Lecture #11-12
Objects & Classes

1. **Graphical User Interface (GUI)**

   Interface structures designed to facilitate user interfaces with Java, e.g.,

   - **import javax.swing.JOptionPane;**
   - **JOptionPane.showMessageDialog(null, "Hello World",**
     JOptionPane.INFORMATION_MESSAGE);

2. **Object**

   An object is a conceptual representation of a real world entity

   - **State of the object – data fields – properties – values**
   - **Behavior of object – set of methods -- ask an object to perform a task**

   For example,

   Circle object

   - **radius – data field – variables**
   - **getArea(); -- method // behavior of Circle – compute area**

3. **Class**

   A template that defines the general properties of a whole set of similar objects

   - **An object is an instance of a class; creating an instance is call instantiation**
   - **Many similar objects can be created from the same class**

4. **Constructors**

   A special type of method, used for

   - **initializing an object upon instantiation**

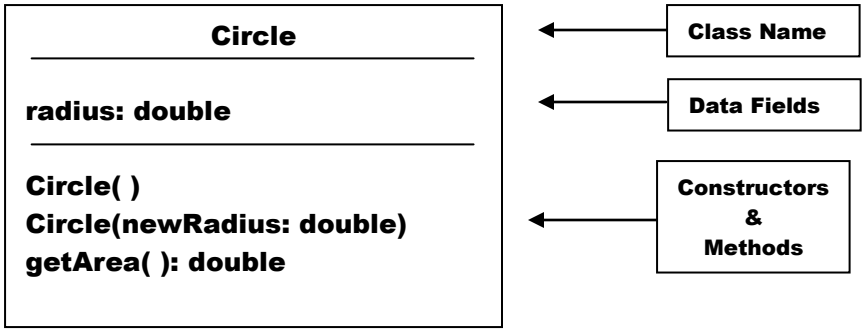
```
class Circle
{
  double radius = 1.0;
  Circle( ) { }
  Circle(double newRadius)
  {
    radius = newRadius;
  }
  double getArea( )
  {
    return radius * radius * Math.PI;
  }
}
```

Constructors

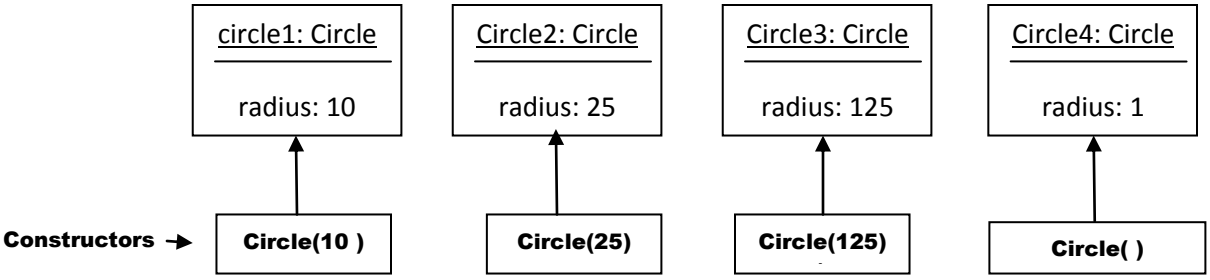
Method

5. Unified Modeling Language (UML) Notation

Class Diagram



Object Diagrams



UML Notation

- **Contractors**

  ClassName(parameterName: parameterType)

- **Methods**

  methodName(parameterName: parameterType): returnType

6. Constructors (Revisited)

- **are used to construct, i.e., create, objects**
- **must always have the same name as the defining class**
- **are used to initialize an object**
- **do not have a return type**
- **can be overloaded, i.e.,**

  multiple constructors with the same name

  but with different signatures, e.g.,

  Circle( )

  Circle(double newRadius)

- **are invoked by using the new operator when an object is to be created,**

  e.g., new Circle(125);

  new Circle( );

  etc.

7. No-Argument Constructors

A class definition normally provides a constructor with no arguments, i.e., a no-arg constructor or no-argument constructor

If the class definition does not provides any constructors, Java provides a no-arg constructor with an empty body, i.e., such a constructor, a default constructor, is implicitly declared in the class

The default constructor is automatically provided only if no constructors of any kind are explicitly declared in the class

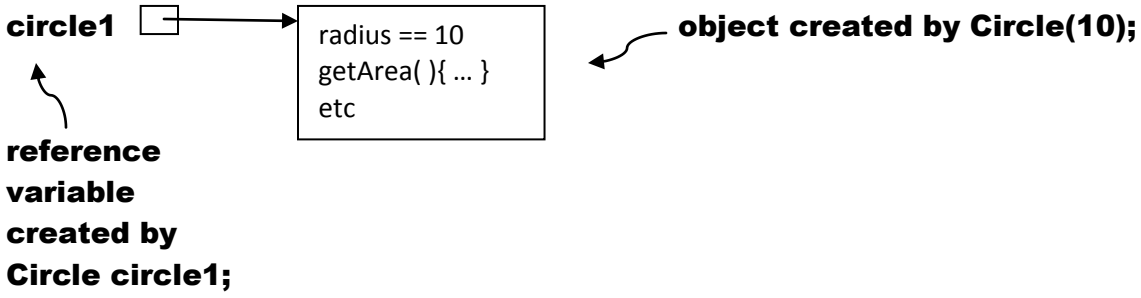
8. Creating Objects

```
Circle circle1 = new Circle(10);
Circle circle2 = new Circle(25);
Circle circle3 = new Circle(125);
Circle circle4 = new Circle( );
```

