TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

Object Inheritance

Taking another step… Extending objects to build other objects

PRESENTED BY:

Texas Advanced Computing Center

EXERCISE 1

Make a class Rectangle (sides parallel to axes) :

- Rectangle(Point bl,float w,float h);

Implement methods

- float area(); float rightedge(); float topedge();

Looks like we'll need either a Point class or structure.

Polymorphism

polymorphism is the provision of a single interface to entities of different types or the use of a single symbol to represent multiple different types.

Polymorphism - this time in English

polymorphism allows us to create objects or call functions in multiple ways by passing different types of data

Polymorphism - this time in English

```
class Segment {
    private:
// a bunch of private variables defined here!
    Segment()
    {
        //default constructor
    }

    Segment(Point start,float length,float angle)
    {
        // constructor code for building a segment from
        // a starting point, a length, and an angle
    }

    Segment(Point start,Point end)
    {
        // constructor code for building a segment from
        // a starting point and an ending point
    }
};
```

EXERCISE 2

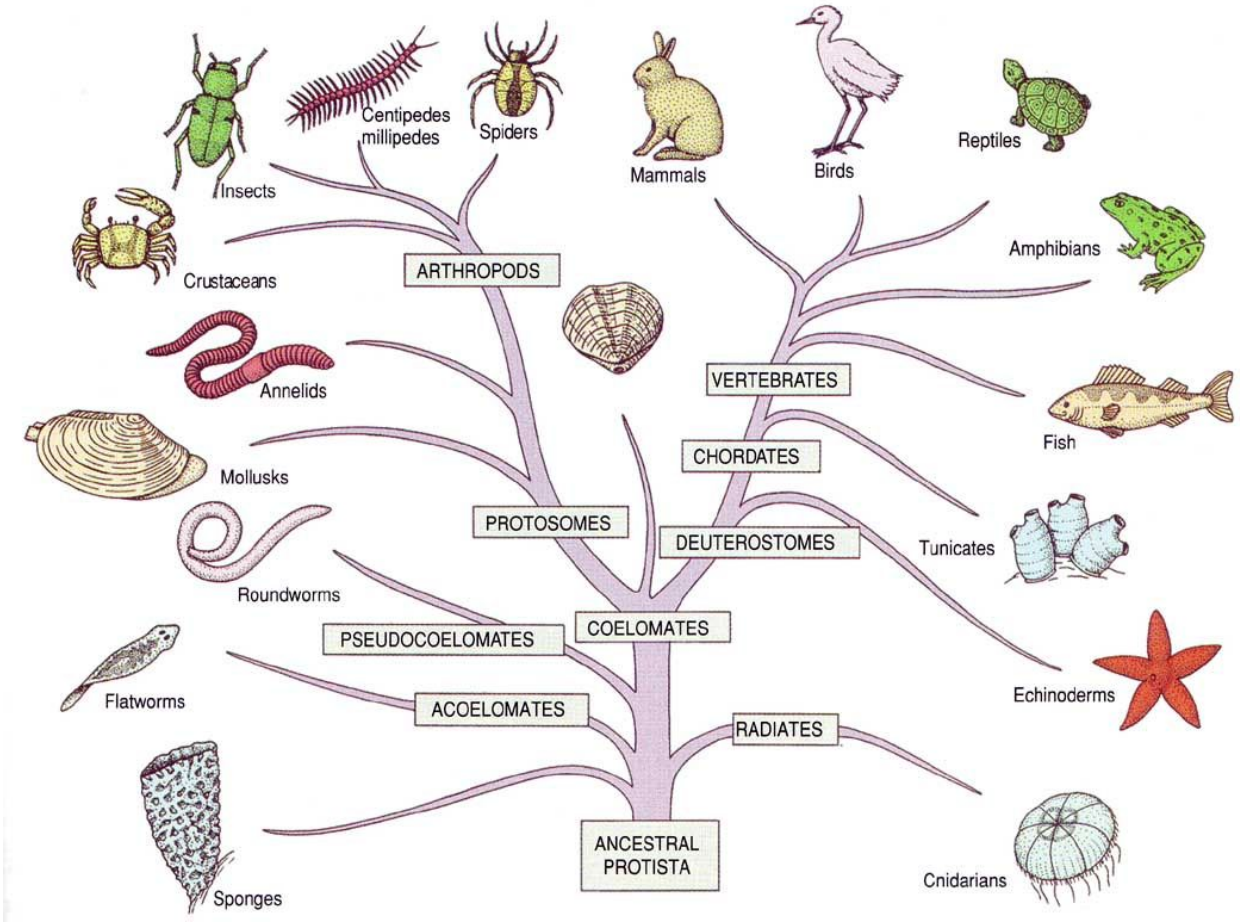
Add a second constructor to your Rectangle Class :

