**Chapter 13**

* Method is a function that is a member of a class.
  + C++ is one of few that can have both
* When you instantiate a class, the code is not implemented
  + **However** if you do inline functions then the code **is** implemented
* Remember the semicolon at the end of the class
* Public - can be accessed outside of class
* Private - is the **default** if not specified
* A **const** in the member function will not change any data in the calling object
* :: scope resolution operator
  + Int rectangle :: setwidth(double w)
  + Remember the scope resolution operator must come **after** the return type and immediately **before** function name
* Accessor - retrieve data from private variable
* Mutator - function that changes or stores value in private value
  + In order to use these accessor or mutators you must use the dot operator
* Stale data
  + When storing area for example you don’t want to use member variable because if length or width change the area won't get immediately updated
* Rectangle r;
  + This **does** contain a rectangle object
    - It instantiates it. Basically creating an instance of the class.
* Pointers to object
  + Rectangle \*rPtr
    - Does not create a rectangle
    - Can store an rectangle object though
  + Can access public members via pointer
    - RPtr = &otherRectangle; // this makes RPtr point to other rectangle
    - RPtr ->setWidth (12.5) // use -> to call member functions
    - RPtr = new Rectangle ();
      * Allocates a rectangle and returns a pointer to it
* Dynamically allocated objects with a pointer
  + Rectangle \*rectPtr;
  + RectPtr = new Rectangle;
  + RectPtr -> setWidth(10.0)
  + Delete rectPtr;
  + RectPtr =0;
* Private members - data can't be accessed outside of functions
* Inline functions can create faster code but not necessarily
* Rectangle r;
  + Calls a constructor automatically when object is created
  + Constructors don’t have a return type because they don’t have an
  + c++ creates a default constructor
* When you use a dynamically allocated object, since the code Rectangle \*rectPtr does not instantiate
  + The statement rectPtr = new Rectangle; calls the default constructor
* **When you are returning a pointer to a constant char for example the code is** 
  + **Const char\*getName () const** 
    - **Each function returns a pointer to a constant char**
* Can create constructors with arguments
  + Rectangle (double,double)
  + Rectangle (double = 0,double= 0)
  + You can only have 1 default constructor
    - **Rectangle ();**
    - **Rectangle (int =0);**
      * **Will not work!**
* You can only have 1 destructor per class
* Only 1 default constructor
* Member function overloading
  + Void setCost(double);
  + Void setCost(char \*);
    - These do very similar things
* Arrays of objects
  + Rectangle rooms [8]
    - Creates 8 rectangle
  + Rectangle \*[8]
    - Creates space for 8 rectangles but not the rectangles themselves
  + InventoryItem inventory []= {"hammer",InventoryItem ("wrench",8.9,6.7),"wrech"}
    - It is valid if there are valid constructors for the calls.
* Accessing array members
  + Inventory[2].setunites(30);
* **NO UML QUESTIONS ON TEST**

**Chapter 14**

* There is one instance of a static variable
  + DECLARE IN, DEFINE OUT
    - **Static int objectCount;**
    - **Int tree::objectCount = 0;**
      * Static member variables if left uninitialized are set to 0
  + You don't even has to have any objects initialized in order to store into static member variables
* Friend variables give you a way to access private and protected members of another class even though they are not derived
  + A function or class that is not a member of a class but has access to private members of the class
    - Friend keyword can allow you to make a class or a member function a friend
    - friend void setAVal(intVal&, int);
    - friend void SomeClass::setNum(int num)
      * Remember when using a friend , you must forward declare
        + Class that holds the someclass::setNum-s

Class s; <- this would have to be in the someclass.h

* Member wise assignment
  + **Box 2= box1 ;** 
    - Sets all member values from box 2 to box 1
* Make a copy constructor when you have a pointer in an object
* Copy constructors should be on test
  + Programmed defined copy constructor
    - SomeClass::SomeClass (SomeClass &obj)
      * THIS CAN modify the object
        + This is why you use const
* **Copy constructors** 
  + The argument **has to be a reference variable**
  + **StudentTestScores (const StudentTestScores &obj)**
    - StudentName =obj.studentName;
    - NumTestScores = obj.numTestScores;
    - TestScores = new double [numTestScores];
    - For(int i =0;i<numTestScores;i++)
      * TestScores [I]= obj.testScores[i];
  + **Use when there is a pointer as a member variable**
  + **So now when you do object2 = object 1** 
    - **They point to different memory locations**
* Operator Overloading
  + Operator +
    - To overload plus operator
  + To invoke a overloaded operator
    - Object1.operator = (object2); is the same as
    - Object1 = object2;
      * This is only true if the prototype of the overloaded operator looks like this
        + **Void operator = (const StudentTestScores &right)**

StudentName =obj.studentName;

