

Programmazione

Prof. Marco Bertini

marco.bertini@unifi.it

<http://www.micc.unifi.it/bertini/>



Coding style guidelines

“Good code is its own best documentation.”
- Steve McConnell





Why using a coding standard ?

- A coding standard may help to reduce errors due to poorly written code, i.e. code that uses programming facilities in (unnecessarily) error-prone way or that expresses ideas in obscure ways.
 - As noted by Guido van Rossum (creator of Python language): code is read much more often than it is written.
 - There's no standard coding standard.
-



Consistency

- A style guide is about consistency. Consistency with a style guide is important. Consistency within a project is more important. Consistency within one module, class or function is the most important.
- However, know when to be inconsistent - sometimes style guide recommendations just aren't applicable.



Classes and Objects

- Names representing types (i.e. classes) and namespaces must be in mixed case starting with upper case, e.g.:

Line, SavingsAccount

- Variable names must be in mixed case starting with lower case, e.g.:

line, savingsAccount



Classes and Objects

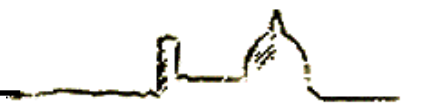
- Names representing types (i.e. classes) and namespaces must be in mixed case starting with upper case, e.g.:

Line, SavingsAccount

- Variable names must be in mixed case starting with lower case, e.g.:

line, savingsAccount

This is the style
enforced in Java



Classes and Objects - cont.

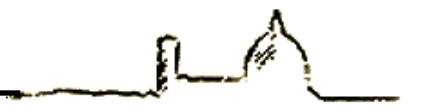
- Bjarne Stroustrup despises this “camel” coding style and in JSF++ proposes the use of underscores, e.g.:

`number_of_elements, Device_driver`

instead of

`numberOfElements, DeviceDriver`

- Suggestion: pick whatever you like and be consistent



Classes and Objects - cont.

- Bjarne Stroustrup despises this “camel” coding style and in JSF++ proposes the use of underscores, e.g.:

`number_of_elements, Device_driver`

This standard is used in Python

instead of

`numberOfElements, DeviceDriver`

- Suggestion: pick whatever you like and be consistent



Classes and Objects - cont.

- The parts of a class must be sorted public, protected and private.
 - All sections must be identified explicitly.
 - Not applicable sections should be left out.
-



Classes and Objects - cont.

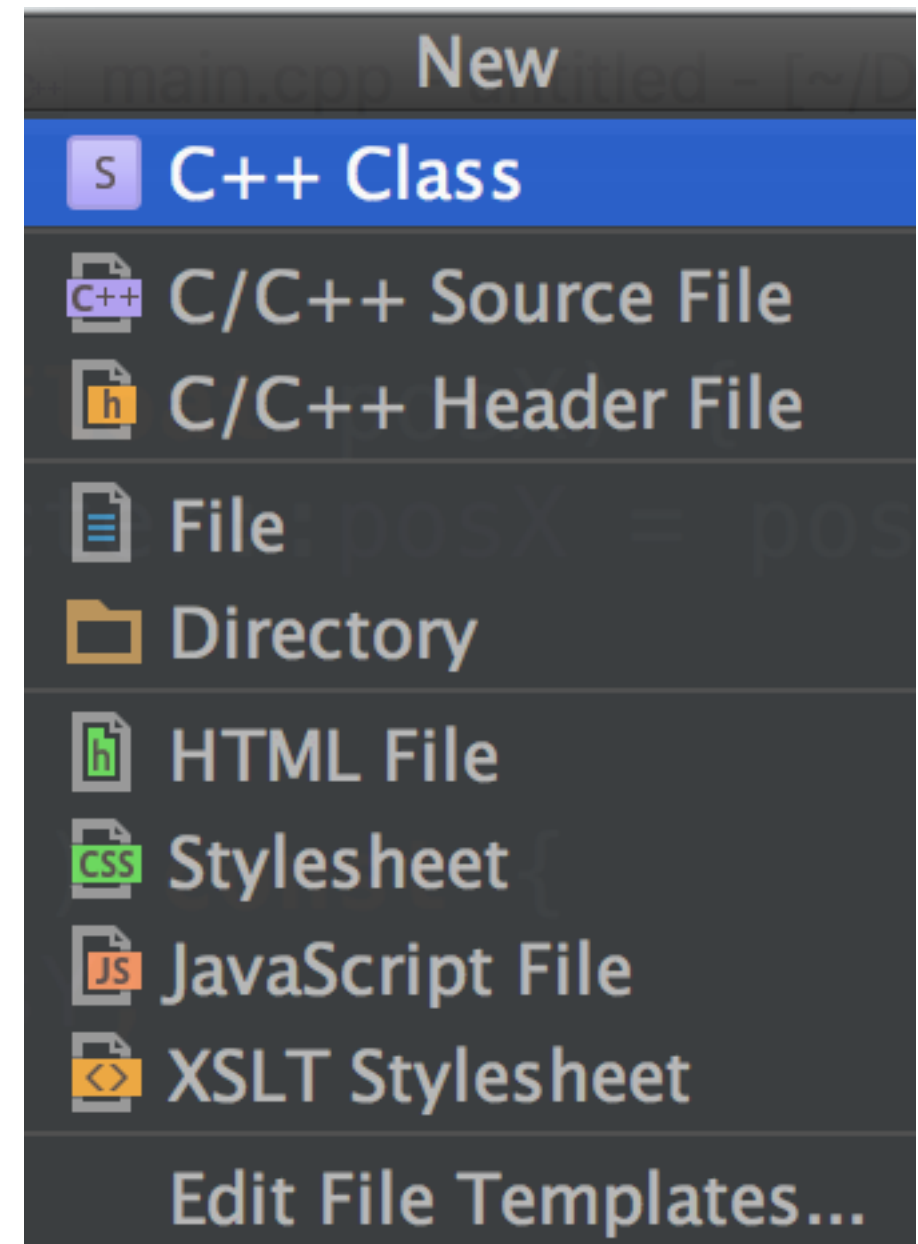
- A class should be declared in a header file and defined in a source file where the name of the files match the name of the class.
- All definitions should reside in source files.