- Rectangle(Point bl, Point tr);

Inheritance

Inheritance in Object Oriented Programming can be described as a process of ***creating*** new classes from ***existing*** classes. New classes ***inherit*** some of the properties and behavior of the existing classes and may ***override*** others.

Inheritance



Inheritance

Class methods and members/data can be:

- **private, they are only used internally;**
- **public, they should be usable from outside a class object, for instance in the main program;**
- **protected, they should be usable in derived classes.**

Inheritance

General case, special case
You can have classes where an object of one class is a special case of the other class. You declare that as

```
class General {
   protected: // note!
      int g;
   public:
      void general_method() {};
};

class Special : public General {
   public:
      void special_method() { g = ... };
};

int main() {
   Special special_object;

   special_object.general_method();
}
```

Inheritance

Inheritance: derived classes

Derived class ***Special*** inherits methods and data from base class ***General***:

```
int main() {
   Special special_object;
   special_object.general_method();
```

Parameters and Methods need to be **protected** to be inheritable.

Inheritance

Constructors

When you run the special case constructor, the general case needs to run, too.
By default the ‘default constructor’ would run, but you can specify a different constructor:

```
class General {
   public:
      General(){ };
   General( double x,double y ) { … };
};

class Special : public General {
   public:
      Special( double x ) : General(x,x+1) {};
};
```

Class initialization and use

Let's look at my rectangle class:

```
class Rectangle {
private: // This is only viewable within the object!
  double l,h;
public:  // This is available to public - anything outside of this class!
  Rectangle(){ //default constructor
  }
  Rectangle( double l,double h ) {  //secondary constructor
    l = length; h = height;
  };
  double getlength() { return l; }; // accessor
  double getheight() { return h; };
  void setlength(double length) { l = length; }
  void setheight(double height) { h = height; }

  double area() { return l*h;}
  double circumference { return 2*(l+h); }
};

int main() {
   Rectangle myRectangle = Rectangle(2, 5);
   Rectangle mySquare = Rectangle(2, 2);
}
```

Class initialization and use

Let's create a class square that inherits from rectangle:
first, change our inheritable variables from `private` to `protected`

```
class Rectangle {
protected: // This is only viewable within the object!
  double l,h;
public:  // This is available to public - anything outside of this class!
  Rectangle(){ //default constructor
  }
  Rectangle( double length, double height ) {  //secondary constructor
    l = length; h = height;
  };
  double getlength() { return l; }; // accessor
  double getheight() { return h; };
  void setlength(double length) { l = length; }
  void setheight(double height) { h = height; }

  double area() { return l*h;}
  double perimeter { return 2*(l+h); }
};
```

EXERCISE 3

- **Create a Square object class that *inherits* from your Rectangle class**

Class initialization and use

Let's create a class square that inherits from rectangle:

```
class Square : Rectangle {
public:  // This is available to public - anything outside of this class!
  Square()
  {
      //default constructor
  }
  Square( double length ) : Rectangle(length, length)
  {
     //secondary constructor
  }
};
```

Inheritance, we can go crazy!