NumTestScores = obj.numTestScores;

TestScores = new double [numTestScores];

For(int i =0;i<numTestScores;i++)

TestScores [I]= obj.testScores[i];

Since this is a member of the StudentTestScores class, it will only run when there is a object from that class on the left side

* + Can return a value
    - Look at slides
    - Object 1 = object 2= object 3
    - Return \*this;
      * This returns the object in this manner
    - THIS -> AREA= LENGTH \*WIDTH
      * + **Void operator = (const StudentTestScores &right)**

Delete [] testScores

StudentName =obj.studentName;

NumTestScores = obj.numTestScores;

TestScores = new double [numTestScores];

For(int i =0;i<numTestScores;i++)

TestScores [I]= obj.testScores[i];

Return \*this

* + - * This returns
  + Char c = s1[1];
  + C=s1.at(1);
    - These are equivalent
  + S1[1]='c';
    - This does not work
* The this pointer
  + Look at slides
* Aggregations
  + A class is a member of a class

**Chapter 15**

* The new notation for inheritance is
  + Class FinalExam : public GradedActivity
    - Indicated the name of the class being declared and name of the base class it is derived from
    - final exam is the derived class and gradedActivity is the base class
  + public is a base class access specification
    - public means that the public members of GradedActivity class will become public members of the FinalExam Class
    - the private members of the GradedActivity base class can’t be directly accessed with final exam
  + inheritance allows us to have the derived class use function from the base class but not the other way around
* protected class access
  + are like private members except they may be accessed by functions in a derived class
    - they are still inaccessible to the rest of the program
* Do not confuse base class access specification with member access
  + Member access determines how members that are defined are accessed
  + Base class determines how inherited members are accessed
* When you have a private base class
  + The private members are inaccessible to the derived class
  + The protected members are private to the derived class
  + The public members are private
* When you have a protected class
  + The private members are in accessible
  + The protected members are protected
  + The public members are protected
* When you have a public base class
  + Private members are private
  + Protected members are protected
  + Public are public
* Passing arguments to base class constructors from derived classes
  + ClassName::ClassName (parameter list ) : BaseClassName (ArgumentList)
  + However in real code this would look something like this
    - Car (int door,double mileage,double cost,int wheels):Automobile (door,mileage,cost,wheels)
      * {doors =carDoors}
  + A default constructor would look like
    - Car () : automobile ()
* Redefining Base class functions
  + REDEFINING is different from overloading because redefining always used the derived class function even if the parameter lists are the same
  + Here is how you redefine a function
    - Void setScore(double s)
      * {
      * rawscore =s ;
      * GradedActivity ::setScore (rawScore \*percentage)
      * }
        + BaseClassname :: functionName (argumentList);
* Because function calls are bound at compile time, if a base class defines x and y in which x calls y, and the derived class redefines y, and an object of derived class is created and x is called. The function from the base class y is called.

Class Hierarchies

Polymorphism

* Because of the Is a relationship, you can pass a derived class in place of a derived class in order to make the function call
  + Virtual functions are declared by placing the key word virtual before the return type
    - **Virtual chat getLetterGrade () const;**
      * The virtual basically ells the compiler to expect the getLetterGrade to be redefined
        + Binds at runtime rather than compile time
* **Polymorphism requires a reference or POINTER**
  + **Static binding will take place if there is no reference**
  + **Basically if you use virtual YOU HAVE TO USE REFERENCE VARIABLES**

Base class Pointers

* **GradedActivity \*exam = new PassFailExam (100,25,75.0);**
  + Dynamically allocated a passfailexam object and assigns its address to exam which is a pointer. We can call member functions too
    - Cout << exam ->getscore()<<endl;
    - Cout << exam ->getLetterGrade()<<endl;
  + The exam variable only knows about member functions in the graedactivity class. It cannot execute funcitons from the passfailexam object

**VIRTUAL FUCNTIONS ARE OVERIDDEN AND NON VIRTUAL FUNCTIONS ARE REDEFINED**

**ANY CLASS THAT HAS A VIRTUAL MEMBER FUNCTION SOHULD HAVE A VIRTUAL DESRUCTOR**

* Virtual ~animal ()

**Pure virtual functions**

* Are used for the basis for other things
* Virtuial int getRemainingHours()=0
  + **the =0 makes it virtual**
  + you must override it
* when you have an pure virtual function, you have an abstract base class
  + **you cant instantiate abstract base classes**
* **class cube : public square, public rectSolid;**
  + **multiple inheritance**
* **constructor for this is** 
  + **DateTime(int dy, int mon,int hr,int mt): Date (dy,mon),Time(hr,mt)**
    - **{}**

Virtual functions

Dynamic v. Static binding

"remember the example" , "this is important"

It decides what to run at runtime because it can't know what to run