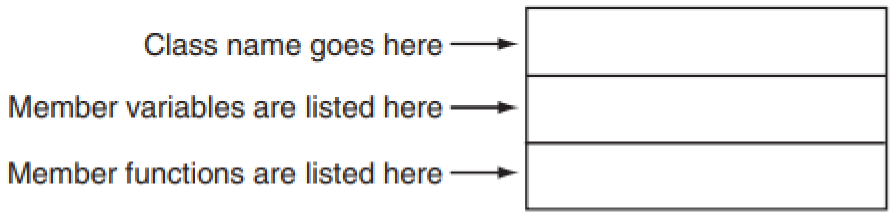
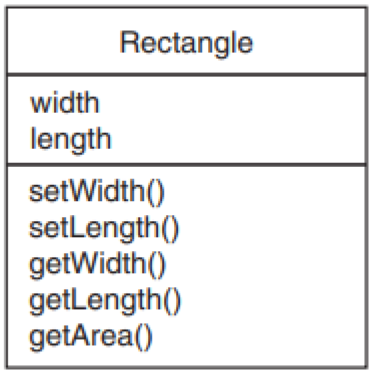
When designing a class it is often helpful to draw a UML diagram. UML stands for Unified Modeling Language. The UML provides a set of standard diagrams for graphically depicting object-oriented systems. Figure 13-18shows the general layout of a UML diagram for a class. Notice that the diagram is a box that is divided into three sections. The top section is where you write the name of the class. The middle section holds a list of the class's member variables. The bottom section holds a list of the class's member functions.

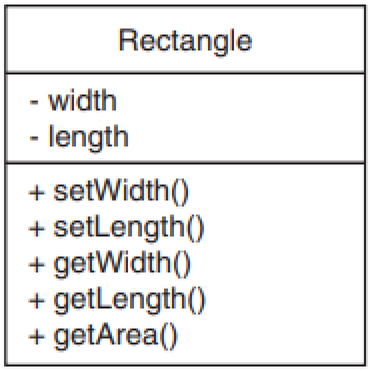

Class name goes here
Member variables are listed here
Member functions are listed here

The first version of the Rectangle class that you studied had the following member variables: • width • length The class also had the following member functions: • setWidth • setLength • getWidth • getLength • getArea From this information alone we can construct a simple UML diagram for the class, as shown



The UML diagram in Figure tells us the name of the class, the names of the member variables, and the names of the member functions. The UML diagram in Figure 13-19does not convey many of the class details, however, such as access specification, member variable data types, parameter data types, and function return types. The UML provides optional notation for these types of details.

Showing Access Specification in UML Diagrams The UML diagram in Figure 13-19lists all of the members of the Rectangle class but does not indicate which members are private and which are public. In a UML diagram you may optionally place a - character before a member name to indicate that it is private, or a + character to indicate that it is public. Figure 13-20shows the UML diagram modified to include this notation.

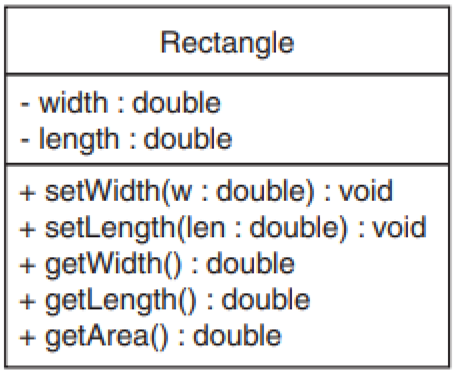


Data Type and Parameter Notation in UML Diagrams The Unified Modeling Language also provides notation that you may use to indicate the data types of member variables, member functions, and parameters. To indicate the data type of a member variable, place a colon followed by the name of the data type after the name of the variable. For example, the width variable in the Rectangle class is a double. It could be listed as follows in the UML diagram: - width : double

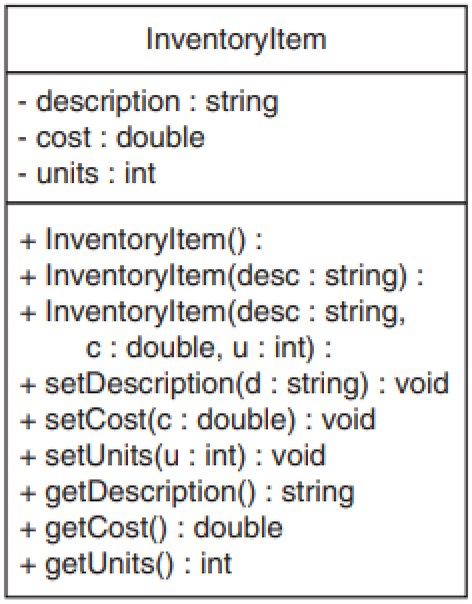
In UML notation the variable name is listed first, then the data type. This is the opposite of C++ syntax, which requires the data type to appear first.

