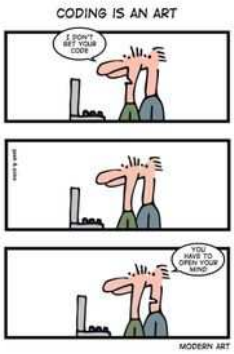

Some Coding Guidelines

www.cs.uoi.gr/~zarras/http://www.cs.uoi.gr/~zarras/se.htm

Don'ts & Do's

- Don't mix the data model and the logic of the application with the GUI classes of the application.
- Classes
  - Make **classes small** and cohesive - A single well defined responsibility for a class
  - Don't break **encapsulation** by making the data representation public
  - **Class names** are important – use descriptive names for the concepts represented by the classes
  - Use **Noun & Noun phrases** for class names
  - **See here for more - http://www.cs.uoi.gr/~zarras/soft-devII.htm**

Don'ts & Do's

- Methods
  - Make **methods small** – A method must **do one thing**
  - **Method names** are important – use descriptive names for the concepts represented by the methods
  - Use **Verb & Verb phrases** for method names
  - **See here for more - http://www.cs.uoi.gr/~zarras/soft-devII.htm**
- Fields
  - Make **fields private** – A method must do one thing
  - **Field names** are important – use descriptive names for the concepts represented by the fields
  - Use **Noun & Noun phrases** for field names

Conventions

- **Source file organization**
  - A Java source file should contain the following elements, in the following order:
    - Copyright/ID block comment
    - package declaration
    - import declarations
    - one or more class/interface declarations
  - At least one blank line should separate all of these elements.

Conventions

- **Package naming**
  - Generally, package names should use only **lower-case letters** and digits, and no underscore. Examples:
    - `java.lang`
    - `java.awt.image`
    - `dinosaur.theropod.velociraptor`

Conventions

- **Class/Interface naming**
- All type names (classes and interfaces) should use the *InfixCaps* style.
  - Start with an **upper-case** letter, and **capitalize the first letter of any subsequent word** in the name, as well as any letters that are part of an acronym. All other characters in the name are lower-case.
  - Do not use underscores to separate words.
  - Class names should be **nouns** or **noun phrases**.
  - Examples:
    - `// GOOD type names:`
      - `LayoutManager, AWTException, ArrayIndexOutOfBoundsException`
    - `// BAD type names:`
      - `ManageLayout // verb phrase`
      - `awtException // first letter lower-case`
      - `array_index_out_of_bounds_exception // underscores`

Conventions

- **Field naming**
- Names of **non-constant fields** (reference types, or non-final primitive types) should use the *infixCaps* style.
  - Start with a **lower-case** letter, and **capitalize the first letter of any subsequent word** in the name, as well as any letters that are part of an acronym. All other characters in the name are lower-case.
  - Do not use underscores to separate words.
  - The names should be **nouns** or **noun phrases**.
  - Examples:
    - `int index;`
    - `String color;`
- Names of fields being used as ***constants*** should be **all upper-case**, with **underscores** separating words.
  - Examples:
    - `MIN_VALUE, MAX_BUFFER_SIZE, OPTIONS_FILE_NAME`

Conventions

- **Method naming**
- Method names should use the *infixCaps* style.
  - Start with a **lower-case** letter, and capitalize the first letter of any subsequent word in the name, as well as any letters that are part of an acronym. All other characters in the name are lower-case.
  - Do not use underscores to separate words.
  - Method names should be **imperative verbs** or **verb phrases**. Examples:
  - // GOOD method names:
    - `showStatus(), drawCircle(), addLayoutComponent()`
  - // BAD method names:
    - mouseButton() // noun phrase; doesn't describe function
    - DrawCircle() // starts with upper-case letter
    - add_layout_component() // underscores

Conventions

- **Blank lines**
  - A blank line should also be used in the following places:
    - After the copyright block comment, package declaration, and import section.
    - Between class declarations.
    - Between method declarations.
    - Between the last field declaration and the first method declaration in a class.

Conventions

- **Blank spaces**
- **A single blank space (not tab) should be used:**
  - Between a **keyword** and its opening **parenthesis**. This applies to the following keywords: `catch, for, if, switch, while`.
  - After any **keyword** that takes an **argument**.
    - Example: `return true;`
  - Between **two adjacent keywords**.
  - Before *and* after **binary operators except .(dot).**
  - After a **comma** in a list.
  - After the **semi-colons** in a for statement, e.g.:
    - `for (expr1; expr2; expr3) {`

Conventions

- **Continuation lines**
  - Lines should be limited to 80 columns.
  - Lines longer than 80 columns should be **broken into one or more continuation lines**, as needed.
  - All the continuation lines should be aligned

```
// RIGHT
foo(long_expression1, long_expression2, long_expression3,
      long_expression4);
// RIGHT
foo(long_expression1,
      long_expression2,
       long_expression3,
       long_expression4);
```

Conventions

- **Classes**
- A class declaration looks like the following. Elements in square brackets [] are optional.

```
[ClassModifiers] class ClassName [Inheritances] {
  1. Static variable field declarations
  2. Instance variable field declarations
  3. Static method declarations
  4. Constructor declarations
  5. Instance method declarations
}
```

Conventions

- **Method declarations**
  - All of the elements of a method declaration up to and including the opening brace "{" **should appear on a single line**
    - (unless it is necessary to break it up into continuation lines if it exceeds the allowable line length).
  - A method declaration looks like the following. Elements in square brackets "[" are optional.

```
[MethodModifiers] Type MethodName(Parameters) [throws Exceptions] {
  Method Body
}
```

Conventions

- **Simple statements**
- **Assignment and expression statements**
  - Each line should contain **at most one statement**. For example,
    - `a = b + c; count++; // WRONG`
    - `a = b + c; // RIGHT`
    - `count++; // RIGHT`

Conventions

- **Simple statements**
- **Local variable declarations**
  - Generally local variable declarations **should be on separate lines**;
    - however, an exception is allowable for temporary variables that do not require initializers. For example,
    - `int i, j = 4, k; // WRONG`
    - `int i, k; // acceptable`
    - `int j = 4; // RIGHT`

Conventions

- **Simple statements**
- **Array declarations**
  - The brackets "[]" in array declarations should immediately **follow the array name**, not the type.
    - The exception is for method return values, where there is no separate name; in this case the brackets immediately follow the type:
    - `char[] buf; // WRONG`
    - `char buf[]; // RIGHT`
    - `String[] getNames() { // RIGHT, method return value`

Conventions

- if **statement**

```
if (condition) {
  statements;
}

if (condition) {
  statements;
} else {
  statements;
}

if (condition) {
  statements;
} else if (condition) {
  statements;
} else {
  statements;
}
```

Conventions

- for **statement**

```
for (initialization; condition; update) {
  statements;
}
```

- while **statement**

```
while (condition) {
  statements;
}
```

Conventions

- switch **statement**

```
switch (condition) {
case 1:
case 2:
  statements;
  break;
case 3:
  statements;
  break;
default:
  statements;
  break;
}
```