Introduction to RAPTOR: OOP Mode

Object-Oriented Mode

Object-oriented mode allows you to create classes with methods and attributes, instantiate objects, and experiment with Object-Oriented Programming (OOP).

To use RAPTOR in OOP, you must select Object-oriented mode, as shown in Figure 1.

![Figure 1 Selecting Object-oriented mode](image)

You will see two tabs: UML and main. RAPTOR uses a type of UML to create the structure of an object-oriented program. The classes are created in the UML screen; therefore, click the UML tab. The button to add a new class is shown in Figure 2. Note that a new Return symbol has been added to the symbols.
Creating a Class

When you click the **Add New Class** button to add a new class, a **Name** box will appear. Enter a name for the **Class**, as shown in Figure 3.
In Figure 3, a **Class** named **Cube** has been created. Double-click inside the class (**Cube**) to add members (methods and attributes). In RAPTOR, note that attributes are called **Fields**. A new window opens to allow you to enter the members (see Figure 4).

![Figure 4 Adding members to a Class](image)

From this point, examples will be used to demonstrate the features of OOP mode and indicate how to use them in a program.
Example: Using the Cube Class to Find the Volume of a Cube

We will use a class named Cube that takes the value of a side of a cube and computes the cube’s volume. So we need the following:

attributes: Side (a number) and Volume (a number)
methods: SetSide(), GetSide(), ComputeVolume(), and GetVolume()

Figure 5 (following page) shows the Class Cube and its members.

• Note the syntax for a Field: A Field must be given a data type. The type of Side and Volume is int and in this case, each field has been given an initial value of 1.

• Note the syntax for a Method. If the Method receives a value passed from main, you must include that parameter. For example,
  o The Method SetSide() is passed a value for the length of a side so the syntax for this Method is
    
        public void SetSide(int NewSide)
  
  o The Method ComputeVolume() uses the value of the side of a cube to do its calculations so it needs one parameter, the integer variable Side. The syntax is
    
        public void ComputeVolume(int Side)
  
  o The Method GetVolume() retrieves the value of the volume of the cube from ComputeVolume() so the syntax for this Method is
    
        public void GetVolume(int Volume)
  
  o The Method GetSide() does not need a parameter so the syntax is
    
        public void GetSide()
Once the **Class** has been created, a new tab is automatically added, with the name of the **Class** (see Figure 6). Now the code for each of the **Class’s** methods must be created. Click the **Cube** tab to see four new tabs—one for each **Method**, as shown in Figure 7.
A new tab is created for the Cube Class

Figure 6 New tab for the Class Cube
Code the Methods

The Methods for this program are as follows: SetSide(\texttt{NewSide}), \texttt{ComputeVolume(Side)}, \texttt{GetVolume(Volume)}, and \texttt{GetSide()}.  

\texttt{SetSide()} Method:

The \texttt{SetSide()} Method does one thing only. It sets the value of the side of a cube, as passed to it from the main program, to the variable \texttt{NewSide}. This assignment is done using the \texttt{this} keyword. The code for this method is shown in Figure 8.
ComputeVolume(Side) Method:

The `ComputeVolume(Side)` method computes the volume of the cube. First, it must receive the value needed for the computation (`Side`). Then, it must do the computation by cubing the value. Finally, it needs to export this result when requested. Figure 9 shows the code.
Figure 9 Code for the ComputeVolume() method

GetVolume(Volume) Method:

The GetVolume(Volume) Method retrieves the value of Volume when it is accessed and then returns it, as shown in Figure 10.
GetSide() Method:

The \texttt{GetSide()} Method retrieves the value of \texttt{Side} when accessed, as shown in Figure 11.
The Main Program

Now the Main program can be created. The program for this example is extremely simple; it will allow the user to enter a value for the side of a cube, compute the volume of that cube, and display the result. This is accomplished by instantiating an object of type Cube, which we will call CubeOne, and using the methods and attributes of Cube. Figure 12 shows how this is done the RAPTOR OOP way.

Figure 12 Code to input a side of a cube and output its volume
Inheritance and Polymorphism

Once you have mastered the basics: creating Classes, Fields, and Methods, and using dot notation in your program, you can use the OOP mode in RAPTOR to create and run more complicated programs.

You create child classes that inherit from a parent class in the UML screen. Figure 13 (following page) shows the association between a parent Class named Animal and two child Classes (subclasses) named Frog and Snake. Use the New Association button to set the inheritance between the parent and child, as indicated in Figure 13.

In this example, Frog and Snake inherit the showAnimalStuff() Method from Animal but each child class has its own Method for makeSound() and showAnimalType(). The OOP characteristics of both polymorphism and inheritance are demonstrated by this example.

[Special thanks to George L. Marshall, Jr. from Calhoun Community College at the Research Park Campus in Huntsville, Alabama for the Animal example.]
Figure 13 Child Classes inherit from the Parent Class

By combining all the features of RAPTOR’s OOP mode and all that you have learned in this text about object-oriented programming, it is possible to create some interesting and sophisticated programs.