Inheritance

Creating a Class Hierarchy

CIS 15 : Spring 2007

Functionalia

HW 5 Out.

Due After Midterm 2.

Small Quiz on Thursday (on Classes)

Today:

- HW 5 Concepts
- Advanced Class Topics

Random Number Generation

How does one generate a random number between 1 and 100 (or 0 and 99)?

A computer can only generate ***pseudo-random numbers***.

They are generated via an algorithm that *approximates* the distribution of a random number sequence.

Every algorithm begins with a ***seed*** - a number that acts as the start-state (or first number in the series). From there the algorithm generates a seemingly random series of numbers (which will repeat after some HUGE number of iterations.

```
#include <cstdlib>

...

srand(9179071232);  // Seed the Random Number Generator

cout << rand() << endl;   // Generates a LARGE integer btw 0 and RAND_MAX
```

Random Number Generation

Problem: Every time you run your program with the same **seed** you get the same sequences of number. That's not very random!

Solution: Seed your random number generator with something that changes every time you run your code. (Like the time!)

```
#include <cstdlib>

...

srand(time(NULL));  // Seed the Random Number Generator
                    // with the current TIME

cout << rand() << endl;   // Now you are generating differing LARGE numbers
```

Random Number Generation

Now: How do you change the range of the random numbers you are generating so that it is the range that you want (like 1 to 100)?

Use ***modulus.*** (In C/C++ modulus uses the % character)

(Remember - modulus is the remainder in integer division)

10 % 3 = 1 (because 10 divided by 3 is 3 remainder 1)

```
#include <cstdlib>

...

srand(time(NULL));

cout << rand() % 100 << endl;  // Random numbers btw 0 and 99
cout << (rand() % 100) + 1 << endl; // Random numbers btw 1 and 100
```

Random Number Generation

You can use a random number generator to generate a random string of characters.

Write a for-loop that generates a **MAX_SIZE** number of characters.

```
#include <cstdlib>

#define MAX_SIZE 100
...

char letters[4] = {'A', 'C', 'G', 'T'};

for...
```

Random Number Generation

You can use a random number generator to generate a random string of characters.

Write a for-loop that prints out a **MAX_SIZE** number of characters.

```
#include <cstdlib>

#define MAX_SIZE 100
...

char letters[4] = {'A', 'C', 'G', 'T'};

for(int i = 0; i < MAX_SIZE; i++)
{
  cout << letters[rand() % 4] << endl;
}
```

Function Overloading

What happens in this case?

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

void bark()
{
  cout << "Void bark!" << endl;
}

int bark()
{
  cout << "Integer bark!" << endl;
  return 0;
}

int main() {
   int i = bark();
   bark();
}
```

Function Overloading

What happens in this case?

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

void bark()
{
   cout << "Void bark!" << endl;
}

int bark()
{
   cout << "Integer bark!" << endl;
   return 0;
}

int main() {
   int i = bark();
   bark();
}
```

Compiler Error!

overload.cpp: In function 'int bark()':
overload.cpp:9: error: new declaration 'int bark()'
overload.cpp:4: error: ambiguates old declaration 'void bark()'

Returning a const

Can you return a `const`?

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


const int test_this()
{
  int i = 0;
  i++;
  return i;
}

int main() {
  cout << test_this() << endl;
}
```

Returning a const

Can you return a `const`?

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


const int test_this()
{
  int i = 0;
  i++;
  return i;
}

int main() {
  cout << test_this() << endl;
}
```

Yes. But it is redundant and not-optimal.

Lakos, John. Large Scale C++ Software Design. (pg 618)

Constructor Being Called

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

class Test
{
   public:

     Test(int i = 100, int j = 200)
     {
       cout << "Two Default Values" << endl;
       cout << "i: " << i << " j: " << j << endl;
     }
};

int main() {
   Test t(20);
}
```

What gets printed?

Constructor Being Called

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

class Test
{
   public:

     Test(int i = 100, int j = 200)
     {
       cout << "Two Default Values" << endl;
       cout << "i: " << i << " j: " << j << endl;
     }
};

int main() {
   Test t(20);
}
```

Two Default Values
i: 20 j: 200

Constructor Being Called

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

class Test
{
   public:

