Clean Names

Christopher Simpkins
chris.simpkins@gatech.edu

Clean Code

What is “clean code?”

- Elegant and efficient. – Bjarne Stroustrup
- Simple and direct. Readable. – Grady Booch
- Understandable by others, tested, literate. – Dave Thomas
- Code works pretty much as expected. Beatuful code looks like the language was made for the problem. – Ward Cunningham

Why do we care abou clean code?

- Messes are costly. Quick and dirty to get it done ends up not getting it done and you will not enjoy it. It’s lose-lose!
- We are professionals who care about our craft.

The Boy Scout Rule

Meaningful Names

- The name of a variable, method, or class should reveal its purpose.
- If you feel the need to comment on the name itself, pick a better name.
- Code with a dictionary close at hand.

Don’t ever do this!

```
int d; // elapsed time in days
```

Much better:

```
int elapsedTimeInDays;
int daysSinceCreation;
int daysSinceModification;
int fileAgeInDays;
```

Intention-Revealing Names

What is the purpose of this code?

```
public List<int[]> getThem() {
  List<int[]> list1 = new ArrayList<int[]>();
  for (int[] x : theList)
    if (x[0] == 4)
      list1.add(x);
  return list1;
}
```

Why is it hard to tell? – Code itself doesn't reveal context.

- What's in `theList`?
- What's special about item 0 in one of the arrays in `theList`?
- What does the magic number 4 represent?
- What is client code supposed to do with the returned list?

Intention-Revealing Names Exercise

Turns out, this code represents a game board for a mine sweeper game and `theList` holds the cells of the game board. Each cell is represented by and `int[]` whose 0th element contains a status flag that means “flagged.”

Look how much of a difference renaming makes:

```
public List<int[]> getFlaggedCells() {
  List<int[]> flaggedCells = new ArrayList<int[]>();
  for (int[] cell : gameBoard)
    if (cell[STATUS_VALUE] == FLAGGED) flaggedCells.add(cell);
  return flaggedCells;
}
```

Even better, create a class to represent cells

```
public List<Cell> getFlaggedCells() {
  List<Cell> flaggedCells = new ArrayList<Cell>();
  for (Cell cell : gameBoard)
    if (cell.isFlagged()) flaggedCells.add(cell);
  return flaggedCells;
}
```

Disinformative Names

Avoid names with baggage, unless you want the baggage.

- `hp` not a good name for hypotenuse. hp could also be Hewlett-Packard or horsepower.

Don't hint at implementation details in a variable name.

- Prefer `accounts` to `accountList`.
- Note: certainly do want to indicate that a variable is a collection by giving it a plural name.

Superbad: using O, 0, l, and 1.

```
int a = l;
if ( O == l )
  a=O1;
else
  l=01;
```

Don't think you'll never see code like this. Sadly, you will.

Names that Make Distinctions

Consider this method header:

```
public static void copyChars(char a1[], char a2[])
```

Which array is source? WHich is desitination? Make intention explicit:

```
public static void copyChars(char source[], char desitination[])
```

Meaningless distinctions:

- `ProductInfo` **versus** `ProductData`
- `Customer` **versus** `CustomerObject`

Don't be lazy with variable names.

Pronouncable and Searchable Names

- You'll need to talk to other programmers about code, so use pronouncable names.
- Also, using English words makes variable names easier to remember.
- Using descriptive names also helps you search using tools like GREP.
- Sometimes short names are acceptable if they are traditional. For example `i`, `j` and `k` for short nested loops.

General rule: the length of a variable name shoudl be proportional to its scope.

Encodings

Some misguided programmers like to embed comments and type informatoin in variable names.

- In the bad old days of Windows programming in C Charles Simponyi, a hungarian programmer at Microsoft, created an encoding scheme for variable and function names. For example, every long pointer to a null-terminated string was prefixed with `lpsz` (long pointer string zero).
- When Microsoft moved to "C++" for their MFC framework, they added encodings for member variables: the `m_` prefix (for "member").

Be very happy you never had to work with the Win API or MFC. They were awful.

Avoid Encodings

Modern type systems and programming tools make encodings even more unnecessary. So, AVOID ENCODINGS! Consider:

```
public class Part {
  private String m_dsc; // The textual descriptio
  void setName(String name) {
    m_dsc = name;
  }
}
```

The `m_` is useless clutter. Much bettwr to write:

```
public class Part {
  String description;
  void setDescription(String description) {
    this.description = description;
  }
}
```

A Few Final Naming Guidelines

- Avoid mental mapping. We’re all smart. Smart coders make things clear.
  - So simple only a genius could have thought of it. – Einstein
  - Simplicity does not precede complexity but follows it. – Perlis
- Use nouns or noun phrases for class names.
- Use verbs or verb phrases for method names.
- Don’t use puns or jokes in names.
- Use one word per concept.
- Use CS terms in names.
- Use problem domain terms in names.