# Polymorphism, References, and the Copy Constructor

***DogsAndCats example***

30 Jan 2009
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

# CAREER FAIR

**Event Details**
Feb. 5, 2009
11:30 am – 3:00 pm

Larson Atrium
Reimer Student Centre,
TWU Campus
7600 Glover Road
Langley, BC V2Y 1Y1

*Attire:*
Business Professional

Open to all TWU Students and Alumni.

You are invited to attend the 2009 TWU Career Fair. The Fair is a chance to meet prospective employers for full-time careers, internships, co-ops, and seasonal positions. Check website for participating employers. **Register at the Student Portal.**

**www.twu.ca/life/career/fair**
**604-513-2017**
**career@twu.ca**

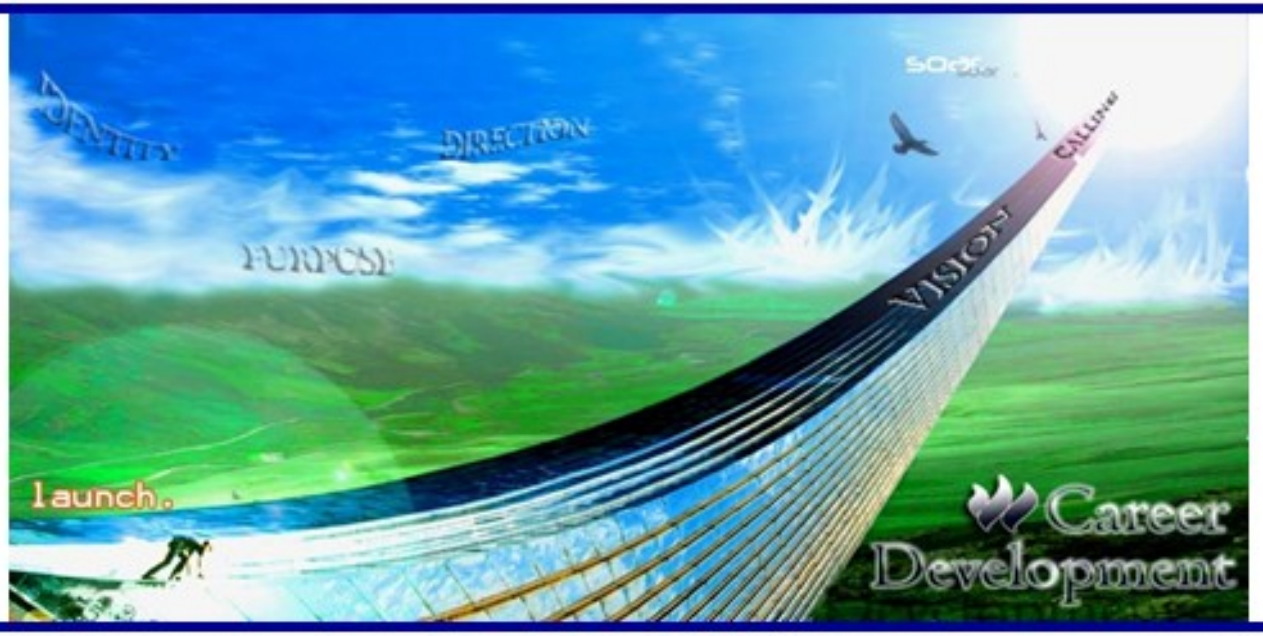
# 2009 Career Fair Exhibitors

- Acts Seminaries
- Adventure Teaching
- African Children's Choir
- Angus One Professional Recruitment
- BC Corrections
- BC Ferries
- BC Human Resources Management Association
- BC Transit Police
- Bethesda Christian Association
- BOP Korea Connections
- Canada Revenue Agency
- Canadian Forces
- Certified General Accountants
- Certified Management Accountants
- Communitas Supportive Care Society
- Community Employment Resource Centre
- Corporate Express
- Correctional Service Canada
- Creative Memories
- Developmental Disabilities Association
- Dulay Burke Financial Recruitment
- Edward Jones
- EV Logistics
- Fraser Health Authority
- Freedom 55 Financial
- Institute of Chartered Accountants of BC
- Logos Bible Software
- Kintec Footlabs
- Meyers Norris Penny LLP
- New Westminster Police
- Northern Health
- Pampered Chef
- Power to Change
- Retirement Concepts
- Royal Bank of Canada
- Royal Canadian Mounted Police
- Southwestern Company
- Studibudi Professional Networking Inc.
- Sun Life Financial
- TD Canada Trust
- Township of Langley
- UHY International
- Vancouver Fire and Rescue Services

TRINITY WESTERN UNIVERSITY

TRINITY WESTERN UNIVERSITY

Career Fair 2009 — February 5, 11:30am-3:00pm
Larson Atrium, Reimer Student Centre
**www.twu.ca/life/career/fair**




Pre-Career Fair Workshops: to be announced on website.