Eclipse CDT let you decide to create the getter/setter as inline methods within the class declaration or in the .cpp file...



Classes and Objects - cont.

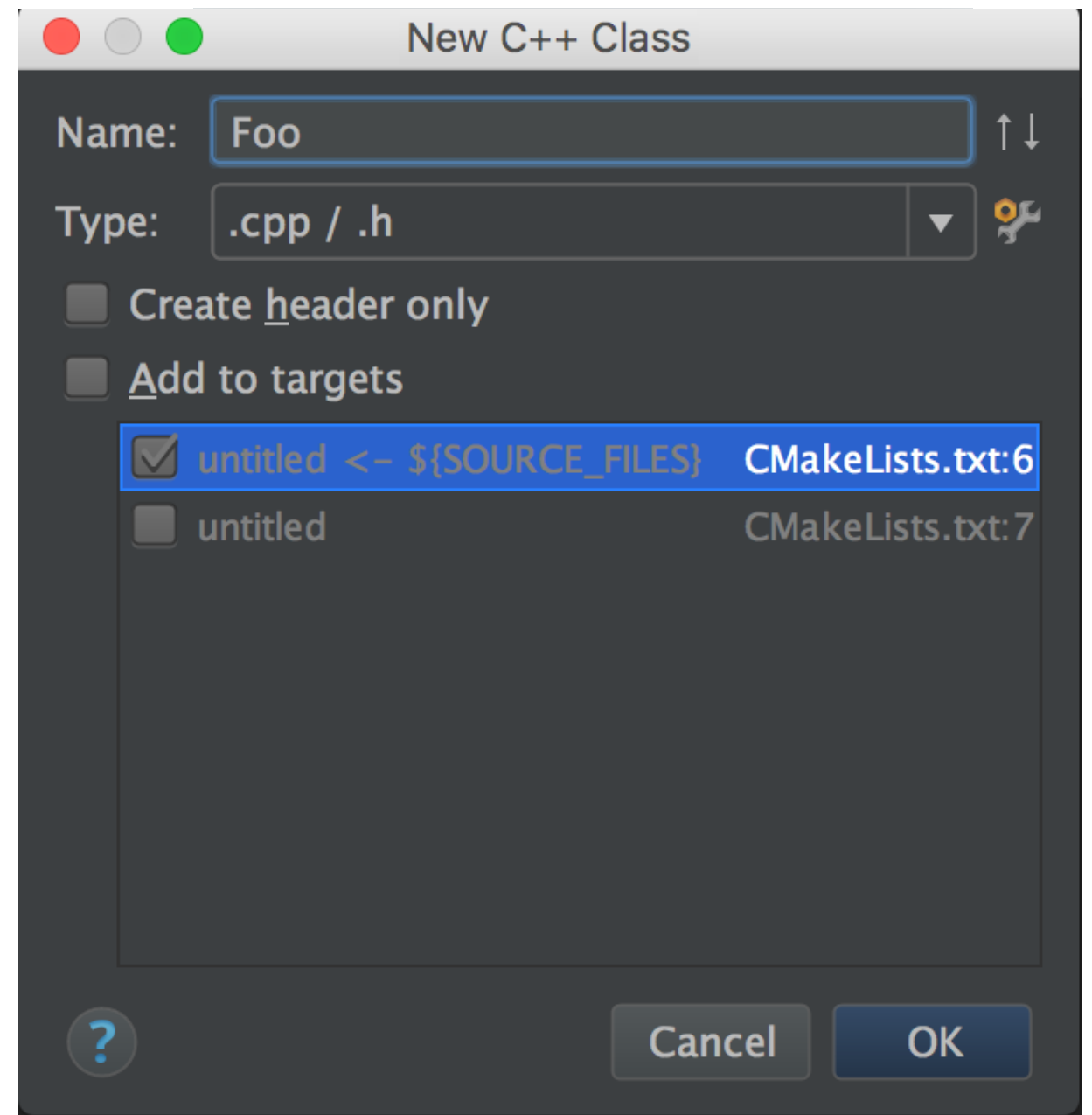
- Many IDEs (e.g. CLion) have a wizard to create classes and follow the `Classname.h + Classname.cpp` approach:





