**Notes on Operator Overloading**

* Operators such as +,-, %, == and so forth are functions that are used with a slightly different syntax than regular functions. We write x +8 rather than +(x,8), but the + operator is a function that takes two arguments and returns a single value. This is an example of what is called ‘Syntactic Sugar’.
* You can overload operators (e.g. +,-.\*) so that they take arguments of a class type.
* You cannot create a new operator, change the number of arguments that an operator takes or change the precedence of an operator.
* Although operators may be call-by-value or call-by-reference parameters and may have the const modifier or not, for efficiency reasons, constant call by reference is usually used in place of call-by-value for classes.
* The basic syntax for operator overloading for classes is:

const ClassName operator +(const ClassName& X, const ClassName& Y);

An example of the use of an overloaded + operator would be:

Sales OntSales = new Sales(); Sales QueSales = new Sales(); Sales TotalSales = new Sales();  
TotalSales = OntSales + QueSales;

* Operators may be overloaded using standalone functions defined outside a class or as member functions inside a class. Inside is preferred as this is considered better OO practice.
* When a binary (single operator such as +) is overloaded **as a member operator**, then there is **ONLY ONE parameter NOT TWO.** In other words, **the calling object serves as the first parameter.**
* e.g. If Object X has a state of 3 and Object Y has a state of 4 and you go   
  total = ObjX + ObjY, you are really calling the + method of the ObjX class. So really it acts like ObjX.+(ObjY) where ObjX’s + method adds its own 3 and ObjY’s 4 together. Note that the + method of ObjY is not called.
* Const is added to the end of the operator declaration and to the end of the operator definition so that the operator invocation (use) does not change the calling object
* If you wish to have code such as Total = ObjX + 25, where 25 is not an object, then you need to build into ObjX’s + operator with an overloaded constructor that takes a single integer. This is called **Automatic Type Conversion** as this constructor takes the 25 which is not an object of the same type as ObjX and converts it to an object of the same type.
* The advantage of overloading an operator as a nonmember gives you **automatic type conversion of all arguments.** 
  + i.e. so both total = ObjX + 25 **and** total = 25 + ObjX are ok
* Overloading an operator as a member gives you **the efficiency of bypassing** accessor and mutator functions providing direct access to member variables. The problem is that only one side of the + in the equation is an argument.
  + i.e. so total = ObjX + 25 is OK but total=25 + ObjX is not OK.
* Friend functions are nonmember functions that have all the privileges of member functions. They are used with overloading because they allow both of the above noted advantages. Friend function definitions exist outside of the class but have direct access to the properties of both of the above ObjX and ObjY.
* You must list the function declaration for the friend in the class definition.
* Friend function declarations (inside the class) look like this:
  + Friend const ClassName operator +(const ClassName& ObjX, const ClassName& ObjY);