

M2 vs. C++ vs. Java vs. Python

Access / Visibility Control

16 Jan 2008

CMPT166

Dr. Sean Ho

Trinity Western University

Modules vs. Classes

- Both **modules** (M2) and **classes** (OO):
 - Have both a **public** interface (DEF) and a **private** implementation (IMP)
 - Allow **data hiding** (in the private portion)
- But there are **differences**:
 - Data items in modules are **singletons**;
 - ◆ Each **instance** of a class has its **own** data items
 - Modules in M2 are not **types**; OO classes are
 - Modules cannot be **derived** from other modules
 - ◆ Classes can **inherit** (subclass) from other classes

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;
```

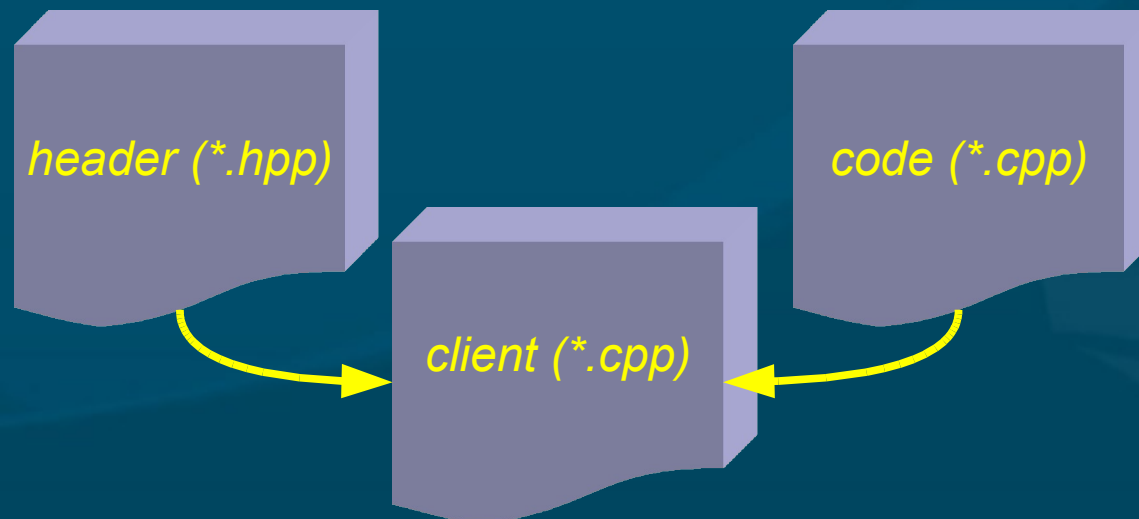
```
rect = new Rectangle();
```

- 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
 - **(default)**: anything in the same **package**

- ◆ C++ terminology: **friend**

	Class	Package	Subclass	World
Private	Y	N	N	N
(none)	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

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 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) {
```

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**

Java packages



- **Group** related classes and interfaces
- Avoids name **collision**
- Package **declaration** at top of each file:
 - ◆ `package mypackage;`
- Popular convention: use reverse **domain** name
 - ◆ `com.sun.java.awt...`
 - ◆ `ca.twu.cmpt167.lab3.seanho.FractalTree`
- Pass “-d” option to `javac` to create **directories** when compiling:
 - ◆ `javac -d . FractalTree.java`

Using packages

- Every **file** should specify what **package** it belongs to in the **first line** of code in the file
- Each file should still have only **one public** class
 - **Non-public** classes have **package** scope
 - ◆ Useful for **internal** helper classes
- **Import** from a package as normal
 - **Classpath** specifies where to search for packages
 - ◆ **Default** classpath includes “.”
 - ◆ **Override** with **java -classpath ./other/path**



jar

- Wrap up a **collection** of related classes/packages into one file with **jar** (**Java ARchiver**)
 - Like **ZIP**, Unix **tar**
- Syntax:
 - **Create** a jar file: `jar cvf mypackage.jar <files>`
 - **Unpack** a jar: `jar xvf mypackage.jar`
 - ◆ **C**: create
 - ◆ **X**: extract
 - ◆ **V**: verbose
 - ◆ **F**: specify jar file

