C++ for Gamebryo: Review for the Introductory/Beginners’ Track

References:

- Ivor Horton’s Beginning Visual C++ 2008 (available on-line) and
- Programming (Principles and Practice Using C++) by Bjarne Stroustrup.

Lecture Five: Inheritance

```cpp
#include <iostream>
using namespace std;

class Horse {
public:
    void talk() {
        cout << "Howdy! I'm a Horse." << endl;
    }
};

int main() {
    Horse a;
    a.talk();
}
```

Basic class description of a Horse.

```cpp
#include <iostream>
using namespace std;

class Horse {
public:
    void talk() {
        cout << "Howdy! I'm a Horse." << endl;
    }
};

class Unicorn : public Horse {
};

int main() {
    Horse a;
    a.talk();
    Unicorn b;
    b.talk();
}
```

A Unicorn is a Horse with a horn. Here a Unicorn is a Horse with (so far) nothing else.
Basic polymorphism: a Unicorn is a Horse. But what if we want to get the French greeting at [1]?

Declaring the functions virtual ensures that they will be selected by the object (not reference) type.
Define a few classes describing objects with an area in the plane: Circle, Rectangle, Triangle. In your main program create a few objects of this kind (a few Circles, a few Rectangles and a few Triangles) and store them in a vector. Then go through the vector and ask the objects to report their areas.

```cpp
#include <string>
#include <vector>
#include <iostream>

using namespace std;

class Shape {
public:
    virtual string area() {
        return "I'm just a generic shape...";
    }
};

class Circle : public Shape {
public:
    virtual string area() {
        return "I'm a circle.";
    }
};

class Rectangle : public Shape {
public:
    virtual string area() {
        return "I'm a rectangle.";
    }
};

class Triangle : public Shape {
public:
    virtual string area() {
        return "I'm a triangle.";
    }
};

int main() {
    Triangle* t = new Triangle();
    Circle* c = new Circle();
    Rectangle* r = new Rectangle();
    vector<Shape*> shapes;
    shapes.push_back(t);
    shapes.push_back(c);
    shapes.push_back(r);
    for (unsigned int i = 0; i < shapes.size(); i++)
        cout << shapes[i]->area() << endl;
}
```

Dynamic method lookup for specific shapes. Does it make any sense to call area() on a generic shape?