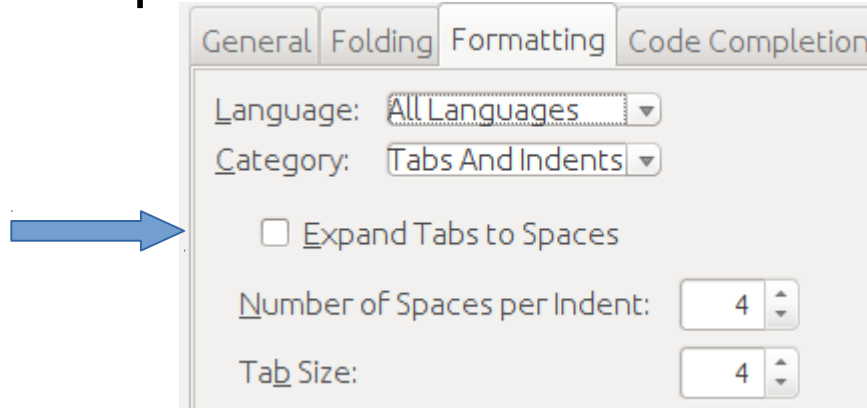
Check your IDE Settings

Eclipse: *default settings are OK*

Window -> Preferences -> General -> Text Editors

Undo history size:	200
Displayed tab width:	4

☐ Insert spaces for tabs

☑ Highlight current line

Netbeans: *inserts spaces by default. Change it.*

Tools -> Options -> Editor -> "Formatting" tab

General | Folding | **Formatting** | Code Completion

Language: All Languages ▾
Category: Tabs And Indents ▾

☐ Expand Tabs to Spaces

Number of Spaces per Indent: 4

Tab Size: 4

Methods

1. Method should do just one thing

2. Strive for simple methods

    - usually < 20 lines of code

3. Create a method for repeated task

4. Create a method to break up bigger tasks

Look for repeated tasks

Do you see any duplicate code? Can you use a method?

```java
/** SKE Restaurant partial code */
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);

    System.out.print("Enter choice (1-4): ");
    int choice = console.nextInt();
    if (choice>= 1 && choice<=3) {
        System.out.print("Enter quantity: ");
        int quantity = console.nextInt( );
    }
    ...
```

Method for a repeated task

```java
// Scanner is static attribute so methods can use it
private static Scanner console = new Scanner(System.in);

public static int getIntReply(String prompt) {
    System.out.print(prompt);
    int reply = console.nextInt();
    return reply;
}

public static void main(String[] args) {
    int choice = getIntReply("Enter choice (1-4): ");
    if (choice>= 1 && choice<=3) {
        int quantity = getIntReply("Enter quantity: ");
    }
    ...
```

Helpful code with Javadoc

```java
/**
 * Print a prompt message and read user's input.
 * @param prompt a prompt message to display
 * @return the user's reply, as an int value
 */
public static int getIntReply(String prompt) {
    System.out.print(prompt);
    // prompt should end with space for readability
    if (!prompt.endsWith(" ")) System.out.print(" ");
    int reply = console.nextInt();
    return reply;
}
```

Write Methods that do just one thing

I need an example...

duplicate code that's hard to read

```java
/** Read words from inputStream. Count by first letter. */
Map<Character,Integer> map =
                    new HashMap<Character,Integer>();
Scanner in = new Scanner(inputStream);
while(in.hasNext()){
    String word = in.nextLine();
    if(map.containsKey(word.charAt(0))){
        map.put(word.charAt(0),map.get(word.charAt(0))+1);
    }
    else{
        map.put(word.charAt(0),1);
    }
}
```

*What is **duplicate** (used many times) in this code?*

use a variable to avoid redundant method call

```java
/** Read words from inputStream. Count by first letter. */
Map<Character,Integer> map =
                        new HashMap<Character,Integer>();
Scanner in = new Scanner(inputStream);
while (in.hasNext()) {
    String word = in.nextLine();
    char firstChar = word.charAt(0);
    if ( map.containsKey(firstChar) ) {
        map.put(firstChar, map.get(firstChar)+1);
    }
    else {
        map.put(firstChar, 1);
    }
}
```

and add space for readability!

add explanatory variable (count)

```java
/** Read words from inputStream. Count by first letter. */
Map<Character,Integer> map =
                        new HashMap<Character,Integer>();
Scanner in = new Scanner(inputStream);
while (in.hasNext()) {
    String word = in.nextLine();
    char firstChar = word.charAt(0);
    int count = 0;  // number of times this char was seen
    if ( map.containsKey(firstChar) ) {
        count = map.get(firstChar);
    }
    map.put(firstChar, count+1);
}
```

Let Eclipse do it for you

"Define a variable" is an example of refactoring.

