Programming Logic Programming Standards

Updated: August 10, 2015

Naming Standards

camelBack notation definition: The first word or the prefix of the name is typed all lowercase. The first letter (only) of the remaining words is capitalized. If the name includes an abbreviation (like MSTC), only the first letter should be capitalized (Mstc).

Title notation definition: Title notation does not include a prefix. The first letter of every word in the name (including the first) must be capitalized. If the name includes an abbreviation (like MSTC), only the first letter of the abbreviation should be capitalized (Mstc).

- Object names
  - Objects referred to in code must be given a name (not the default).
    - Labels that are only used to describe other form objects and are not referred to in code do not need to be renamed.
    - Group boxes and panels that are not referred to in code do not need to be renamed.
    - Most other objects will need to be renamed (including the form itself).
  - Object names must be in camelBack notation.
  - Object names must begin with a prefix.
  - Object names must be spelled correctly.
  - The following is my *recommended* list of object prefixes. You do not have to use these, but you must be consistent with the prefixes you use.
    - btn Button
    - cmb ComboBox
    - dtp DateTimePicker
    - frm Form
    - grd GridBox
    - grp GroupBox
    - lbl Label
    - lst ListBox
    - mnu Menu items
    - pan Panel
    - pic PictureBox
    - rdo RadioButton
    - tmr Timer
    - txt TextBox

- Variable names
  - Variable names must be in camelBack notation.
  - Variable names that are single words must be all lowercase.
  - Variable names must be spelled correctly.
  - Prefixes designating variable type are **not** required, but you may use them. If you do, be consistent.
  - Variable names must accurately reflect what the variable stores.
  - Avoid abbreviating words. If you do abbreviate, use common, well-known abbreviations.
  - Do not give local variables the same name as form-level or global variables.

- Method Names (Programming Logic – Intermediate)
  - Method (subroutines and functions) names must be in title notation.
  - Method names must be spelled correctly.
  - The method name must accurately describe the method's purpose.
  - Since methods *do* things, the method name should begin with a verb.

- Class Names (Programming Logic – Intermediate)
  - Class names must be in title notation.
  - Class names must be spelled correctly.
  - Class names must accurately describe what the class represents.

- Constant Names
  - Constant names must be all uppercase.
  - Constant names must be spelled correctly.
  - Constant names must accurately reflect what the constant stores.
  - Avoid abbreviating words. If you do abbreviate, use common, well-known abbreviations.

Documentation Standards

- Variable documentation
  - Each variable must include its own descriptive comment.
  - Multiple variable declarations in one line of code are not allowed.
  - Variable descriptions may be sentence fragments.
  - Variable descriptions cannot include spelling errors.
  - Variable descriptions should provide details about what the variable stores. Do not simply repeat the variable name.

- Constant documentation
  - Each constant must include its own descriptive comment.
  - Constant descriptions may be sentence fragments.
  - Constant descriptions cannot include spelling errors.
  - Constant descriptions should provide details about what the constant stores. Do not simply repeat the constant name.

- Method documentation
  - Each method must include documentation that describes the method's purpose. Describe the overall purpose of the method, not the detail of its processing.
  - Method documentation cannot include spelling errors.
  - Method documentation must be grammatically correct (including complete sentences)
  - Some professionals like to include their pseudocode in the method documentation. Though not required, this is a pretty good idea.

- Application documentation
  - Each program (solution, project) must include application documentation.
    - Before Programming Logic-Beginning **Unit 4**, this documentation must appear at the top of the program's only form.
    - Starting with Programming Logic-Beginning **Unit 4**, this document must appear in a separate, documentation *module* named _Documentation.cs.
  - All application documentation will include the following:
    - Project Name: *Name of this project*
    - Programmer Name: *Your name*
    - Due Date: *Due date of this assignment (see Blackboard grade book)*
    - Purpose: *Description of the project's purpose*
      - One or two lines should do.
      - No spelling errors, grammatically correct, accurate.
    - Starting with Programming Logic-Beginning **Unit 4**, the application documentation must include (after the project purpose) form descriptions.
      - List the forms alphabetically.
      - List the name of the form.
      - Provide the purpose of the form (correct spelling, grammar and accuracy).
    - Change Log that includes:
      - The initial submittal
    - After the initial submittal of a project (in updated assignments), the Change Log must be kept current..
      - At the beginning of the log entry put the date the change was made.
      - Describe the changes made. Include the form name and method name that was changed.
      - Sort the log entries by date, descending (most recent changes first)
  - See Adding Comments to Code (PL-Beginning Unit 2) for additional information
  - See Program Documentation (PL-Beginning Unit 4) for additional information