Classes and Objects - cont.

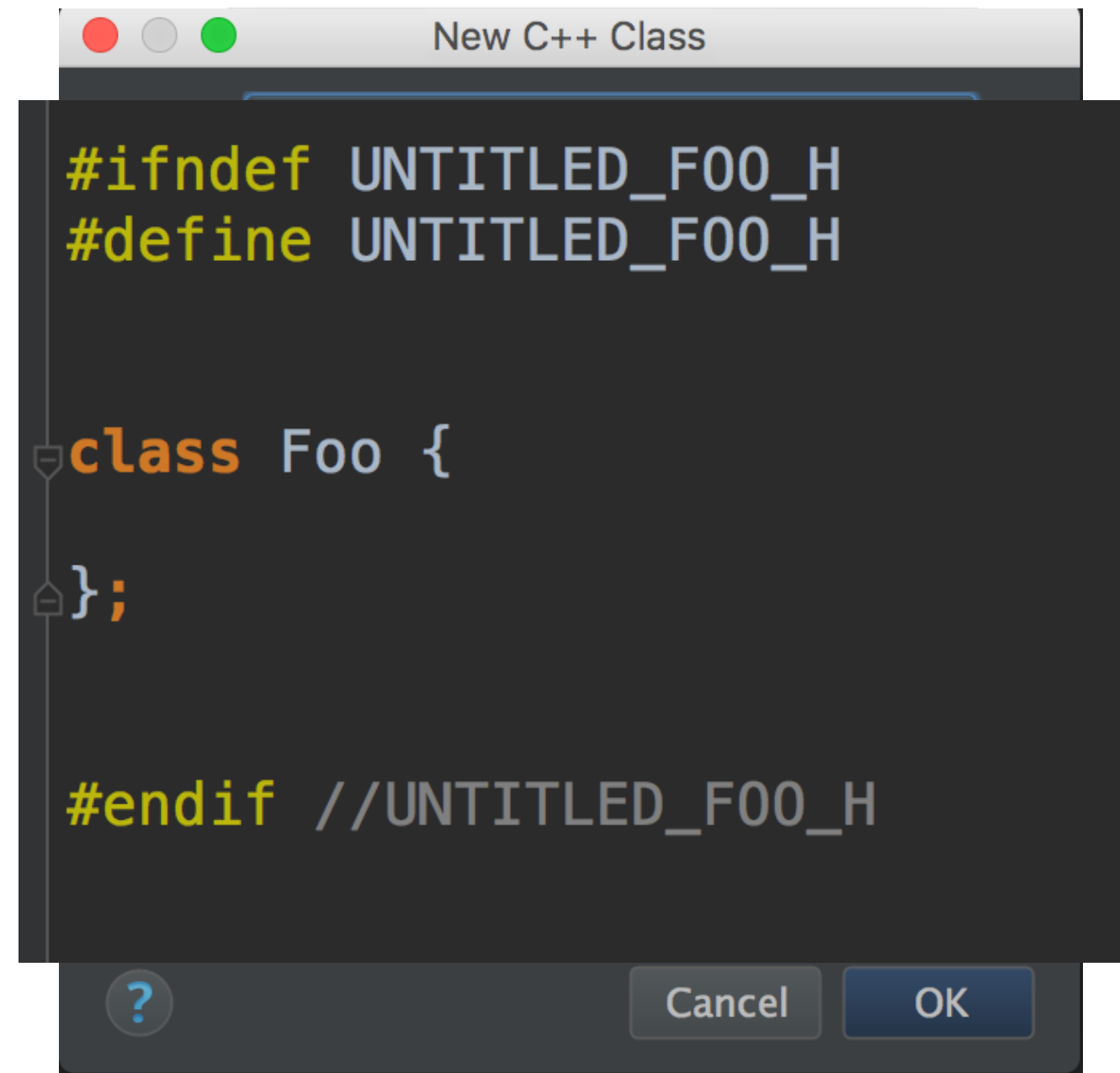
- Many IDEs (e.g. CLion) have a wizard to create classes and follow the `Classname.h + Classname.cpp` approach:





Classes and Objects - cont.

