Coding Standards

Semester 1 TU856/3

Dr. Susan McKeever

There is no point in writing code unless you follow basic standards

There is no point in writing code unless you follow basic standards

- 80% of the cost of software is on *maintaining* code.
  - Code is rarely left as is – it will be reused/improved/ extended
  - Typically not the original developer

There is no point in writing code unless you follow basic standards

- 80% of the cost of software is on *maintaining* code.
  - Code is rarely left as is – it will be reused/improved/ extended
  - Typically not the original developer

There is <u>**no point**</u> in writing code unless you follow basic standards

One man's crappy software is another man's full time job. Jessica Gaston

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live"

Martin Goldin

"If your code looks a mess (and can't be understood without any reading any comments).. It IS a mess. *(ME)*

Common sense

- Comment your code
  - Header at the top of your code
  - Every method explained
  - But avoid **over-commenting** to compensate for badly written code
  - comments do not refactor.

- meaningful names for variables, classes, objects…
- java conventions (see overleaf)
- Many more.. To be covered.

Comment Header blocks

•Always include a Header block of comments at the top of your program explaining

Author, date written, date modified, and a description of What the code does

```
*******************************************************************************
*                                                                            *
*   Author: Audrey Clinton                                                      *
*                                                                            *
*   Created: 03/03/12
         *
*   Modified: 12/Mov/2012                                                        *

*   Modification1: to change all date foramts from dd/mm to
dd/mmm
*etc                                                                         *
*************************************************
**************
```

Self documenting code

```
public void abc(int a){
       r = a / 2;
       while ( abs( r - (a/r) ) > t ) {
            r = 0.5 * ( r + (a/r) );
       }
       System.out.println( "r = " + r );
}
```

```
public void squareRoot(int num)
{
       root = num/ 2;
       while ( abs(root - (num/ root) ) > t )
      {
             r = 0.5 * (root + (num/ root));
       }
             System.out.println( " root = " + root );
}
```

Comments blocks

•*A comment will/may be needed for every method to explain it’s purpose*

•And may be for relevant lines of code

•**But** be mindful that code *can be self documenting*

•**GOLDEN RULE**

- **Always comment as if the code will be modified by someone else, without access to your guidance**

Use the Conventions for the language

- Java is Case sensitive
- Use the conventions
    - Classes should be nouns, capitalised first letter e.g.

      ```
      public class Student
      ```
    - Variables mixed case starting with lower. E.g. **acctBalance, line**
    - Constant are all upper case e.g. **COLOR_RED**
    - Methods are verbs, mixed case starting with lower e.g. **getBalance()**

Use the Conventions for the language

•**Classes**: Names - first letter of each word capitalised (CamelCase). Use Nouns

```
 class Customer
class CurrentAccount
```

•**Interfaces:** CamelCase. Name usually = an operation that a class can do:

```
Interface Logger
interface ColourManager
```

Indentation, alignment and spacing

- Proper alignment/ indentation of code is critical

```
int myfunction(int a)
   { if ( a == 1 ) {
    printf("one"); return 1; //
the cursor is in this line }
return 0; }
```

```
int myfunction (int a)
 {
     if ( a == 1 )
     {
          printf("one");
          return 1; // the cursor is in this line
      }
      return 0;
}
```

Refactoring

- **Definition**
- "disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior",. In order to improve the NON functional aspects of the code

In plain English:
Reorganising your code to make your code clearer and cleaner and simpler - without changing the functionality

Refactoring: techniques used

- More "abstraction" (i.e. hiding the implementation details/complexity)
- Examples of this:
  - Encapsulation of attributes
    - Use getter and setter methods for attributes
  - Replacing conditional code with polymorphism
    - E.g. Using Shape (super) /Circle (sub) /Square (sub) – instead of "if/else"

Refactoring: techniques used

- **Improve names and location of code**
- Examples of this:
  - Move <u>repeating</u> code in a class into a method
    - E.g. Eclipse and some other IDEs will identify and move
  - Change field or method name to a more meaningful name
    - Eclipse will find all occurrences
  - Push class up or down (i.e. to super/sub class)

Refactoring: techniques used

- **Break Code into smaller, logical pieces**
- Examples of this:
  - Avoid large unnecessarilycomplex classes
    - Extract class – moves part of class to a new class
  - Avoid large complex methods
    - Extract method