# Review: inheritance

- "has a" vs. "is a kind of" vs. "knows how to"
- public/private/protected
- Constructors, calling superclass constructor
- Overloading functions

# What's on for today

- Upcasting
- Virtual methods
- Abstract superclasses and pure virtual methods
- References, pass-by-reference, const refs
- The copy constructor
- Operator overloading

# Upcasting

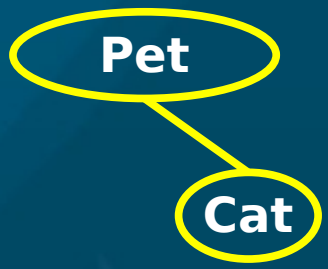
- A reference to an instance of a subclass may also be treated as an instance of the superclass
  - class Dog : public Pet { ...
  - Dog fido
  - Every Dog is also a Pet
- Pointer to fido:
  - Pet* myPetPtr = &fido;
  - This assignment works!
  - "forgets" the object is a Dog, only thinks of it as a generic Pet

# Virtual methods

- A subclass can redefine a method defined by the superclass
  - Every Pet knows how to speak()
  - But Dogs speak() differently from Cats
  - Subclasses overload speak()
- Flag the method as virtual in the superclass
- Late binding: which version of speak() to use?
  - Decided at run-time
- Polymorphism: same code works on several different types, all subclasses of the same parent

# Polymorphism

- Think carefully about class hierarchy in program design
- Write programs/algorithms to operate on superclass objects
  - As generic as possible
- Instances of subclasses can be operated on by the algorithms without need for modification
- Dynamic method binding:
  - C++ runtime chooses correct method (e.g., speak() ) from subclass



# Abstract superclasses

- Sometimes you may want to make a superclass as a category to cover many subclasses, but don't want to allow instantiation of superclass
- Make a pure virtual function: declare as '= 0':
  - **virtual void myfun() = 0;**
- The compiler will prevent anyone from instantiating this class: abstract superclass
- Subclasses will need to override this method and provide actual bodies to the method

# References vs. pointers

- References are like constant pointers that are automatically dereferenced by the compiler
  - Must be initialized when created
  - Always refers to same object
  - No such thing as a NULL reference
- Equivalent to Python alias
  - `int a = 5;`
  - `int& ref = a;`
  - `ref++;  // a is now 6`

# References and functions

- References are often used to pass an object to a function, or return an object from a function:
  - C only has call-by-value, so use pointers:
    - `int a = 5;`
    - `void dbl_me(int* x) { (*x) *= 2; }`
    - `dbl_me(&a);  // pass a pointer`
  - C++ allows call-by-reference:
    - `void dbl_me(int& x) { x *= 2; }`
    - `dbl_me(a);  // call-by-ref`

# const references

- Since a function may modify its argument if passed by reference, it fails on const references
  - `void dbl_me(int& x) { x *= 2; }`
  - `dbl_me( 5 );` `// won't work!`
  - `const int& pi = 3;`
  - `dbl_me( pi );` `// won't work!`
- If we're not going to modify the argument, declare it as a const reference:
  - `void dbl_me(const int& x) { cout << x; }`
  - `dbl_me( 5 );` `// will work`
- Get into a habit of making parameters const refs unless you need to modify the argument

# The copy constructor

- References are aliases: point to same object
- How about copying?
- C++ default way of copying is a low-level bitcopy: copies pointer addresses! (shallow)
- You can define your own copy constructor:
  - **class MyClass {**
    - **int attrib;**
    - **MyClass( const MyClass& x ) {**
      - **attrib = x.attrib; ....**
- Needed if you want to pass your object by value! Otherwise, must pass by reference

# Operator overloading in C++

- Just as in Python, operators may be overloaded: define specially-named methods:

```
class Nation {
    int pop;
public:
    const Nation operator+(const Nation& n) {
        return Nation(pop + n.pop);
```

- Note return line instantiates a new temporary object inline

```
Nation x, y;
x = x + y;
```