- Many IDEs (e.g. CLion) have a wizard to create classes and follow the `Classname.h + Classname.cpp` approach:



```
#ifndef UNTITLED_F00_H
#define UNTITLED_F00_H

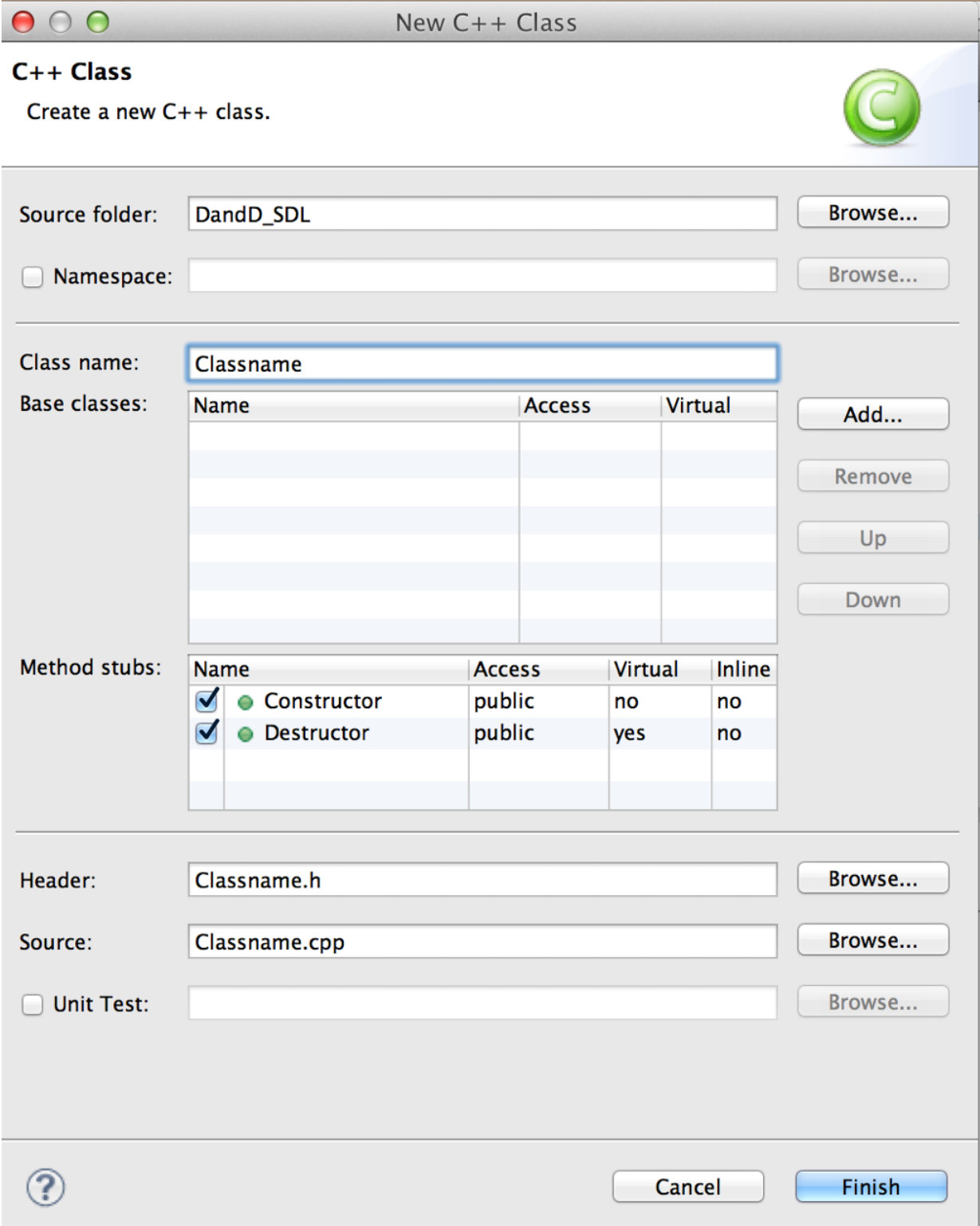
class Foo {
};

#endif //UNTITLED_F00_H
```



Classes and Objects - cont.

