INHERITANCE
Introduction to Inheritance

• Inheritance is a relationship between two or more classes where derived class inherits behaviour and attributes of pre-existing (base) classes

• Intended to help **reuse** of existing code with little or no modification
Inheritance

• Inheritance can be continuous
  – Derived class can inherit another class, which inherits another class and so on
  – When changing the base class all the derived classes changes also

• Example:
  – Mammal ← Human ← Worker ← Programmer

• Could mammal be a derived class? If so, what would be the base class?
Inheritance

Class A features: a, b

Class B features: a, b, c

Class C features: a, b, d, e

Class D features: a, b, d, e, f
Multiple Inheritance

- In multiple inheritance a derived class has multiple base classes
- C++ supports multiple base classes, Java don't
Inheritance and Capsulation

• private
  – Is **accessible** only via the base class

• public
  – Is accessible everywhere (base class, derived class, other classes)

• protected
  – Is accessible by the base class and derived classes
Overriding

• Since programmer eats and drinks differently than humans (only Coke and Pizza) the eat and drink methods are overridden in Programmer!
Abstract Class

• Abstract class is a class which you cannot instantiate (create objects)
• You can inherit abstract class and create objects from the inherited class, if it is concrete one
• Abstract class in C++ has abstract methods, that do not have implementations
• These methods forces derived classes to implement those methods
Example

`<<abstract>>
Mammal

string name

void makesound() {abstract}

Elephant

int trunkLength

makesound()
Example

Figure
int x, y
double calculateArea() {abstract}

circle

double radius
double calculateArea()
Exercises
INHERITANCE IN C++
Declaring Inheritance

class Circle : public Figure
{
}

Declaring Inheritance

class Figure
{
    public:
        int x, y;
};

class Circle : public Figure
{
    public:
        int radius;
};

int main()
{
    Circle a;
    a.x = 0;
    a.y = 0;
    a.radius = 10;
}
Encapsulation

class Figure
{
    protected:
        int x, y;
};

class Circle : public Figure
{
    public:
        int radius;
};

int main()
{
    Circle a;
    a.x = 0;
    a.y = 0;
    a.radius = 10;
}

example.cpp: In function ‘int main()’:
example.cpp:5: error: ‘int Figure::x’ is protected
example.cpp:17: error: within this context
example.cpp:5: error: ‘int Figure::y’ is protected
example.cpp:18: error: within this context
Encapsulation

class Figure
{
    protected:
        int x_, y_; 
};

class Circle : public Figure
{
    private:
        int radius_; 
    public:
        Circle(int x, int y, int radius);
};

circle::Circle(int x, int y, int radius)
{
    x_ = x;
    y_ = y;
    radius_ = radius;
}

int main()
{
    Circle a(0,0,10);
}
Encapsulation

class Figure
{
    private:
        int x_, y_; 
};

class Circle : public Figure
{
    private:
        int radius_; 
    public:
        Circle(int x, int y, int radius);
    int main()
    {
        Circle a(0,0,10);
    }
};
Encapsulation

class Figure
{
    private:
        int x_, y_;
    public:
        void SetX(int x);
        void SetY(int y);
};
void Figure::SetX(int x)
{
    x_ = x;
}
void Figure::SetY(int y)
{
    y_ = y;
}
class Circle : public Figure
{
    private:
        int radius_;  
    public:
        Circle(int x, int y, int radius);
    };
Circle::Circle(int x, int y, int radius)
{
    SetX(x);
    SetY(y);
    this->radius_ = radius;
}
int main()
{
    Circle a(0, 0, 10);
}
What is the result?

class Figure
{
    public:
    Figure()
    {
        cout << "Figure Constructor\n";
    }
    ~Figure()
    {
        cout << "Figure Destructor\n";
    }
};

class Circle : public Figure
{
    public:
    Circle()
    {
        cout << "Circle Constructor\n";
    }
    ~Circle()
    {
        cout << "Circle Destructor\n";
    }
};

int main()
{
    Circle a;
}
Inheritance and Constructors

• When creating a object from derived class, also the member values of the base class must be initialized
• Base constructor is called before the derived classes constructor
• Destructors vice versa.
Calling the Base Classes

class Figure
{
    public:
    Figure()
    {
        cout << "Figure Constructor
";
    }

    ~Figure()
    {
        cout << "Figure Destructor
";
    }
};

class Circle : public Figure
{
    public:
    Circle() : Figure()
    {
        cout << "Circle Constructor
";
    }

    ~Circle()
    {
        cout << "Circle Destructor
";
    }
};

int main()
{
    Circle a;
}
class Figure {
    private:
        int x_, y_; 
    public:
        Figure(int x, int y) : x_(x), y_(y) {
            cout << "Figure Constructor\n";
        }
        ~Figure() {
            cout << "Figure Destructor\n";
        }
};
class Circle : public Figure
{
    private:
        double radius_;  
    public:
        Circle(int x, int y, int radius) : Figure(x, y),
        radius_(radius)
        {
            cout << "Circle Constructor\n";
        }
    ~Circle() {
            cout << "Circle Destructor\n";
        }
};

int main()
{
    Circle a(0,0,5);
}
Abstract Class

• In C++, Abstract class is a class that has one abstract method
• Abstract method is a method without implementation.
• Abstract method is created by reserved word "virtual"
Example of Abstract class

class Figure
{
    private:
        int x_, y_;  
    public:
        Figure(int x, int y): x_(x), y_(y) {
            cout << "Figure Constructor\n";
        }
        ~Figure() {  
            cout << "Figure Destructor\n";
        }

        virtual double calculateArea() = 0;
    };

Example of Abstract class

class Circle : public Figure
{
    private:
        double radius_
    
    public:
        Circle(int x, int y, int radius) : Figure(x, y),
            radius_(radius)
        {
            cout << "Circle Constructor\n"
        }
        ~Circle() {
            cout << "Circle Destructor\n"
        }
        double calculateArea() {
            return 3.14 * radius_ * radius_;
        }
};
Example of Abstract class

```cpp
int main()
{
    Circle a(0,0,5);
    cout << a.calculateArea() << endl;

    // This Does not work, since figure is abstract:
    // Figure f(0,0);
}
```