Defining Classes, Access Modifiers and Packages

18 Jan 2010
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Outline for today

- Formatted output: System.out.printf()
- Declaring classes
  - Python vs. Java vs. C++ vs. OO-M2
  - Creating instances of classes
- Access modifiers: public, private, protected
  - Python vs. Java vs. C++ vs. OO-M2
  - Set/get methods
- Java packages
  - jar

Console output: System.out

- System.out is the standard output channel
  - Default is console
  - But can be redirected to a file
- Methods of System.out
  (and other output channels):
  - .print(*str*): output *str* exactly as-is
  - .println(*str*): output *str* plus a newline
  - .printf(*fmt*, *arg1*, *arg2*, ...):
    use a format string (with %d, %s, etc.)
- Or use formatter objects

Formatted output: printf

- .printf() uses a format string just like Python:
  - **System.out.printf("I have %3d apples\n", numApples);**
- Format specifiers (with optional field width):
  - %d: integer (%3d, %03d, %-3d, %-03d)
  - %f: float (%5f, %5.1f, %05.1f, %-05.1f)
  - %e: scientific-notation float, e.g., 1.23e4
  - %s: string (%12s)
  - %c: character (%-2c)

Declaring classes: Python

- Python:

```
class Rectangle:
    def __init__ (self, l=0, w=0):
        self.setDims( l, w )
        self.__SIDES = 4
    def setDims (self, l, w):
        self.__length = l
        self.__width = w
```

- Specify private (hidden) attributes with '__' prefix
- Specify constants in ALL_CAPS by convention
  - No way to enforce constants

Declaring classes: Java

- Java: this would go in Rectangle.java:

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

Declaring classes: C++

- Header (public definition) file: Rectangle.h

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

- Code (private implementation) file: Rectangle.cpp

```
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

- Allocate memory and call constructor
- C++/Java:

```
Rectangle rect;
rect = new Rectangle();      // 'new' optional in Java
```

- Python:

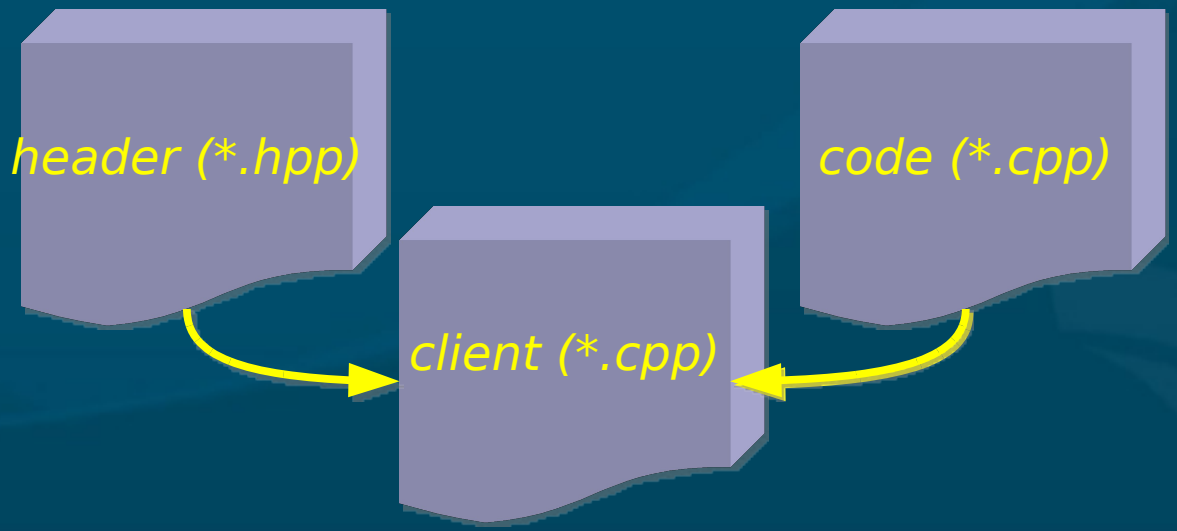
```
rect = Rectangle()
```

- OO-M2:

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

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
(default)	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() {              // constructor
        balance = 0.;
    }
    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

Private attributes

- 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.cmpt166.seanho.lab3.BankAccount
- Pass "-d" option to javac to create directories when compiling:
  - javac -d . BankAccount.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
    - Directories separated by colon ":"
    - Default classpath includes "."
    - Override with java -classpath .:/other/path

jar

- Wrap up a collection of related classes/packges 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

TODO

- By Today: Eclipse tutorial
  - Get familiar with a Java development environment: Eclipse, NetBeans, or other
  - Write a simple "Hello, World!" program
  - Nothing to turn in
- Lab1 (due this Thu): Control/Flow
  - Savitch text, pp.162-164. Choose one of:
  - #2: game of craps
  - #5: loan calculator
  - #8: cryptarithmetic puzzles