- Also Eclipse CDT has a wizard to create classes and follow the `Classname.h` + `Classname.cpp` approach:



The image shows the 'New C++ Class' wizard in Eclipse CDT. The dialog is titled 'New C++ Class' and contains the following fields and options:

- Source folder:** `DandD_SDL` (with a 'Browse...' button).
- Namespace:** (empty field with a 'Browse...' button).
- Class name:** `Classname` (highlighted with a blue border).
- Base classes:** A table with columns 'Name', 'Access', and 'Virtual'. It is currently empty.
- Method stubs:** A table with columns 'Name', 'Access', 'Virtual', and 'Inline'. It contains two entries: 'Constructor' (checked) and 'Destructor' (checked).
- Header:** `Classname.h` (with a 'Browse...' button).
- Source:** `Classname.cpp` (with a 'Browse...' button).
- Unit Test:** (empty field with a 'Browse...' button).

At the bottom right, there are 'Cancel' and 'Finish' buttons. A help icon (?) is located at the bottom left.

Name	Access	Virtual

Name	Access	Virtual	Inline
<input checked="" type="checkbox"/> Constructor	public	no	no
<input checked="" type="checkbox"/> Destructor	public	yes	no



Naming a variable

- The name of a variable should describe fully and accurately the entity the variable represents.
- State in words what the variable represents, probably you'll immediately see a good name.
- Do not be cryptic, do not use strange acronyms



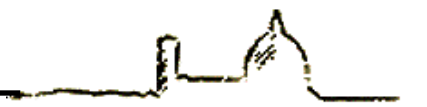
Naming a variable: examples

Purpose of the variable	Good name	Bad name
Current Date	currentDate	CD, current, cD
Lines per page	linesPerPage	LPP, lines, l
Running total of checks written to date	runningTotal, checksTotal, numChecks, nChecks	checks, written, checkTTL, x1



The 2 worst variable names

- “data” is a terrible name: every variable contains data... a variable name should describe what data is contained
- “data2” is another terrible name, like any other `variableX` with $X \in \mathbb{N}$
- rethink what’s the difference w.r.t. `variable` and what it should contain.
Avoid to write code like:
`if(total2 < total3)`



Variables

- Declarations shall be declared in the smallest possible scope:
 - keeping initialization and use close together minimize chance of confusion;
 - letting a variable go out of scope releases its resources.
 - In C++ you can declare a variable wherever you want: do it!
 - Initialize a variable: uninitialized variables are a common source of errors
-



Methods

- Names representing methods or functions must be verbs (followed by an object) and written in mixed case starting with lower case (like Java), e.g.:

`getName()`, `computeTotalWidth()`

- The name of the object is implicit, and should be avoided in a method name, e.g.:

`line.getLength();` // NOT:
`line.getLineLength();`



Methods

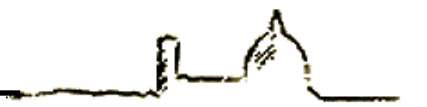
- Names representing methods or functions must be verbs (followed by an object) and written in mixed case starting with lower case (like Java), e.g.:

`getName()`, `computeTotalWidth()`

- The name should be

Alternatively, as in JSF++ standard:
`example_function_name ()`

`line.getLength()`
`line.getLength();`



Methods - cont.

- Use strong verbs, not wishy-washy verbs:
 - OK: `calcMonthlyRevenue()`
 - NO: `handleCalculation()`,
`processInput()`

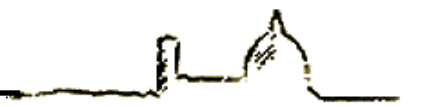


Attributes

- Private class variables often have underscore suffix, e.g.:

```
class SomeClass {  
    private:  
        int length_  
};
```

- This is **HIGHLY** controversial. Other acceptable approaches are: underscore prefix, m_ prefix, no suffix/prefix (use syntax highlighting of the IDE)
-



Numbers

- Avoid “magic” numbers, i.e. numbers that appear in code without being explained
- E.g.:

```
for(int i = 0; i < 255; i++)...
```

versus

```
for(int i = 0; i < maxEntries; i++)...
```



Numbers

- Avoid “magic” numbers, i.e. numbers that appear in code without being explained

- E.g.:

```
for(int i = 0; i < 255;
```

versus

```
for(int i = 0; i < maxEntries; i++)...
```

Consider the case in which the number, used through the code, has to be changed...



String

- Avoid “magic” strings as you avoid “magic” numbers. E.g.:

```
if ( inputChar == '\027' )...
```

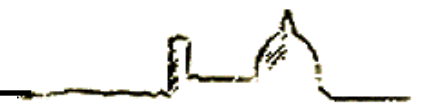
versus

```
if ( inputChar == ESCAPE )...
```



Indentation

- In C/C++ whitespace is insignificant, but indentation of code blocks help readability showing relationships between control flow constructs.
- Can use tabs or spaces: many guidelines suggest spaces, though.
- 1 space is too low, 5 is too much: 2, 3 or 4 are OK.
Suggestion: use 2 or 4 spaces.



