**LAB – 12**  
  
**Task1:**  
  
Define a pure abstract base class called **BasicShape**. The BasicShape class should have the  
  
following members:  
  
**Private Member Variable:**  
  
area, a double used to hold the shape's area.  
  
  
  
**Public Member Functions:**  
  
getArea. This function should return the value in the member variable area.  
  
calcArea. This function should be a pure virtual function.  
  
  
  
Next, define a class named **Circle**. It should be derived from the BasicShape class. It should have the following members:  
  
**Private Member Variables:**  
  
centerX, a long integer used to hold the x coordinate of the circle’s center.  
  
centerY, a long integer used to hold the y coordinate of the circle’s center.  
  
radius, a double used to hold the circle's radius.  
  
  
  
**Public Member Functions:**  
  
constructor—accepts values for centerX, centerY, and radius. Should call the overridden calcArea  
  
function described below.  
  
getCenterX—returns the value in centerX.  
  
getCenterY—returns the value in centerY.  
  
calcArea—calculates the area of the circle (area = 3.14159 \* radius \* radius) and stores the result  
  
in the inherited member area.  
  
  
  
Next, define a class named Rectangle. It should be derived from the BasicShape class. It should have the following members:  
  
  
  
**Private Member Variables:**  
  
width, a long integer used to hold the width of the rectangle.  
  
length, a long integer used to hold the length of the rectangle.  
  
  
  
**Public Member Functions:**  
  
constructor—accepts values for width and length. Should call the overridden calcArea function described below.  
  
getWidth—returns the value in width.  
  
getLength—returns the value in length.  
  
calcArea—calculates the area of the rectangle (area = length \* width) and stores the result in the inherited member area.  
  
  
  
After you have created these classes, create a driver program that defines a Circle object and a Rectangle object. Demonstrate that each object properly calculates and reports its area.  
  
  
**Task2:**  
**Radio frequency identification** (RFID) chips are small tags that can be placed on a product. They behave like wireless barcodes and can wirelessly broadcast an identification number to a receiver. One application of the RFID chips is to use them to aid in the logistics of shipping freight. Consider a shipping container full of items. Without RFID chips a human has to manually inventory all of the items in the container to verify the contents. With an RFID chip attached to the shipping container, the RFID can electronically broadcast to a human the exact contents of the shipping container without human intervention.  
  
To model this application, write a base class *ShippingContainer* that has a **container ID number** as an integer. Include member functions to set and access the ID number. Add a virtual function called **getManifest**. The purpose of this function is to return the contents of the shipping container.  
  
Create a derived class called *ManualContainer* that represents the manual method of inventorying the container. In this method, a human simply attaches the textual description of all contents of the container. For example the description might be “4 crates of apples. 10 crates of pears.” Add a new class variable of type string to store the manifest. Add a function called **setManifest** that sets this string. Override the **getManifest** function so that it returns this string.  
  
Create a second derived class called *RFIDShippingContainer* that represents the RFID method of inventorying the container. To simulate what the RFID chips would compute, create an add function to simulate adding an item to the container. The class should store a list of all added items (as string) and their quantity using the data structures of your choice. For example, if the add function were invoked three times as follows:  
rfidContainer.add(“crate of pears”);  
rfidContainer.add(“crate of apples”);  
rfidContainer.add(“crate of pears”);  
  
At this point the data structure should be storing a list of two items: crate of apples and crate of pears. The quantity of apples is one and the quantity of pears is two. Override the **getManifest** function so that it returns the string of all items that is built by traversing the list of items. In the above example, the return string would be “2 crate of pears. 1 crate of apples.”  
  
Finally write a main program that creates an array of pointers to 6 ShippingContainer objects. Instantiate the array with 3 ManualSjippingContainer objects and 3 RFIDShippingContainer objects. For the ManualShippingContainer objects you will have to invoke setManifest to set the contents. For the RFIDShippingContainer objects you will have to invoke add to set the contents.   
  
Finally write a loop that iterates through all the ShippingContainer pointers and outputs each object’s manifest along with the shipping container id.