C++ (classes)

Hwansoo Han
Inheritance

- Relation among classes
  - shape, rectangle, triangle, circle, …
Base Class: shape

- Members of a class
  - Methods: rotate(), move(), Shape(), ~Shape()
  - Attributes: _center, _color

```cpp
class Shape {
    Point _center;
    Color _color;

public:
    Shape (Point& pos = Point(), Color& c = Black)
    { _center = pos; _color = c; }
    virtual ~Shape(void) { }
    virtual void rotate(double angle) = 0;
    virtual void move(Point& p) { _center = p; }
};
```
Subclass: triangle

- A new class is derived from an existing class
  - New class ← existing class
    
    \[
    \text{subclass} \leftarrow \text{base class}
    \]
  
- Inherits all attributes and methods from the base class
- Virtual functions can be defined differently in subclasses

```cpp
class Triangle : public Shape {
private:
    Point vertices[3];

public:
    Triangle (Point p[3]);
    virtual ~Triangle(void) { ... };
    virtual void rotate(double angle) { ... };
    virtual move(Point& p) { vertices[0] = p; };
}
```

Method list
- Triangle();
- ~Triangle();
- rotate();
- move();
Virtual Functions

- Methods in C++ are non-virtual by default
  - Derived subclass has the same functions as the base class
  - Cannot be changed in subclass
  - [NOTE] methods in Java are all virtual by default.

- Virtual methods are specified with a virtual in C++
  - virtual func_name (arg_list) { ... };
  - Virtual methods can be changed in subclass
    - Virtual means “overridable”

- E.g. move(), rotate() in Shape and Triangle
Abstract Class

- Base class `shape` is an abstract class
  - At least one “pure virtual method” (abstract method) exists
    - `virtual func_name (arg_list) = 0;`
  - Abstract method MUST be virtual
  - Abstract method MUST be defined in a subclass

- Cannot instantiate an object of an abstract type
  - `Shape a_shape; // ERROR`
  - `Triangle a_triangle; // OK`
Constructor

- A special method
  - Constructor is called to "initialize" when an object is created
    - `classname(arg_list) { … };`
  - Cannot be virtual,
  - Cannot return any value,
  - Automatically called
  - Can have several constructors with different argument types

```cpp
class Triangle: class Shape {
private:
    Point vertices[3];

public:
    Triangle (Point& p[3]) { … };  
    Triangle (Point& c) vertices({c,c,c}) { … };  

...  
}
```
If you omit constructor, compiler provides a default one

\[
\text{Triangle::Triangle( ) } \{ \text{ } \} \]

Once you specify any constructor, default constructor should be provided by programmers

```cpp
class Triangle: {
    ...
public:
    // no constructor
};

int main() {
    Triangle t1; // OK
}
```

```cpp
class Triangle: {
    ...
public:
    Triangle (Point& p[3]) { ... };
    ...
};

int main() {
    Triangle t1; // ERROR
    Triangle t2(p); // OK
}
```
Destructor

- Another special method
  - Destructor is called to “clean up” when its lifetime ends
    - `[virtual] ~classname() { … }`;
    - Can be virtual, if one of other methods is virtual
    - Can have only one destructor, cannot return any value
    - Useful to use if dynamic memory is assigned as a part of the object
    - Destructor is called by “delete”
    - For a local variable, destructor is called when the function returns

```cpp
Point ps[3] = {Point(0,0), Point(1,1), Point(1,0)};

Func() {
    Triangle t(ps);
    Triangle *pT = new Triangle(ps);
    Triangle &rT = new Triangle(ps);
    ...  
    delete pT;
}
```
Scope

- Default scope of fields and methods are *private* in C++
  - [NOTE] In Java, default is package (accessible within a file)

- Three scopes in C++
  - Private: accessible only within the *same class*
  - Protected: accessible from the *same class and subclass*
  - Public: accessible from anywhere
Location of Objects

- Object is allocated in stack or heap
  - Defined as a local variable – e.g. Triangle t;
  - Created with “new” – e.g. Triangle *pT = new Triangle;

- Global object is placed in data section and initialized (with constructor) when program starts
  - Defined as an external variable – e.g. Triangle gT; // global var

- [NOTE] In Java, all objects of non-primitive types are allocated in heap!
  - Primitive types in Java: char, short, int, long, float, double, …
  - No delete – all garbage collected
Keyword **this**

- **Keyword “this”** refers to the object itself
  - Instance variables can be accessed only with their names
  - But they can be hidden by argument names
  - **Keyword “this”** is useful in such case

```cpp
class Point {   // declaration of class Point
private:
    int x, y;

public:
    Point(int, int);
};

Point::Point(int x, int y)  // Point Constructor
{
    this->x = x;
    this->y = y;
}
```
Operator Overloading

- Redefine the meaning of operator for class type objects

```cpp
class Point {  // declaration of class Point
    int x, y;
public:
    ...
    double operator-(const Point & p) const;
};

double Point::operator-(const Point & p) const
{
    double dist;
    int xdiff = x - p.x, ydiff = y - p.y;
    return dist = sqrt( xdiff*xdiff + ydiff*ydiff );
}

int main()
{
    Point p1 (0,0), p2 (1,1); // tow Point objects
    double dist = p1 - p2;   // p1.operator-(p2);
}
```