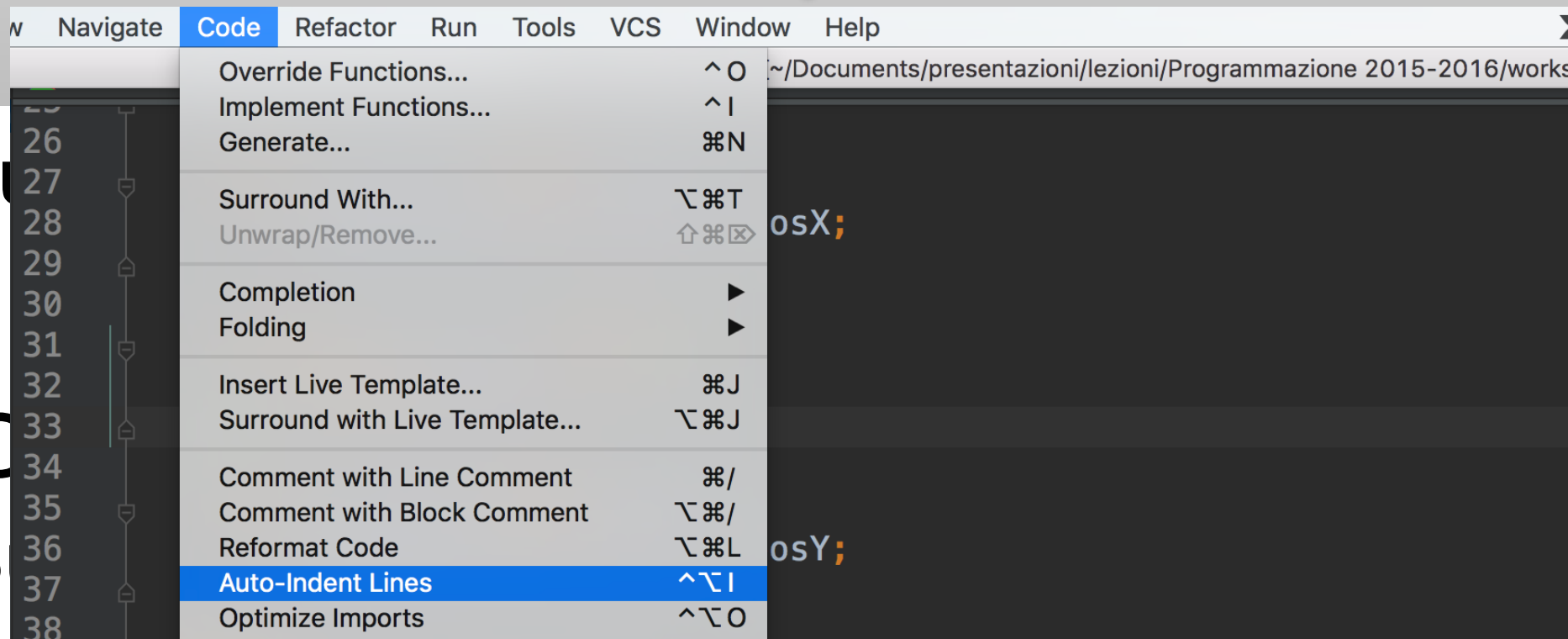
Indentation

Python uses indentation instead of { and } so you better learn to be very precise when indenting code. Python uses spaces not tabs.

Any good editor and IDE will help to indent code while writing, and will re-indent badly written code: learn how to do it.



• I
C
S



re



Layout - cont.

- Use only one statement per line, to improve readability / debugging, e.g.:

```
// NO:
```

```
if ( p > q ) cout << p;
```

```
// OK:
```

```
if ( p > q )
```

```
    cout << p; // notice also the use  
               // of indentation
```



Layout - cont.

- Group lines in “paragraphs” using empty lines
- If there’s need to split a line (some coding standards require a certain length) make it obvious and indent, e.g.:

```
totalBill = shippingCost + customerPurchase[ customerId ] +  
    salesTax;  
drawLine( window.North, window.South, window.East,  
    window.West, currentWidth);
```



Layout - cont.