- Inline documentation
  - Within your methods add additional comments (beyond the requirements above) to describe code processing.
  - Inline documentation cannot include spelling errors.
  - Inline documentation may incorporate sentence fragments (complete sentences not required).
  - You do not have to document every line of code, only complicated code, tricky code, or code that requires further explanation.
  - If you are maintaining (modifying) code and you can't immediately determine its purpose, add an inline comment to remind you next time.
  - In your early programs, you may use inline comments to help you remember what a line of code does (as I often do in class). As your experience grows, you should need to do this less.

Coding Standards

- Variable Scope
  - Define variables and constants at the lowest scope possible (local, class-level, global).
  - I recommend starting all variables as local variables and increasing their scope only as necessary.
  - Most constants should start as class-level variables. Make them global only if more than one form needs access them.
  - Define local variables at the beginning of a method.
    - (Unit 6) The exception to this rule is For loop control variables which be defined in the For command itself.

- Class Variables
  - All class-level variables must be designated either private or public

- Indentation
  - The Visual Studio editor indents code automatically (and correctly). Learn to recognize this indentation—not all editors automatically indent (Java editors).
  - If you break a long command across multiple lines, indent the subsequent lines to show ownership.
    - Indent wisely to increase readability.

- Spacing
  - Use spacing to increase the readability/maintainability of your code.
  - Where possible, align variable comments.
  - Method documentation must be preceded by one blank line (separate your methods) but should be directly above its method (no blank line after)

- Alphabetizingmethods
  - All methods must appear in alphabetical order to make them easier to locate
  - (Programming Logic – Intermediate) All methods within a region must appear in alphabetical order.

- Alphabetizing variables
  - Variables (local, form-level, global) do **not** need to be declared in alphabetical order
  - If there are many variables, alphabetizing their declarations can make them easier to locate.

- Unused variables
  - Remove unused variables
  - Visual Studio designates unused variables with green wavy lines

- Objects in Calculations
  - Do not use objects (TextBoxes, DateTimePickers, Labels, etc) in calculations. Transfer the contents of the objects to variables and use the variables in calculations instead.
  - Do not use objects (Labels) as the destination of a calculation. Store the calculation result in a variable, then transfer the variable to the object.

- Duplicate code (Unit 5)
  - If the exact same command appears in both the Then and the Else clause of an If statement, remove it from both clauses and place it above or below the If statement.
  - If the exact same command appears in all cases of a Switch statement, remove it from the cases and place it above or below the Switch command.
  - See (Decision structure) If/Else (PL-Beginning Unit 5) for an example.

- Empty Then clauses (Unit 5)
  - Do not write If statements that include commands in the Else clause, but no commands in the Then clause.
  - Instead, reverse the logic of the If condition so the statements appear under the Then clause and remove the Else clause.
  - See (Decision structure) Simple If (PL-Beginning Unit 5) for an example.

- Merged Event Methods (Unit 5)
  - Rename the method to an appropriate name (rdoWheat→rdoBread).

Form Standards

- Each form must be renamed according the object naming standards.
  - See Changing Form Names (PL-Beginning Unit 2).
- For clarity and maintainability, the file containing the form object must have the same name as the form itself (with the .cs extension).
  - See Changing Form Names (PL-Beginning Unit 2).
- Each form must include a form icon (not the default icon).
  - See Forms (PL-Beginning Unit 2).
- Each project icon must be set (not the default icon).
  - The project icon may be the same as a form icon but this is not required.
  - See Project Icons (PL-Beginning Unit 2).
- Define an appropriate title for each form (correct spelling and spacing).
  - Use Title Case with appropriate spacing
  - See Forms (PL-Beginning Unit 2).
- Forms centered
  - Most forms should appear centered on the screen.
  - If you choose to break this standard, be prepared to explain your reasons.
  - See Form – Start Position (PL-Beginning Unit 2).

- Programming Logic – Beginning Only
  - Each form should have a designated Accept and Cancel button.
  - All form buttons should designate an access key.