1) select expression you want to define as variable:

   `word.charAt(0)`

2) From menubar (or right-click):

   Refactor -> Extract Local Variable...

Anticipate and avoid errors

```
System.out.print("Input item number: ");
int item = scanner.nextInt();
```

What could go wrong?

```
Input item number:  one
java.util.InputMismatchException
```

```
int item = Integer.MIN_VALUE;
do {
    System.out.print("Input item number: ");
    if (scanner.hasNextInt()) item = scanner.nextInt();
    else {
        scanner.nextLine(); // discard the input line
        System.out.println("Sorry. Please input an int.");
    }
} while (item == Integer.MIN_VALUE);
```

Be careful of Strings

```
System.out.print("Are you awake? ");

String awake = scanner.next();

if (awake.equals("yes")) ...          What could go wrong?

else System.out.println("Then wake up!!");
```

```
Are you awake?   Yes
Then wake up!!
```

```
// case insensitive comparison
if ( awake.equalsIgnoreCase("yes") ) ...
// convert string to lowercase (or uppercase)
awake = awake.toLowerCase();
```

Avoid Duplicate Logic

```java
/** Insert money if the purse is not full. */
public boolean insert(Valuable obj) {
    if (money.size() == capacity) return false;
    return money.add(obj);
}
```

Avoid duplicate logic and make the *intention* clear:

```java
public boolean insert(Valuable obj) {
    if ( isFull() ) return false;
    return money.add(obj);
}
```

Localize variables

Use the smallest possible scope

- attributes only for things that object/class must remember
- minimize sharing of information between methods

```
public class Factor {
  static int sumFactors;

  public static int findFactors(int n) {
    sumFactors = 0;
    for(int factor=2; factor<=n; factor++) {
      if (n%factor == 0) sumFactors += factor;
    }
    return sumFactors;
  }
}
```

Avoid Magic Numbers

SKE Restaurant code:

```java
public static void printMenu() {
    System.out.println("1) Pizza \t250 Baht");
    System.out.println("2) Chicken \t120 Baht");
    ... // more menu choices
}
public static double getTotal(int numPizza,
        int numChicken, ...) {
    double total = 250.0*numPizza + 120*numChicken
                    + ... ;
    return total;
}
```
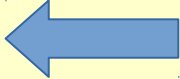
Requirements Change:

Pizza only 199 Baht!
(this week only)

How are you going to fix the code?

Use Named Constants

`final` declares a variable that cannot be changed (constant)

```java
static final double PIZZA_PRICE = 250.0;
static final double CHICKEN_PRICE = 120.0;
public static void printMenu() {
    System.out.printf("1) %-20s %6.2f Baht%n",
                        "Pizza", PIZZA_PRICE );
    System.out.printf("2) %-20s %6.2f Baht%n",
                        "Chicken", CHICKEN_PRICE);
}
public static double getTotal(int numPizza,
        int numChicken, ...) {
    double total = PIZZA_PRICE*numPizza
                    + CHICKEN_PRICE*numChicken + ... ;
    return total;
}
```

Constant for Duplicate Format String

final declares a variable that cannot be changed (constant)

```
static final double PIZZA_PRICE = 250.0;
static final double CHICKEN_PRICE = 120.0;
public static void printMenu() {
    final String FORMAT = "%d) %-20.20s %6.2f Baht%n";
    System.out.printf(FORMAT, 1,"Pizza", PIZZA_PRICE );
    System.out.printf(FORMAT, 2,"Chicken",CHICKEN_PRICE);
    ...
}
public static double getTotal(int numPizza,
        int numChicken, ...) {
    double total = PIZZA_PRICE*numPizza
                    + CHICKEN_PRICE*numChicken + ... ;
    return total;
}
```

Avoid Magic Strings, too

```java
public class Coin {
    public Coin(double value) {
        this.valiue = value;
        this.currency = "Baht";     //BAD
    }
```

```java
public class Coin {
  public static final String
        DEFAULT_CURRENCY = "Baht"; //GOOD
    public Coin(double value) {
        this.valiue = value;
        this.currency = DEFAULT_CURRENCY;
    }
```

