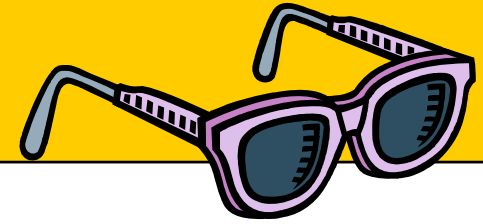




SWEN221: Software Development #4 - Style

David J. Pearce & Nicholas Cameron & James Noble & Petra Malik
Computer Science, Victoria University

Why bother with style?



- Java allows you to write code that is very easy/hard to understand
- The aim of code, comments, diagrams, documentation is **to communicate**
 - With yourself
 - With your team
 - With those who will come after you
- Style guide helps to produce code that is clear and consistent and thus easier to read and maintain

Files & Comments

- Files:
 - Organise them according to Java conventions
 - Eclipse will do this for you!
- Comments:
 - `/** Javadoc comment */`, `/* */` or `//`
- Tips
 - Good code does not need many comments
 - Good names (variable, class, field etc.) help
 - Only tricky code needs commenting
 - Do not comment just for the sake of it:
 - E.g. `x = 1 // 1 is assigned to x`
 - **Use Javadoc!!!**

Quiz: what's good/bad about this?

```
public class Book { // This class represents a Book
    private String x;
    private String y;

    public Book(String t, String a) {
        x = t; // set title
        y = a; // set author
    }
    public String getAuthor() { // Returns Book's Author
        return y;
    }
    public String getTitle() { // Returns the Book's Title
        return x;
    }
}}
```

Layout

- Be consistent!
- Indentation (use some)
- Braces (either beginning or end of lines)
- Declare fields together (at beginning or end)
- Declare **public** methods together
- Likewise **protected/private** & same for fields
- Don't be a Jerk!

```
int ivl_billclint0n[]; foo.bar  
(  x<      ivl_billclint0n,  
(0, true))};}} class nextclass { ...
```

Two ways to use curly braces

```
int method(int x) {  
    int y=3;  
    return x+y;  
}
```

```
int method(int x)  
{  
    int y=3;  
    return x+y;  
}
```



Dave says: I prefer the left version. Writing code is like gazing at the world through a porthole - you can never see everything you want at once. The version on the right is too verbose; it makes my "window to the world" even smaller than it needs to be.

Names



- Packages
 - lowercase
- Classes
 - `CapWithWholeWordsCaps`
- Exception
 - `ClassNamesWithException`
- Interface (when necessary to distinguish from class)
 - `EndWithI` or `Ifc`
- Class (when necessary to distinguish from interface)
 - `EndWithImpl` or `EndWithObject`
- Constant (finals)
 - `UPPERCASE_UNDERSCORE`
- Avoid magic numbers — use constants instead

Names

- Fields
 - `firstLowerThenCaps` (or `trailing_` or `thisVar`)
- Local variables
 - `firstLowerThenCaps` (or `lowecase_with_underscores`)
- Methods
 - `firstLowercaseThenCaps`
- Getters/Setters
 - `(T getX() or T x()), (setX(T v) or x(T v))`
- Factory/Creator Method
 - `newT()`
- Converter Method
 - `T toT()`

Keep variables local!

- Always use smallest **scope** possible
- E.g. prefer A to B

class Date { A

```
    int day;  
  
    int nextDay() {  
        int r = day + 1;  
        return r;  
    }  
}
```

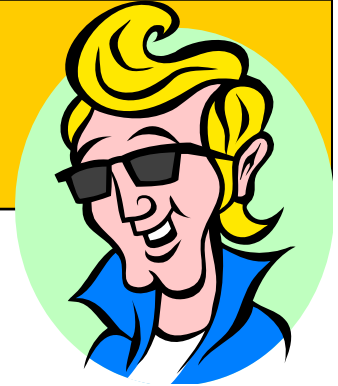
class Date { B

```
    int day;  
    int r;  
  
    int nextDay() {  
        r = day + 1;  
        return r;  
    }  
}
```

Others

- **Arrays:**
 - `"Integer[] x"` (not `"Integer x[]"`)
- **Guard casts with conditionals**
 - E.g. `if(x instanceof C) { C y = (C) x; ... } else ...`
- **Separate *accessors* and *mutators***
 - Otherwise people are forced to mutate
 - E.g. `T pop() => void pop() & T top()`
- **Avoid "=" inside if- and loop-conditions**
 - E.g. `if((x=aMethod()) == 2) { ... }`
- **Prefer `Object.equals()` rather than "=="**
 - Otherwise strange things can happen ...

Example: wheres the bug?



```
class CLS_VeHicle { int WHEELZ =  
    3; int how_manyweeehlz() {return  
    (int) WHEELZ;}  
void set_wheels(CLS_VeHicle _W){  
    _W.WHEELZ = WHEELZ; }  
void set_wheels(CLS_Motor_Car W_)  
    { WHEELZ      =  
      ((CLS_Vehicle)W_).WHEELZ;  
      }}
```

Another Example

```
class Date {  
    int day;    // day field  
    int month; // month field  
    int year;   // year field  
  
    int nextDay() { // next day method  
        int r = day + 1; // r is day + 1  
        return r;        // return r  
    }  
}
```

- What's wrong with this?

Yet Another Example

```
class Date {  
    int day, month, year;  
  
    public Date(int day, int month, int year) { ... }  
  
    /**  
     * Return the day after this one.  
     */  
    Date nextDay() { return new Date(day+1,month,year); }  
  
    /**  
     * Return the day after this one.  
     */  
    Date prevDay() { return new Date(day-1,month,year); }  
}
```

Still Anther Example

```
private Block parseTry(Tree stmt, FlowGraph cfg) {
    FlowGraph.Point exit = codePoint(null, stmt);
    Block body = parseStatement(stmt.getChild(0), null, cfg);
    Block rb = new Block(body);
    for (int i = 1; i < stmt.getChildCount(); ++i) {
        Tree child = stmt.getChild(i);
        if (child.getType() == CATCH) {
            Tree param = child.getChild(0);
            Type.Reference exceptionT = (Type.Reference) parseType(param
                                                                    .getChild(0));

            scopes.push(new Scope());
            String name = scopes.peek().id + param.getChild(1).getText();
            scopes.peek().variables.add(name);
            cfg.add(new FlowGraph.LocalVarDef(name, exceptionT, 0, false));
            ...
        }
    }
}
```

- What does this code do ?

Tools can help ...

- Checkstyle
 - <http://checkstyle.sourceforge.net/>
- Jalopy (source code beautifier)
 - <http://jalopy.sourceforge.net/>
- PMD
 - <http://pmd.sourceforge.net/>
- Jlint, FindBugs, etc.
 - Look for possible bugs in Java code

SWEN221 Style

- See “**Good Programming Style**” page
 - http://ecs.victoria.ac.nz/Courses/SWEN221_2015T1/StyleGuide
- Read it and use it!
- You will be marked according to this style!