The return type of a member function can be listed in the same manner: After the function's name, place a colon followed by the return type. The Rectangle class's getLength func⍰tion returns a double, so it could be listed as follows in the UML diagram: + getLength() : double Parameter variables and their data types may be listed inside a member function's paren⍰theses. For example, the Rectangle class's setLength function has a double parameter named len, so it could be listed as follows in the UML diagram: + setLength(len : double) : void

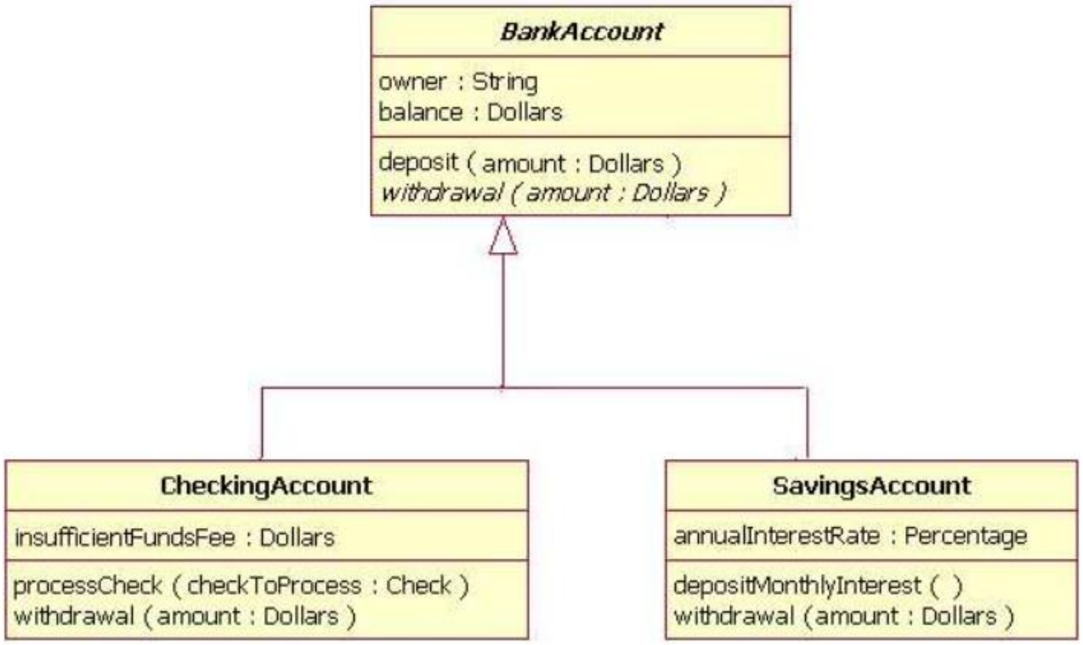
Figure shows a UML diagram for the Rectangle class with parameter and data type notation.

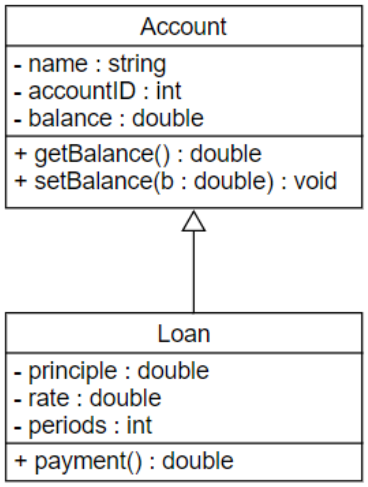


Showing Constructors and Destructors in a UML Diagram There is more than one accepted way of showing a class constructor in a UML diagram. In this book we will show a constructor just as any other function, except we will list no return type. For example, Figure shows a UML diagram for the InventoryItem class that we looked at previously in this chapter.


InventoryItem
- description : string
- cost : double
- units : int
+ InventoryItem() :
+ InventoryItem(desc : string) :
+ InventoryItem(desc : string, c : double, u : int) :
+ setDescription(d : string) : void
+ setCost(c : double) : void
+ setUnits(u : int) : void
+ getDescription() : string
+ getCost() : double
+ getUnits() : int

Inheritance:


BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)



```
double Loan::payment()
{
    double p = principle * rate / (1 - pow(1 + rate, -periods));
    setBalance(getBalance() - p + balance * rate);

    return p;
}

int main()
{
    Loan carLoan;

    cout << carLoan.getBalance() << endl;
    cout << carLoan.payment() << endl;

    return 0;
}
```

Figure 3. Inheriting functions and variables. A superclass's ability to share its features with its subclasses is one of the most important aspects of inheritance. If we were to write an object-oriented program to automate a bank or credit union, we might find it convenient to create a general `Account` class and several specialized kinds of accounts, such as `Loan`. An `Account` has a name, an account ID, and a balance. Through inheritance, a `Loan` *is-a* special kind of `Account`, and it inherits the name, account ID, and balance - it has and can use these members without redefining them.

The `Loan` payment function can call the setter and getter functions in the `Account` class, which in turn access the `Account`'s private member variables. An instance of the `Loan` class (`carLoan` in this example) can call any public functions in the `Account` class (e.g., `getBalance`) in addition to any public functions defined in the `Loan` class (e.g., payment). In this sense, inheritance is a way to reuse code in a program. While we have used functions to reuse code in the past, inheritance gives us a way, at least to some extent, to also reuse variables.