

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. Your text book and this appendix introduce some of the more commonly used ones. Figure F-1 shows the general layout of a UML diagram for a class. Notice that the diagram is a box 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 member variables. The bottom section holds a list of the class member functions.

**Figure F-1**



For example, in Chapter 7, Introduction to Classes and Objects, you studied a `Rectangle` class that could be used in a program that works with rectangles. The class has the following member variables:

- `length`
- `width`

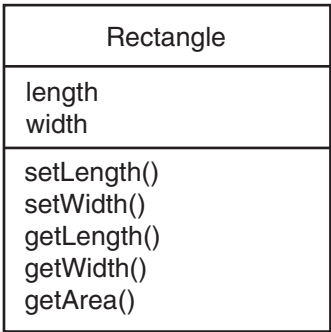
The class also has the following member functions:

- `setLength`
- `setWidth`
- `getLength`
- `getWidth`
- `getArea`

From this information we can construct a simple UML diagram for the class, as shown in Figure F-2.

**Figure F-2**

---



The UML diagram in Figure F-2 tells us the name of the class, the names of the member variables, and the names of the member functions. Compare this diagram to the actual C++ class declaration, which follows.

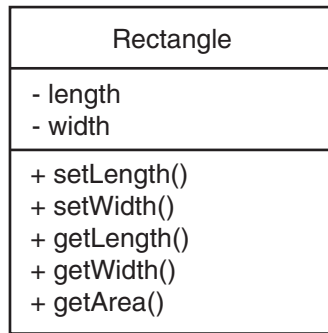
```
class Rectangle
{
    private:
        double length;
        double width;
    public:
        void setLength(double);
        void setWidth(double);
        double getLength();
        double getWidth();
        double getArea();
};
```

The UML diagram shown in Figure F-2 does not include many of the class details, such as access specification, member variable data types, parameter data types, and function return types. However, the UML does provide optional notation if you wish to include these types of details.

**Showing Access Specification in UML Diagrams**

The UML diagram in Figure F-2 lists all of the members of the `Rectangle` class but does not indicate which members are private and which are public. To include this information in a UML diagram you place a `-` character before a member name to indicate that it is private, or a `+` character to indicate that it is public. Figure F-3 shows the UML diagram modified to include this notation.

Figure F-3



## 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 variable name. For example, the `width` variable in the `Rectangle` class is a `double`, so it could be listed in the UML diagram like this:

```
- width : double
```



**NOTE:** In UML notation the variable name is listed first, then the data type. This is the reverse 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 `getLength` function returns a `double`, so it could be listed in the UML diagram like this:

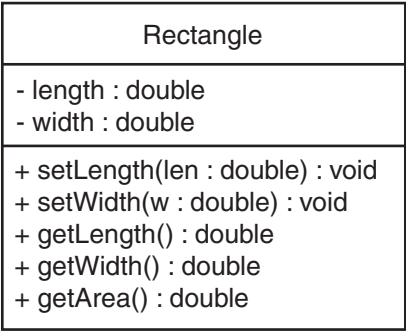
```
+ getLength() : double
```

Parameter variables and their data types may be listed inside a member function's parentheses. For example, the `Rectangle` class `setLength` function has a `double` parameter named `len`, so it could be listed in the UML diagram like this:

```
+ setLength(len : double) : void
```

Figure F-4 shows a UML diagram for the `Rectangle` class that includes data type, return type, and parameter notation.

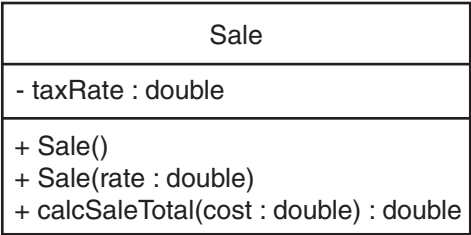
Figure F-4



Showing Constructors and Destructors in a UML Diagram

There is more than one accepted way of showing a constructor in a UML diagram. In this appendix we will show a constructor just as any other method, except we will list no return type. Figure F-5 shows a UML diagram for the `Sale` class, which was also discussed in Chapter 7. This class has two overloaded constructors, as well as a `calcSaleTotal` method.

Figure F-5

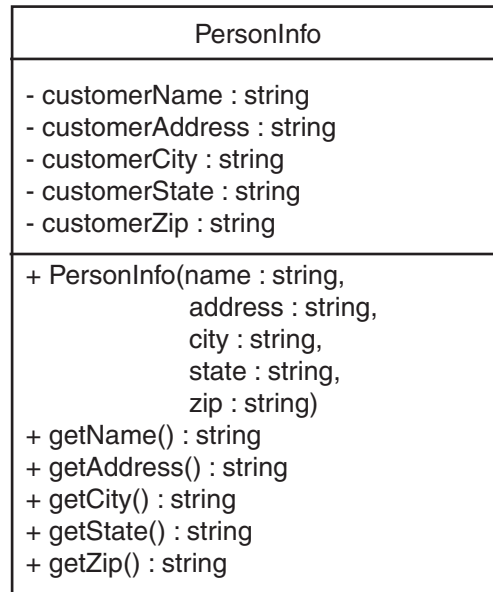


Object Composition in UML Diagrams

Chapter 11 of the text introduces object composition. This occurs when a class contains an instance of another class as a member. Object composition is shown in a UML diagram by connecting two classes with a line that has an open diamond at one end. The diamond is closest to the class that contains instances of other classes.