     Test(int i, int j = 200)
     {
       cout << "One Default Value" << endl;
       cout << "i: " << i << " j: " << j << endl;
     }
};

int main() {
   Test t(20);
}
```

What gets printed?

Constructor Being Called

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

class Test
{
   public:

     Test(int i, int j = 200)
     {
       cout << "One Default Value" << endl;
       cout << "i: " << i << " j: " << j << endl;
     }
};

int main() {
   Test t(20);
}
```

One Default Value
i: 20 j: 200

Constructor Being Called

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

class Test
{
   public:

     Test(int i = 100, int j)
     {
       cout << "One Default Value" << endl;
       cout << "i: " << i << " j: " << j << endl;
     }
};

int main() {
   Test t(20);
}
```

What gets printed?

Constructor Being Called

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

class Test
{
   public:

      Test(int i = 100, int j)
      {
        cout << "One Default Value" << endl;
        cout << "i: " << i << " j: " << j << endl;
      }
};

int main() {
   Test t(20);
}
```

Compiler Error!
constructor.cpp:22: error: no matching function for call to 'Test::Test(int)'

Inheritance

No longer are Classes singularly defined lumped objects. But they can be related to each other through ***inheritance*** - which defines *is-a-kind-of* relationship.

(NOT to be confused by the *has-a* relationship)

Dog

Greyhound

Boxer

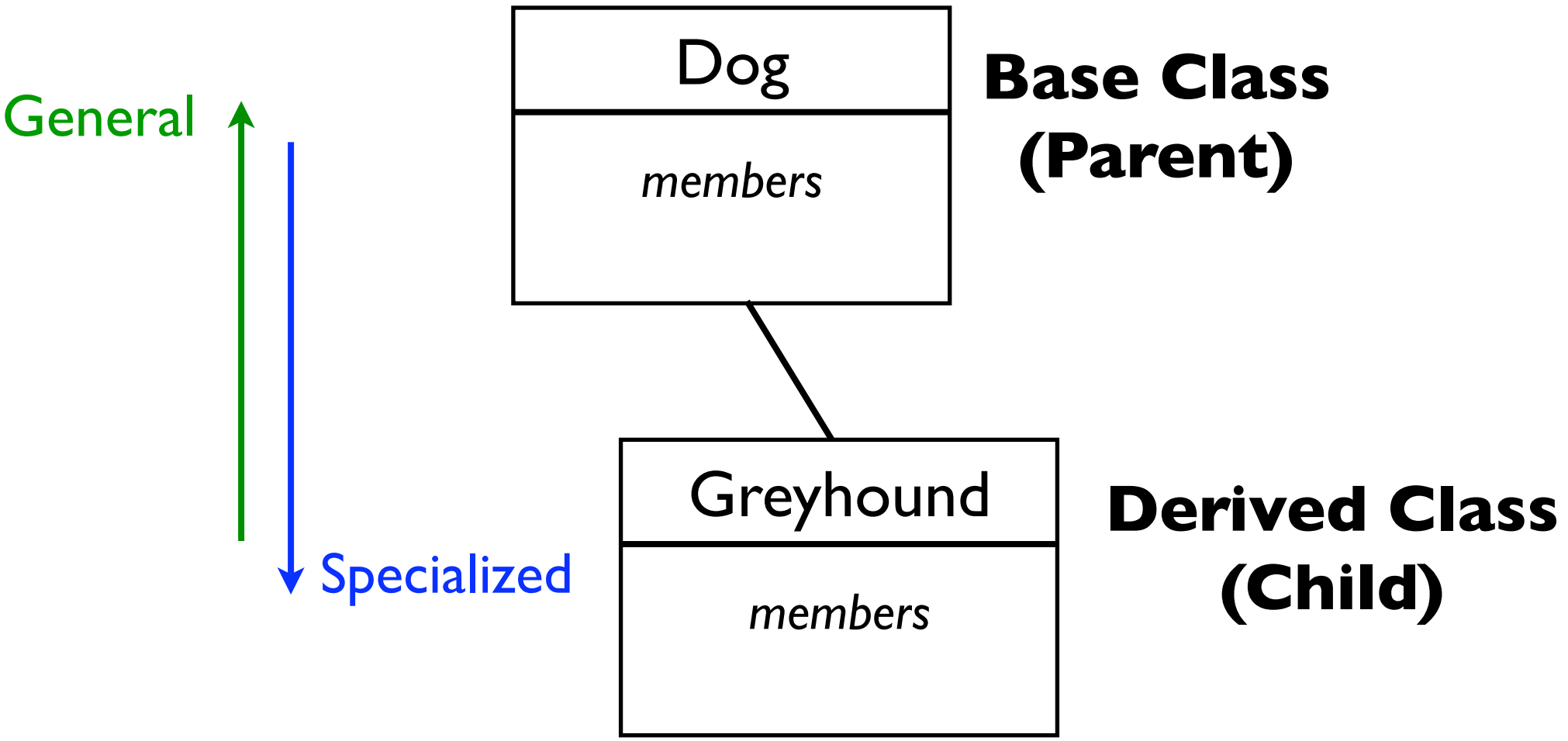
Poodle

AiboRobot

Inheritance

Inheritance allows a new class to be *based* on an existing class.