- Group lines in “paragraphs” using empty lines
- If there’s need to split a line (some coding standards require a certain length) make it obvious and indent, e.g.:

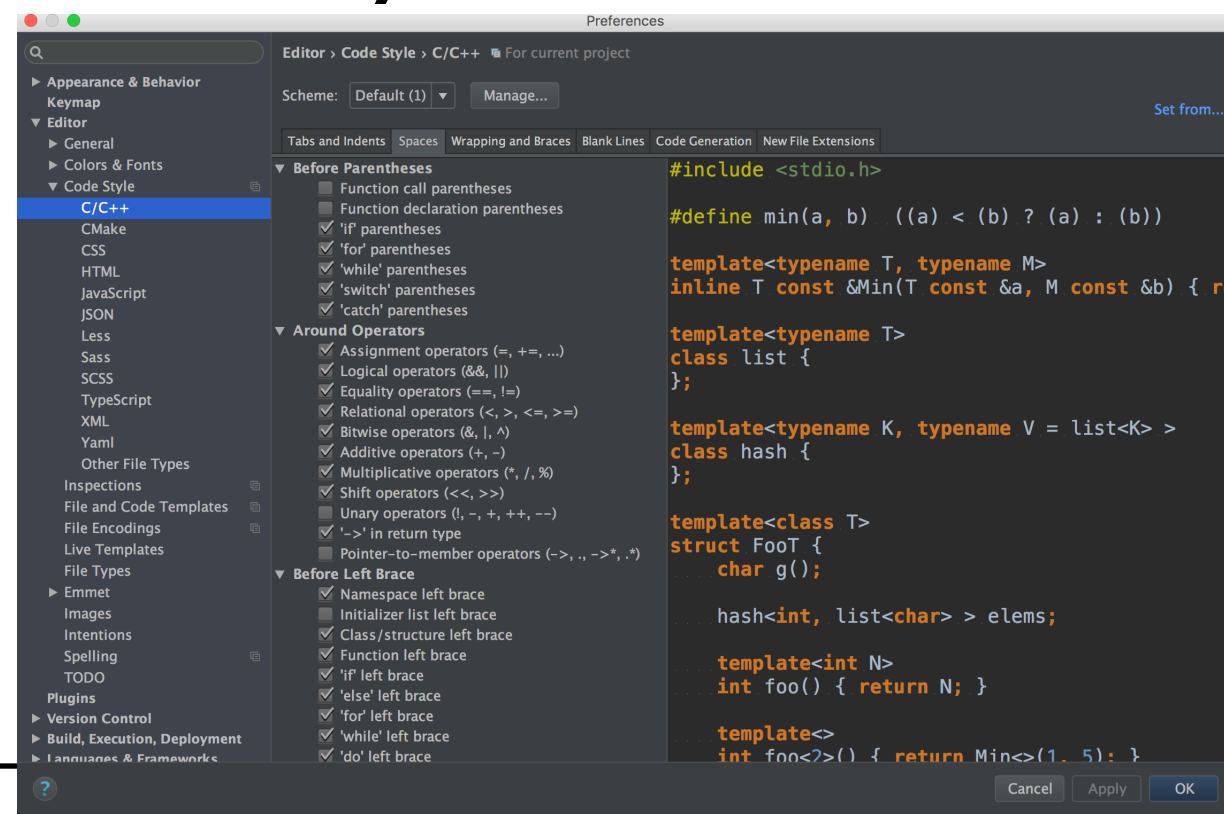
```
totalBill = shippingCost + customerPurchase[ customerID ] +  
    salesTax;  
drawLine( window.North, window.South, window.East,  
    window.West, currentWidth);
```

