

# Java program style guide

## Lecture 10a

Waterford Institute of Technology

March 16, 2015

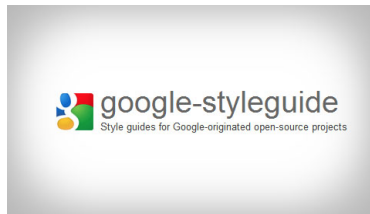
John Fitzgerald

# Java language style guide

What is a style guide: is it necessary?

A style guide:

- Describes rules to use when writing code, for example;
  - Where to locate curly braces
  - Where to use whitespace
- Convention is to adhere to guide rules
  - Convention: community-wide agreement to comply with non-legally binding practices



# Style guide

## An approach to styling to your code

Presentation based on Google style guide because:

- The guide is being maintained:
  - Last changed: March 2014
- Official Java guide 20 years old
  - Some of its rules no longer always adhered to in practice.
  - Example 2 spaces now usual for indentation rather than traditional 4 spaces.
  - However, more recent Oracle online guide referenced below.

*No Style is My Style*

# Style guide

## Advantages

- Easier to understand code
- Helps communication among developer team
- Helps maintainers
- Improves efficiency
- Reduces risk of error, for example:
  - `int index=startTime+1;`
  - `int index = startTime + i;`



# Style guide

## Why bother?

This code works fine:

```
public int newCustomer(String name, int balance){int accNmr =  
    nextAccountNumber;if(accountNumberSet.contains(accNmr) == true){  
    System.out.println("Fatal error: invalid account number");return  
    Integer.MIN_VALUE;}accountNumberSet.add(accNmr);Customer  
    customer = new Customer(name, accNmr, balance);customers.add(  
    customer);customerRecord.put(accNmr, customer);  
    nextAccountNumber += 1;return accNmr;}
```

# Style guide

## Why bother?

Same code but styled:

```
public int newCustomer(String name, int balance)
{
    int accNmr = nextAccountNumber;
    if (accountNumberSet.contains(accNmr) == true) {
        System.out.println("Fatal error: invalid account number");
        return Integer.MIN_VALUE;
    }
    accountNumberSet.add(accNmr);
    Customer customer = new Customer(name, accNmr, balance);
    customers.add(customer);
    customerRecord.put(accNmr, customer);
    nextAccountNumber += 1;
    return accNmr;
}
```

# Google Java Style

## File name

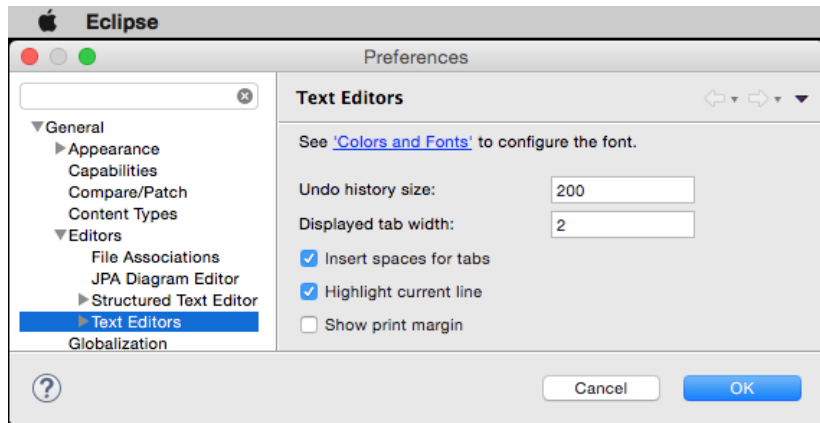
- Source file name case sensitive
- File name and corresponding class name begin with upper case letter
  - *Circle.java* : ok
  - *circle.java* : not ok

```
/*  
 * @file Circle.java  
 */  
public class Circle  
{  
  
}
```

# Formatting

## Indentation

- Indent 2 spaces
  - Do not use Tab characters to indent
  - Check IDE configuration





# Formatting

## Braces where optional

*Braces are used with if, else, for, do and while statements, even when the body is empty or contains only a single statement.*

```
// this is what the guide calls for
// here for is followed by a single statement
for (int i = 0; i < a.length; i += 1) {
    a[i] *= 2;
}
```

```
// omission of {} is legal in this example
// and we have already seen this style in labs
for (int i = 0; i < a.length; i += 1)
    a[i] *= 2;
```

# Formatting

## Positioning of braces

Two styles possible: a language design error?

```
// this is what the guide calls for
for (int i = 0; i < a.length; i += 1) {
    a[i] *= 2;
    b[size - 1 - i] = a[i];
}
```

```
// here is another style
for (int i = 0; i < a.length; i += 1)
{
    a[i] *= 2;
    b[size - 1 - i] = a[i];
}
```