The new class inherits all the member variables and functions (except the constructors and destructors) of the class it is based on.



Hierarchy of Shapes

Let's start with a **shape** class (a general base class)

```
class Shape {
   private:
      double area;
   public:
      void setArea(double a)
         { area = a; }
      double getArea()
         { return area; }
};
```

No different than any other class that we've seen so far.

Hierarchy of Shapes

Now we will define a child class to inherit properties from the parent.

```
class Shape {
   private:
      double area;
   public:
      void setArea(double a) { area = a; }
      double getArea() { return area; }
};

class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }
};
```

Components of Class Inheritance

Now we will define a child class to inherit properties from the parent.

```
class Shape {
   private:
      double area;
   public:
      void setArea(double a) { area = a; }
      double getArea() { return area; }
};

class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }
};
```

Single ':' indicates inheritance

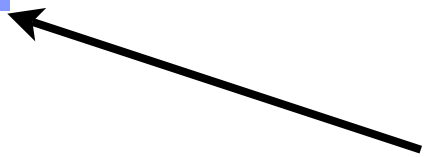
Components of Class Inheritance

Now we will define a child class to inherit properties from the parent.

```
class Shape {
   private:
      double area;
   public:
      void setArea(double a) { area = a; }
      double getArea() { return area; }
};

class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }
};
```



Class to inherit from.

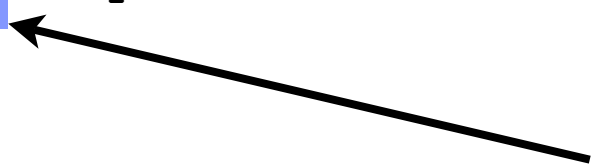
Class access specification

Now we will define a child class to inherit properties from the parent.

```
class Shape {
   private:
      double area;
   public:
      void setArea(double a) { area = a; }
      double getArea() { return area; }
};

class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }
};
```



Determines how *inherited* members are accessed.

Class access specification

Determines how *inherited* members are accessed.

```
class Shape {
   private:
      double area;
   public:
      void setArea(double a) { area = a; }
      double getArea() { return area; }
};

class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }
};
```

Private members of parent class are ***inaccessible*** by the child class.

Child class uses a public member accessor function.

Class access specification

Public base class inheritance.

```
class Shape {
   private:
      double area;
   public:
      void setArea(double a) { area = a; }
      double getArea() { return area; }
};

class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }
};
```

Public members of the parent class are treated as public members of the child class.

Use the Circle Class

```
int main()
{
  Circle c;
  c.setRadius(10);

  cout << c.getArea() << endl;
}
```

Output?

```
class Shape {
    private:
        double area;
    public:
        void setArea(double a)
            { area = a; }
        double getArea()
            { return area; }
};

class Circle : public Shape {
    private:
        double radius;
    public:
        void setRadius(double r)
            { radius = r;
              setArea(3.14 * r * r); }

        double getRadius()
            { return radius; }
};
```

Use the Circle Class

```
int main()
{
  Circle c;
  c.setRadius(10);

  cout << c.getArea() << endl;
}
```

314

```
class Shape {
    private:
        double area;
    public:
        void setArea(double a)
            { area = a; }
        double getArea()
            { return area; }
};

class Circle : public Shape {
    private:
        double radius;
    public:
        void setRadius(double r)
            { radius = r;
              setArea(3.14 * r * r); }

        double getRadius()
            { return radius; }
};
```

Use the Circle Class

```
int main()
{
  Circle c;
  c.setRadius(10);
  c.setArea(157);

  cout << c.getArea() << endl;
  cout << c.getRadius() << endl;
}
```

Output?

```
class Shape {
    private:
        double area;
    public:
        void setArea(double a)
            { area = a; }
        double getArea()
            { return area; }
};

class Circle : public Shape {
    private:
        double radius;
    public:
        void setRadius(double r)
            { radius = r;
              setArea(3.14 * r * r); }

        double getRadius()
            { return radius; }
};
```

