Early in our [introduction](https://www.eimacs.com/eimacs/mainpage?epid=E2346018901&cid=162149) to classes and objects, we implemented an APPoint class to represent points on a Cartesian plane. In [Exercise 101](https://www.eimacs.com/eimacs/mainpage?epid=E2134229576&cid=162149#Exe094), you implemented a static method ptDistance for calculating the distance between two points. The following replacement definition includes the ptDistance method as a class method of APPoint:

public class APPoint   
{   
  private double myX;   
  private double myY;   
  
  public APPoint( double x, double y )   
  {   
    myX = x;   
    myY = y;   
  }   
  
  public void move( double x, double y )   
  {   
    myX += x;   
    myY += y;   
  }   
  
  public double getX()    
  {   
    return myX;   
  }   
  
  public void setX( double x )   
  {   
    myX = x;   
  }   
  
  public double getY()   
  {   
    return myY;   
  }   
  
  public void setY( double y )   
  {   
    myY = y;   
  }   
  
  public static double ptDistance( APPoint p, APPoint q )   
  {   
    double xdiff = Math.abs( p.getX() - q.getX() );   
    double ydiff = Math.abs( p.getY() - q.getY() );   
    double dist = Math.sqrt( ( xdiff \* xdiff )   
                              + ( ydiff \* ydiff ) );   
    return (int)( 100.0 \* dist ) / 100.0;   
  }   
}

We would now like to add a new instance method, circleArea, to the APPoint class that takes an APPoint object d as its only argument and calculates the area of the circle that passes through d and is centered at the point represented by the instance, that is, at the point with coordinate (myX,myY).

The area of a circle may be calculated as (PI \* r \* r) where r is the radius of the circle, so we can define circleArea as follows:

  public double circleArea( APPoint d )   
  {   
    double r; // radius   
  
    r = ptDistance( /\* ... ??? ... \*/, d );   
  
    return (Math.PI \* r \* r);   
  }

As we indicated in the method body, there is a small problem in calculating the radius. The ptDistance method requires two APPoints, one of which should be the APPoint whose circleArea method is being called, and the other of which should be the APPoint d. One solution is to generate a new APPoint using the coordinate (myX,myY) like this:

  public double circleArea( APPoint d )   
  {   
    double r; // radius   
  
    r = ptDistance( new APPoint( myX, myY ), d );   
  
    return (Math.PI \* r \* r);   
  }

It seems strange, however, that an existing APPoint object should have to make a copy of itself just so that it can call its own ptDistance method. Why not just pass itself as the argument to ptDistance?

Java provides a keyword for just such a purpose. The keyword this may be used by an instance to pass itself as an argument to one of its own methods. So the preferred way to define circleArea is as follows:

  public double circleArea( APPoint d )   
  {   
    double r; // radius   
  
    r = ptDistance( this, d );   
  
    return (Math.PI \* r \* r);   
  }

In the following program, the keyword this is used by an instance of the T class to insert itself into an ArrayList of T instances:

public class T   
{   
  public T()   
  {   
  }   
  
  public void insertMe( ArrayList<T> a )   
  {   
    a.add( this );   
  }   
}   
  
public class MainClass   
{   
  public static void main( String[] args )   
  {   
    T t1 = new T(), t2 = new T();   
    ArrayList<T> r = new ArrayList<T>();   
  
    t1.insertMe( r );   
    System.out.println( r.size() );   
  
    t1.insertMe( r );   
    System.out.println( r.size() );   
       
    t2.insertMe( r );   
    System.out.println( r.size() );   
        
    t2.insertMe( r );   
    System.out.println( r.size() );   
  }   
}

1   
2   
3   
4

The keyword this may be thought of as a variable in which Java stores the instance whose method body is currently being executed. This "variable" is only visible while the instance method's body is being evaluated.

Important Note: this is not visible during execution of a class method. You can only use the keyword this in the body of an instance method.

If you examine the output from the program above, you will notice that objects are inserted in the ArrayList regardless of whether or not they are already present. To avoid the possibility of repeats, we can modify the insertMe as shown below. In this new version of the method, each of the existing elements of the ArrayList is compared to the current instance (that is, the value of this) using the identity operator. ([Recall](https://www.eimacs.com/eimacs/mainpage?epid=E2336183816&cid=162149#EqualObjects) that, when the operands to == are objects, it returns true if and only if the objects are actually the same instance.) In this modified program, the current instance adds itself to the Arraylist — once again using this — only if it is not already there.

public class T   
{   
  public T()   
  {   
  }   
  
  public void insertMe( ArrayList<T> a )   
  {   
     for ( T t : a )   
       if ( t == this )   
         return;   
  
    a.add( this );   
  }   
}   
  
public class MainClass   
{   
  public static void main( String[] args )   
  {   
    T t1 = new T(), t2 = new T();   
    ArrayList<T> r = new ArrayList<T>();   
  
    t1.insertMe( r );   
    System.out.println( r.size() );   
  
    t1.insertMe( r );   
    System.out.println( r.size() );   
       
    t2.insertMe( r );   
    System.out.println( r.size() );   
        
    t2.insertMe( r );   
    System.out.println( r.size() );   
  }   
}

1   
1   
2   
2

Notice the empty return statement in the definition of the insertMe method. [Recall](https://www.eimacs.com/eimacs/mainpage?epid=E2104716560&cid=162149#EmptyReturn) that the effect of executing such a statement is to stop execution of the method without returning anything and in this case, because of the order in which the statements appear, without adding anything to the ArrayList. Run this program and you will see from the output that each of the instances t1 and t2 is added to the ArrayList only once.