For example, suppose we have the `PersonInfo` class shown in Figure F-6, which holds information about a person.

Figure F-6



Suppose we also have the `BankAccount` class shown in Figure F-7, which holds the balance of a bank account, and can perform operations such as making deposits and withdrawals.

Figure F-7

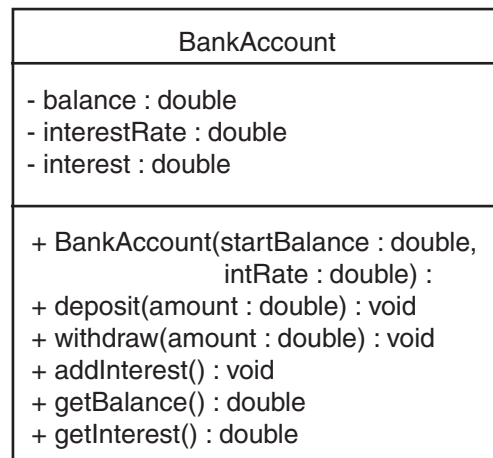
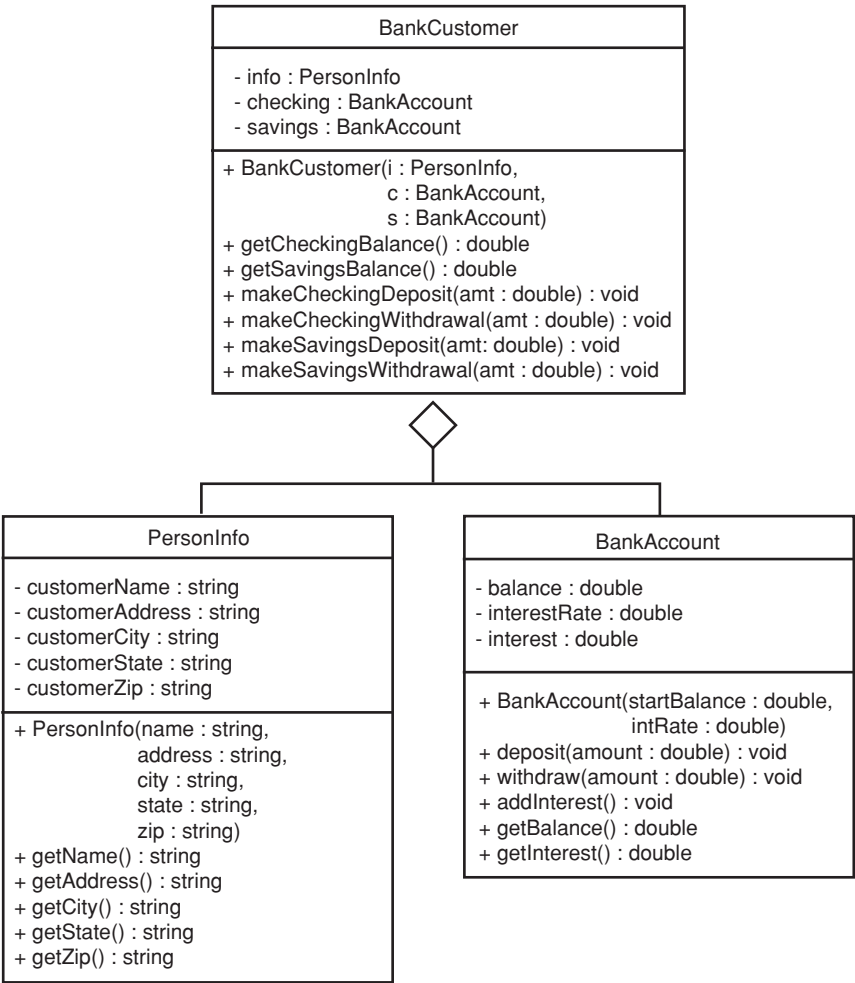


Figure F-8 shows a UML diagram for another class, `BankCustomer`, which contains instances of the `PersonInfo` and `BankAccount` classes as members. The relationship between the classes is shown by the connecting lines with the open diamond. The open diamond is closest to the `BankCustomer` class because it is the one that contains instances of the other classes as members.

Figure F-8

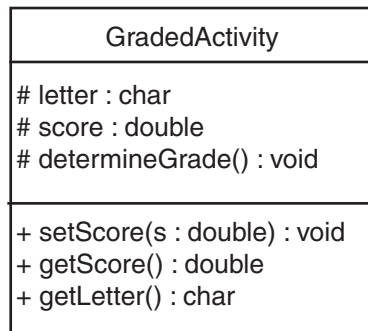


### Showing Protected Members

Chapter 11 of the text also introduces inheritance, which allows new classes to be derived from existing classes.

If a class is going to serve as a base class from which other classes are derived, it is often desirable to designate some of its members as `protected`, rather than `public` or `private`. As explained in Chapter 11, protected members of a class are safeguarded from outside use like private members, but they can be accessed by member functions of derived classes. Protected class members may be denoted in a UML diagram with the `#` symbol. Figure F-9 shows a UML diagram for a class with two protected variables and a protected method.

Figure F-9



## Inheritance in UML Diagrams

You show inheritance in a UML diagram by connecting two classes with a line that has an open arrowhead at one end. The arrowhead points to the base class. For example, Figure F-10 shows a UML diagram depicting the relationship between a class named **GradedActivity** and a class named **FinalExam** that is derived from it. The arrowhead points toward the **GradedActivity** class because it is the base class.

Figure F-10