Named Constant Explains a Number
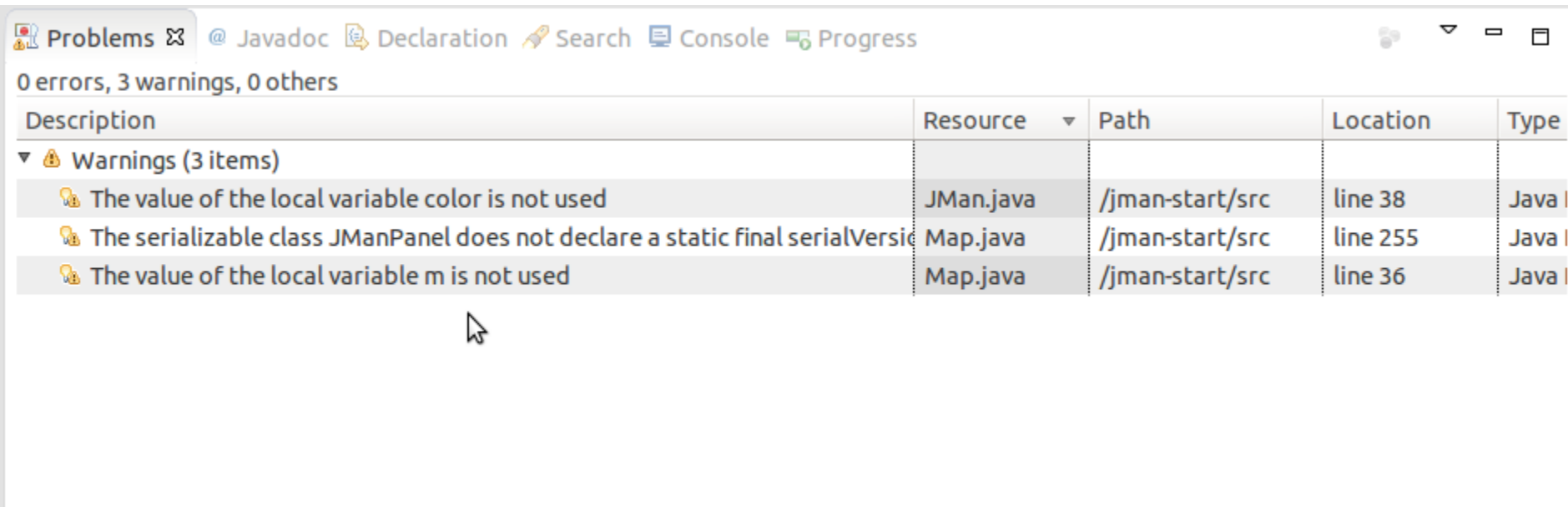
This is code from a StopWatch. Times are in nanoseconds.

```java
public double getElapsedTime() {
    return (stopTime - startTime) * 1.0E-9; // Bad
}
```

```java
/** conversion from nanoseconds to seconds */
static final double NANOSECONDS = 1.0E-9; // Good


public double getElapsedTime() {
    return (stopTime - startTime) * NANOSECONDS;
}
```

Treat Warnings Like Errors

Try to eliminate all the warnings in Eclipse or IntelliJ or ...

■Eclipse: *Close unrelated projects* so you only see problems for <u>this</u> project.

TestScores problem from elab: find average test score

```java
public class testscores {
    private static Scanner console = new Scanner(...);
    public static void main(String[] args)
    {
        System.out.print("Enter student's name: ");
        String name = console.next();    // bug?

        // avoid duplicate code - "Don't Repeat Yourself"
        System.out.print("Enter score 1: ");
        int score1 = console.nextInt();
        System.out.print("Enter score 2: ");
        int score2 = console.nextInt();
        System.out.print("Enter score 3: ");
        int score3 = console.nextInt();

        // There is a BUG here.  What is it?
        double avg = (score1 + score2 + score3)/3;
        System.out.printf("%s average is %.2f\n", name, avg);
    }
}
```

Example

Let's review a student code for SKE Restaurant,

and look for improvements.

References

Steve McConnell, *Code Complete, 2nd Edition.*

Robert Martin, *Clean Code*.

http://www.unclebob.com - Robert Martin's blog