+ and , signal that
the statement is not
complete



Format

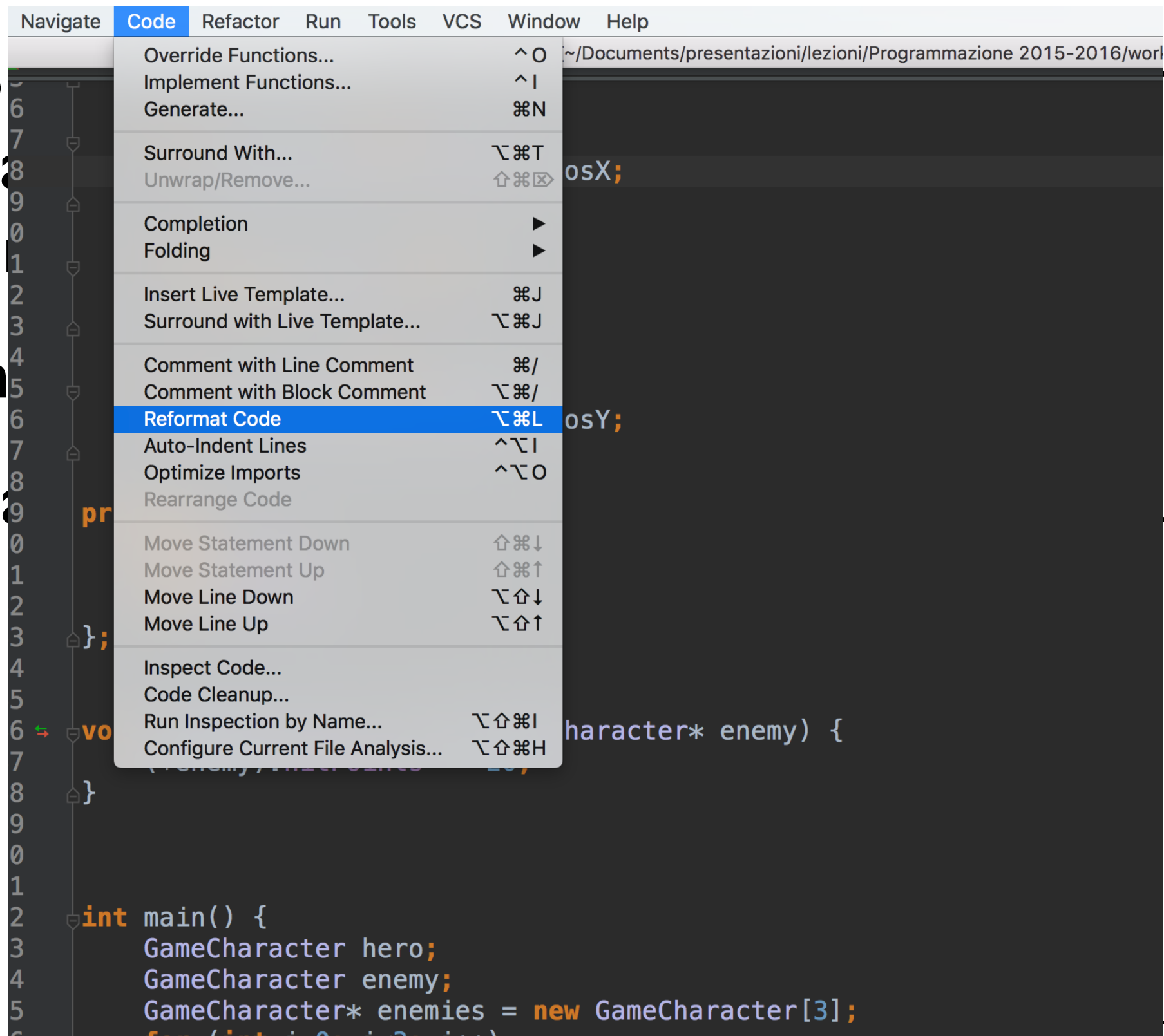
- Formatting code is more than just indenting it. Spaces, wrapping and braces, blank lines all contribute to improve readability.
- There are several standards: choose one.
- Learn how to fully reformat code with your IDE.





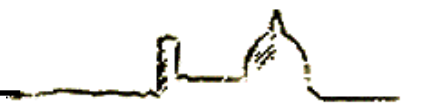
Format

- For
- Space
- Com
- Th
- Lea



ing it.

ur IDE.



Comments

- Describe code intent, e.g.:

```
// get current employees info
```

instead of

```
// update EmpRec vector
```

- Do not repeat the code, e.g.:

```
delete aVehicle; // free
```



Code can only tell you how the program works;
comments can tell you why it works.

- Describe code intent, e.g.:

```
// get current employees info
```

instead of

```
// update EmpRec vector
```

- Do not repeat the code, e.g.:

```
delete aVehicle; // free
```



Preprocessor

- Do not use macros except for source control, using `#ifdef` and `#endif`
 - macros don't obey scope and type rules and make code hard to read. All that can be done with macros can be done using C++ features
 - `#includes` should precede all non-preprocessor declarations
 - nobody will notice the `#include` in the middle of a file
-



Preprocessor and includes

- A suggested order of inclusion (Google's C++ guideline) is:
 - the header of the file
 - C library
 - C++ library
 - other libraries' .h
 - your project's .h.
-



Preprocessor and includes

- A suggested order of inclusion (Google's C++ guideline) is:
 - the header of the file
 - C library
 - C++ library
 - other libraries' .h
 - your project's .h

E.g., in fooserver.cpp:

```
#include "foo/public/fooserver.h"
```

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
#include <hash_map>
```

```
#include <vector>
```

```
#include "base/basicctypes.h"
```

```
#include "base/commandlineflags.h"
```

```
#include "foo/public/bar.h"
```



Credits

- These slides are based on the material of:
 - C++ Programming Style Guidelines
Geotechnical Software Services
<http://geosoft.no/development/cppstyle.html>
 - “Code Complete”, Steve McConnell,
Microsoft Press
 - JSF++ coding guidelines
 - Python PEP-8 guideline
-