# Formatting

Empty blocks may be concise

As example consider a default constructor that does nothing

```
public class Circle {  
    public Circle() {  
  
    }  
}
```

```
// written concisely  
public class Circle {  
    public Circle() {}  
}
```

# Formatting

Block indentation + 2 spaces

```
public class Financials
{
    public static String generatePin() {
        String pin = new String();
        for (int i = 0; i < NUMBER_PIN_DIGITS; i += 1) {
            pin += Byte.toString((byte)(Math.random()*9 + 1));
        }
        return pin;
    }
}
```

# Formatting

## Vertical whitespace

A single blank line appears between:

- constructors
- methods
- within method bodies to create logical groupings

```
public class Cylinder
{
    double radius;
    double height;

    public Circle() {}

    double area() {
        return Math.PI*Math.pow(radius, 2);
    }
}
```

# Formatting

## Horizontal whitespace

A single space appears:

- separating reserved word from open parens

```
for (int i = 0; i < size; i++)
```

- before { in situation similar to here:

```
for (int i = 0; i < size; i++) { . . . }
```

- On both sides of the double slash (//) that begins an end-of-line comment.
- Between type & variable of declaration

```
ArrayList<String> list;
```

# Formatting

## Horizontal whitespace

A single space appears:

- Optional just inside both braces of an array initializer

```
new int[] {5, 6}; // ok
```

```
new int[] { 5, 6 }; // ok
```

- After comma, colon, semi-colon.

```
switch (month) {  
    case 5 : monthString = "May";  
}
```

- On both sides colon in ternary operator

```
boolean b = x > y ? true : false;
```

- On both sides colon enhanced for (foreach)

```
for (String s : strings) { . . . }
```

# Formatting

## Horizontal whitespace

A single space appears after:

- Closing parens of cast

```
int x = (int) obj.getPosition();
```

- Both sides binary and ternary operators

```
boolean b = x > y ? true : false;
```

```
int index = startIndex + i; // ok (clear)
```

```
int index=startIndex+i; // not ok (error prone)
```



# Formatting

Horizontal alignment never required

```
private int x; // this is fine
private Color color; // this too
```

```
private int    x;    // permitted, but future edits
private Color color; // may leave it unaligned
```

# Formatting

One statement per line

However, we have seen when this rule is broken with advantage:

- example merge sort (Sedgewick & Wayne)

```
for (int k = lo; k <= hi; k++)  
{  
    if (i > mid)                a[k] = aux[j++];  
    else if (j > hi)            a[k] = aux[i++];  
    else if (aux[j] < aux[i]) a[k] = aux[j++];  
    else                        a[k] = aux[i++];  
}
```

# Formatting

## One variable per declaration

Each variable whether field (instance or class) or local:

- declared individually

```
// this is ok
```

```
int a;
```

```
int b;
```

```
// this is legal but not ok
```

```
int a, b;
```

# Formatting

## Comments

Used to help reader understand code

```
/*  
 * This is  
 * okay  
 */
```

```
// And so  
// is this
```

```
/* Or you may  
 * even use this style */
```

# Naming

## Rules to apply naming identifiers

### Rules applicable to all identifiers

- Use only ASCII letters and digits
  - `int circleOfFire100;`
- Exception: underscore in constant names
- Class names written UpperCamelCase
  - `public class CircleOfFire`
- Method names written lowerCamelCase
  - `int getRadiusCircleOfFire();`
- Constant names ALL\_UPPER\_CASE

# JavaDoc

## Formatting

Where JavaDoc used:

- public class
- public member of class
- protected member of class

```
/** An especially short bit of Javadoc. */
```

```
/**  
 * Multiple lines of Javadoc text are written here,  
 * wrapped normally...  
 */  
public int method(String p1) { ... }
```

# JavaDoc

## Formatting

```
/**
 * @file   Circle.java
 * @brief   This class describes a geometric 2—d circle . . .
 * @version 1.0 April 1, 2014
 * @author  . . .
 */
public class Circle
{
    double radius

    /**
     * Constructs a new Circle object defined by user—supplied parameters
     * @param radius  radius of circle
     */
    public Cone(double radius) { . . . }
}
```

# Summary

## Styling code

- Importance of style
- Various style conventions
- What style to use?
  - Styling consistency throughout project
- What style includes
  - Selection names for files, fields & methods
  - Formatting such as whitespace, indentation, braces, comments.
  - JavaDoc



# Referenced Material

## 1. Google Java Style.

`https://google-styleguide.googlecode.com/svn/trunk/javaguide.html`

[Accessed 2015-03-03]

## 1. How to Write Doc Comments for the Javadoc Tool

`http://www.oracle.com/technetwork/articles/java/index-137868.html`

[Accessed 2015-03-03]