Use the Circle Class

```
int main()
{
  Circle c;
  c.setRadius(10);
  c.setArea(157);

  cout << c.getArea() << endl;
  cout << c.getRadius() << endl;
}
```

157
10

Is that correct?

```
class Shape {
    private:
        double area;
    public:
        void setArea(double a)
            { area = a; }
        double getArea()
            { return area; }
};

class Circle : public Shape {
    private:
        double radius;
    public:
        void setRadius(double r)
            { radius = r;
              setArea(3.14 * r * r); }

        double getRadius()
            { return radius; }
};
```

No.

10 is not the radius of a circle of area 157.

Try to fix the Shape class to make it update.

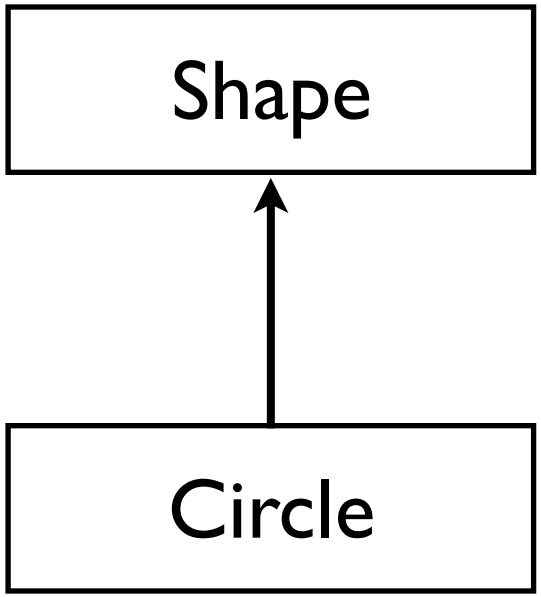
```
class Shape {
   private:
      double area;
   public:
      void setArea(double a)
         { area = a;
           radius = sqrt(a / 3.14); }
      double getArea()
         { return area; }
};
```

error: 'radius' was not declared in this scope

```
class Shape {
   private:
      double area;
   public:
      void setArea(double a)
         { area = a;
           radius = sqrt(a / 3.14); }
      double getArea()
         { return area; }
};
```

Shape class knows nothing about Circle Class

```
class Shape {
   private:
      double area;
   public:
      void setArea(double a)
         { area = a;
           setRadius(sqrt(a / 3.14)); }
      double getArea()
         { return area; }
};
```



Protected Members and Class Access

Protected members are like private members, but they may be accessed by derived classes.

```
class Shape {
   protected:
      double area;
   public:
      void setArea(double a)
         { area = a; }
      double getArea()
         { return area; }
};
```

private to everybody, except to `Shape` class and derived classes.

Protected Members

```
class Shape {
   protected:
      double area;
   public:
      void setArea(double a) { area = a; }
      double getArea() { return area; }
};

class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           area = 3.14 * r * r; }

      double getRadius()
         { return radius; }
};
```

Circle class can access the `area` member.

Base Class Access Specifiers

Can be flexible about how derived classes can access it's inherited parent class members.

```
class Circle : public Shape
```

Base Class Access Specifiers

Can be flexible about how derived classes can access it's inherited parent class members.

```
class Circle : public Shape
```

Private members of the Shape (Base) class are ***inaccessible*** to the Circle (Derived Class)

Protected members of the Shape (Base) class become Protected members of the Circle (Derived Class)

Public members of the Shape (Base) class become Public members of the Circle (Derived Class)

Base Class Access Specifiers

Declaring a protected Base class accessor is *more restrictive* than a public Base class accessor.

```
class Circle : protected Shape
```

Private members of the Shape (Base) class are ***inaccessible*** to the Circle (Derived Class)

Protected members of the Shape (Base) class become Protected members of the Circle (Derived Class)

Public members of the Shape (Base) class become ***Protected*** members of the Circle (Derived Class)

Base Class Access Specifiers

Declaring a private Base class accessor is ***even*** *more restrictive* than a protected Base class accessor.

