

Classes and Access Control in Java / C++ / Python / OO-M2

18 Jan 2010

CMPT166

Dr. Sean Ho

Trinity Western University

Declaring classes: Python

- Python:

```
class Rectangle:
```

```
    def setDims (self, l, w):
```

```
        self.length = l
```

```
        self.width = w
```

```
    def __init__ (self, l=1, w=1):
```

```
        self.setDims( l,w )
```

```
        self.__SIDES = 4
```

- Specify private/hidden attributes with '__' prefix
- Specify constants in all-caps: convention
 - No way to enforce constants

Declaring classes: Java

- Java:

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

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 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, instantiating objects

- Instantiating **allocates** memory and calls constructor

- OO-M2:

VAR

```
rect : Rectangle;
```

BEGIN

```
CREATE(rect);
```

- C++/Java:

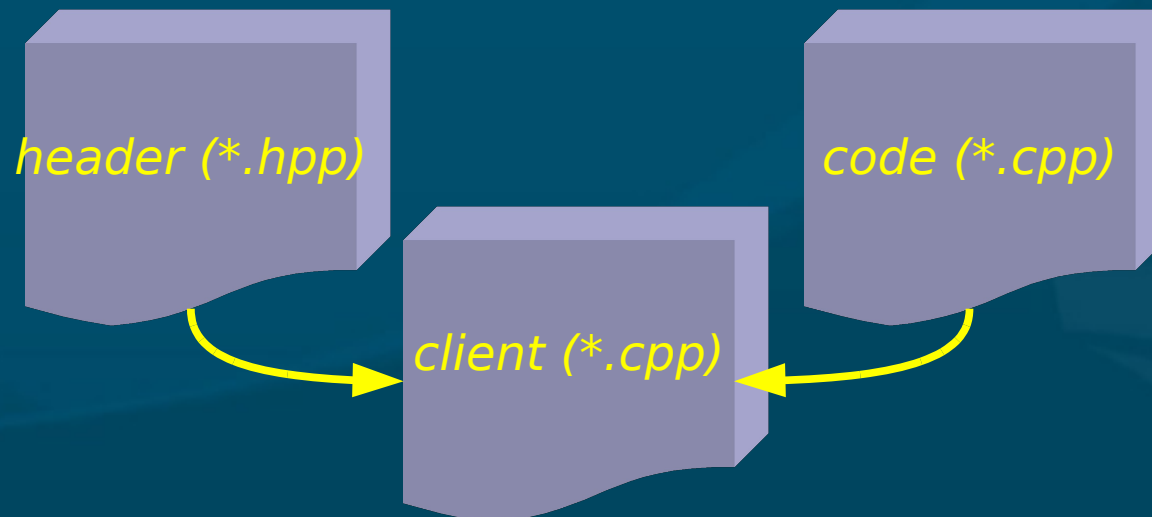
```
Rectangle rect;
```

```
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

	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 Java, Python

- Java uses **public/private/protected** keywords applied to each **item**:

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

- Designate **constant** items with **final**
- 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) {
```

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

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

- Commonly, 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; }  
}
```

- **Why** use **set/get** instead of 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

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: `jar cvf mypackage.jar <files>`
 - **Unpack** a jar: `jar xvf mypackage.jar`
 - ◆ **C**: create
 - ◆ **X**: extract
 - ◆ **V**: verbose
 - ◆ **F**: specify jar file

