M2 vs. C++ vs. Java vs. Python: Access / Visibility Control

16 Jan 2009
CMPT166
Dr. Sean Ho
Trinity Western University

Quiz 1 (10min, 20pts)

- Describe and contrast the roles of the following: preprocessor, compiler, linker. [6]
- Name 6 out of the 8 types built-in to C++ [6]
- #include <iostream> vs. #include “mylib.h” [4]
    - What's the difference between <> and “”?
- Write a complete C++ program that prints “Hello, CMPT166!” to the console. [4]
    - Docstring, comments not necessary

Quiz 1 answers: #1-2

- Preprocessor vs. compiler vs. linker:
  - Preprocessor: #includes, text macros
  - Compiler: translates from high-level code (*.cpp) to machine code to be executed (*.o)
  - Linker: connects several compiled object files (*.o) and libraries into one executable (*.o)
- Name 6 out of the 8 types built-in to C++:
  - char, bool
  - short, int, long int
  - float, double, long double

Quiz 1 answers: #3-4

- #include <iostream> vs. #include “mylib.h”
  - Angle brackets <iostream> tell the pre-processor to look in standard library directories to find the file iostream.h
  - Quotes “mylib.h” tell the pre-processor to look in the current directory
- “Hello, CMPT166!”
  - **#include <iostream>**
  - **using namespace std;**
  - **int main() {**
    - **cout << “Hello, CMPT166!” << endl; }**

Declaring classes: OO-M2

- Declaring a class in object-oriented M2:

```
CLASS Rectangle;
    CONST
        sides = 4;
    VAR
        length, width: INTEGER;
    PROCEDURE SetDims (l, w: INTEGER);
    BEGIN
        length := l;
        width := w;
    END SetDims;
BEGIN
        SetDims (0, 0);
END Rectangle;
```

Declaring classes: C++

- Header (public definition) file:

```
class Rectangle {
    const int sides = 4;
    int length, width;
    void SetDims (int l, int w);
}
```

- Code (private implementation) file:

```
void Rectangle::SetDims (int l, int w) {
    length = l;
    width = w;
}
```

Declaring and instantiating objects

- Instantiating allocates memory and calls constructor
- OO-M2:

```
VAR
    rect : Rectangle;
BEGIN
    CREATE(rect);
```

- C++/Java:

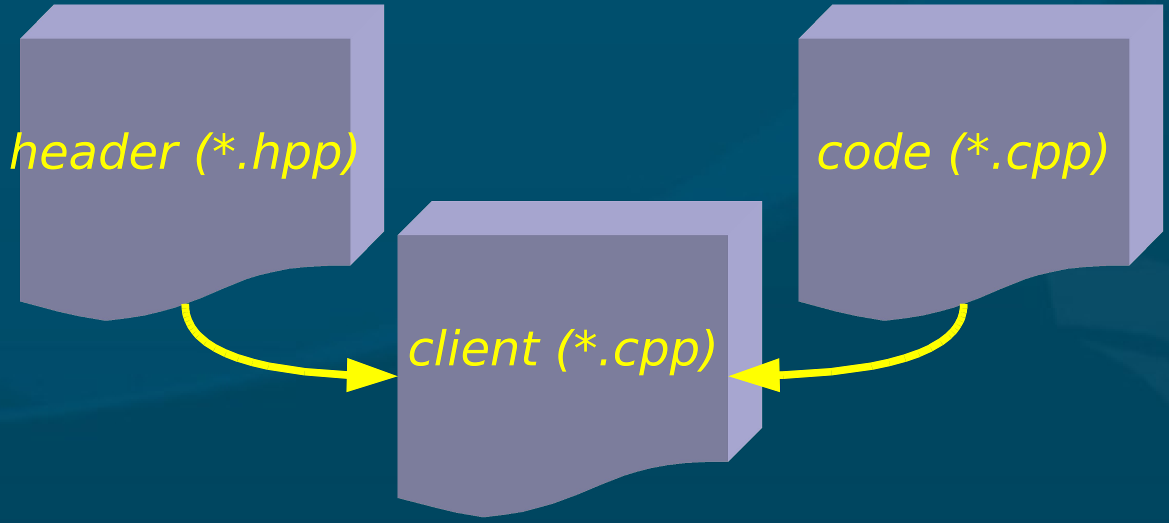
```
Rectangle rect;
rect = new Rectangle();
```

- Python:

```
rect = Rectangle()
```

Header files and visibility

- M2 and C++ put header (DEF) and code (IMP) in separate files
- Anything in a M2 DEF file is visible to any client that imports the library
- Anything in a C++ header file is visible to any client that includes the header



Access / visibility control

- Access modifiers limit who can see variables and methods:
  - public: anyone who imports this class
  - private: only methods within this class
  - protected: subclasses of this class
- You can also grant a specific function or class access to your private attributes/methods by declaring it a *friend*:
  - **class MyClass {**
    - **int mySecretInt;**
    - **friend void otherFunction();**

Access control in OO-M2

- To make something public, mark it with REVEAL
- You may also mark items as READONLY
- Everything else is protected by default

```
CLASS Account;
    REVEAL credit, debit, READONLY balance;
    VAR
        balance : REAL;
    PROCEDURE credit (amount : REAL);
    PROCEDURE debit (amount : REAL);
END Account;
```

- Make things private by hiding them in IMP file

Access control in Java

- Java uses public/private/protected keywords just like C++, but applied to each item instead of in sections:

```
    public class Account {
        float balance;          // default is package visibility
        private boolean overdrawn;
        public Account() {balance = 0;}   // initializer
        public void credit (float amount) {
```

- Designate immutable items with final (C++: const)

- Python: __*names* are private; all others public

Access control in C++

- Members are grouped under headings: *public*, *private*, *protected*

```
class Account {
    public:
        float balance;
        void credit (float amount);
        void debit (float amount);
    private:
        bool overdrawn;
}
```

- In code file:

```
Account::credit (float amount) {
    …
```

public/private keywords

- So far most of our classes/attributes/methods have been declared public
- The private keyword specifies that only methods within this class can access this entity:

```
class Student {
    private String name;
}
Student s1 = Student();
s1.name;    // error!
```

- This is for information hiding: prevent others from directly accessing/modifying an entity.

Set/get methods

- A common idiom is to declare instance variables private but provide public set/get methods:

```
class Student {
    private String name;
    public String getName() { return name; }
    public setName(String n) { name = n; }
}
```

- Advantages of set/get over just declaring public?
  - Control access to the instance variable
    - Can add error checking
  - Hides underlying storage type of variable
    - Can upgrade to different data structure later