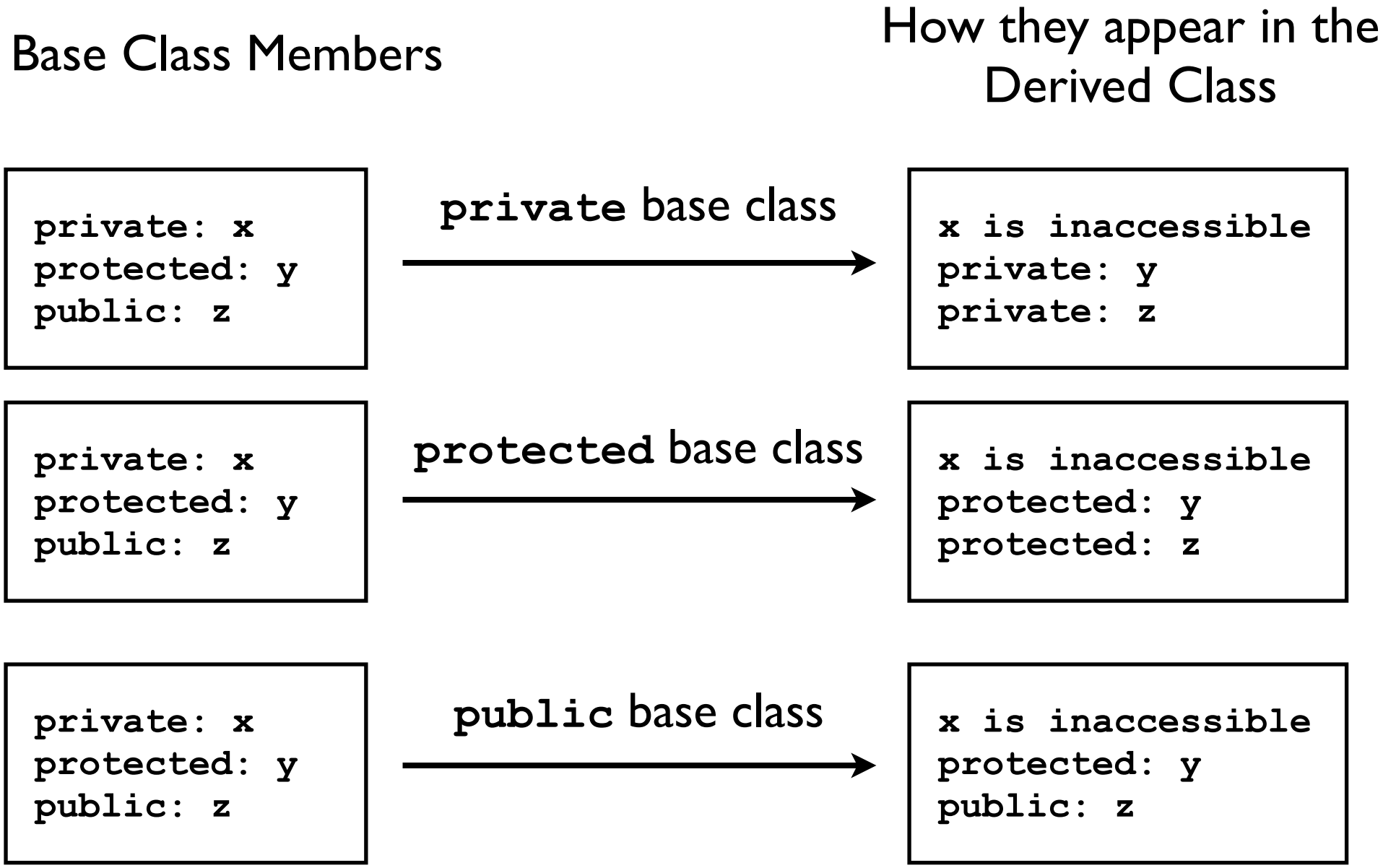
```
class Circle : private Shape
```

Private members of the Shape (Base) class are ***inaccessible*** to the Circle (Derived Class)

Protected members of the Shape (Base) class become ***Private*** members of the Circle (Derived Class)

Public members of the Shape (Base) class become ***Private*** members of the Circle (Derived Class)

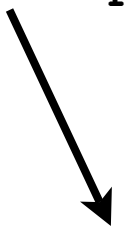
Base Class Access Specifiers



Base Class Access Specifiers

If no Access Specifier is given, the it is ***private*** by default.

```
class Circle : Shape
```



by default

```
class Circle : private Shape
```

Extend your Cat class through a UML Diagram.

```
class Cat
{
```

```
};
```

3 Member Variables
Constructor
Destructor
Accessor/Mutator Functions.