The class is used to create

- **a reference type ⟶ Circle circle1;**
- **instantiate a new object ⟶ new Circle(10);**
- **assign the reference value of the new object to the reference type ⟶ circle1 = new Circle(10);**

circle1 is a variable that holds a reference to the circle object created by Circle(10);



9. Accessing the Object's Data & Methods

- **dot operator**, i.e., **object member access operator**

```
circle1.radius = 15; // changes the value of the radius from 10 to 15
circle1.radius *= 15; // changes the value of the radius from 15 to 225
circle1.radius++; // changes the value of the radius from 225 to 226

double area = circle1.getArea( ); // assuming radius = 226
        invokes the getArea( ) method in circle1 and
        returns the computed value of 226*226*3.1415927 to
        the variable area, i.e., area == 160459.99
```

- **instance variables**
  variables that are dependent upon a particular instance, i.e., on a particular object
- **instance methods**
  methods that can only be invoked on a particular instance, i.e., on a particular object
- **calling object**
  object that is used to invoke instance variables &/or instance methods

In the class Circle,

- **radius** is an **instance variable**
- **getArea( )** is an **instance method**
- **circle1** is the **calling object**

Remark:

Later, we will see that there are both variables and methods that are not dependent upon an instance of a class, i.e., an object, but that can be applied without creating an instance of the class.

For example, the Math class contains PI, random( ), and sqrt( ) which we have already invoked without making an instance of the math class. These methods are defined to be **static** methods, this allows their use without an instance being created.

In fact we are prohibited to make an instance of the Math class, i.e., an object of type Math, this feature is implemented by a special use of defining constructors which we will see later.

10. Anonymous Objects

- an object created <u>without a reference variable</u> to hold the object, e.g.,

  ```
  new Circle(23);
        or
  System.out.println(new Circle(23).getArea( ));
  ```

- since an anonymous object has no reference variable to hold it, it is of <u>limited use</u>;
  it is <u>destroyed immediately after the invocation is completed</u>, e.g.,
  - new Circle(23);
    is destroyed immediately, it has no real effect on the encompassing program
  - System.out.println(new Circle(23).getArea( ));
    is destroyed immediately after the close of the println( ) statement; its only effect it the printing of the area of the object created by new Circle(23) object

11. Declaring Classes & Creating Objects

```
public class testCircle1      <-- only one class may contain a main method
{
  public static void main(String [ ] args)
  {
    Circle1 myCircle = new Circle1(5.1);
    System.out.println(myCircle1.radius + " :  " + myCircle1.getArea( ));
    Circle1 yourCircle = new Circle1( );
    System.out.println(yourCircle1.radius + " :  " + yourCircle1.getArea( ));
    yourCircle.radius = 100;
    System.out.println(yourCircle1.radius + " :  " + yourCircle1.getArea( ));
  }
}
```

```
class Circle1 <--  not declared public, does not contain a main method
{
  double radius;
  Circle1( ) { radius = 1.0; }
  Circle1(double newRadius) { radius = newRadius; }
  double getArea( ) { return radius * raius * math.PI; }
}
```

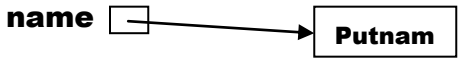
12. Declaring Classes & Creating Objects (continued)

```
public class Circle1 ← only one class and it contains the main method
{
  public static void main(String [ ] args)
  {
    Circle1 myCircle = new Circle1(5.0);
    System.out.println(myCircle1.radius + " :  " + myCircle1.getArea( ));
    Circle1 yourCircle = new Circle1( );
    System.out.println(yourCircle1.radius + " :  " + yourCircle1.getArea( ));
    yourCircle.radius = 100;
    System.out.println(yourCircle1.radius + " :  " + yourCircle1.getArea( ));
  }
  double radius;
  Circle1( ) { radius = 1.0; }
  Circle1(double newRadius) { radius = newRadius; }
  double getArea( ) { return radius * raius * math.PI; }
}
```

13. Reference Data Fields & the null Value

```
class Student
{
  String name;   ← predefined Java class; reference data field
  int age;
  boolean isScienceMajor;   ← primitive data types
  char gender;
}
```

While the objective of the String reference variable in the Student class is ultimately to hold a person's name, e.g.,



if the reference variable has not yet been assigned a value, it is given a default value, which for String reference variables is null.

Default Values

- **Reference Variables -- null**
- **Numeric Variables -- 0**
- **Boolean Variables -- false**
- **Character variables – '\u000'**
- **Java assigns NO default values to Local Variables inside a Method**

14. NullPointerExceptions – Run Time Error
occurs when a method is invoked on a reference variable with a null value

15. Primitive Variables versus Reference Variables
See Liang page 237-238

Remark:

Deletion of Objects

- **assign the object's reference variable the null value**
- **objects not held by a method or a reference variable are no longer useful and are classified as garbage**
- **the java runtime system detects garbage and automatically reclaims the space it occupies, i.e., the Java Virtual Machine (JVM) will automatically delete the object and reclaim the memory space if the object is no longer referenced by a reference variable**

16. Java Library Classes

java.util.Date class **System.currentTimeMillis( );**

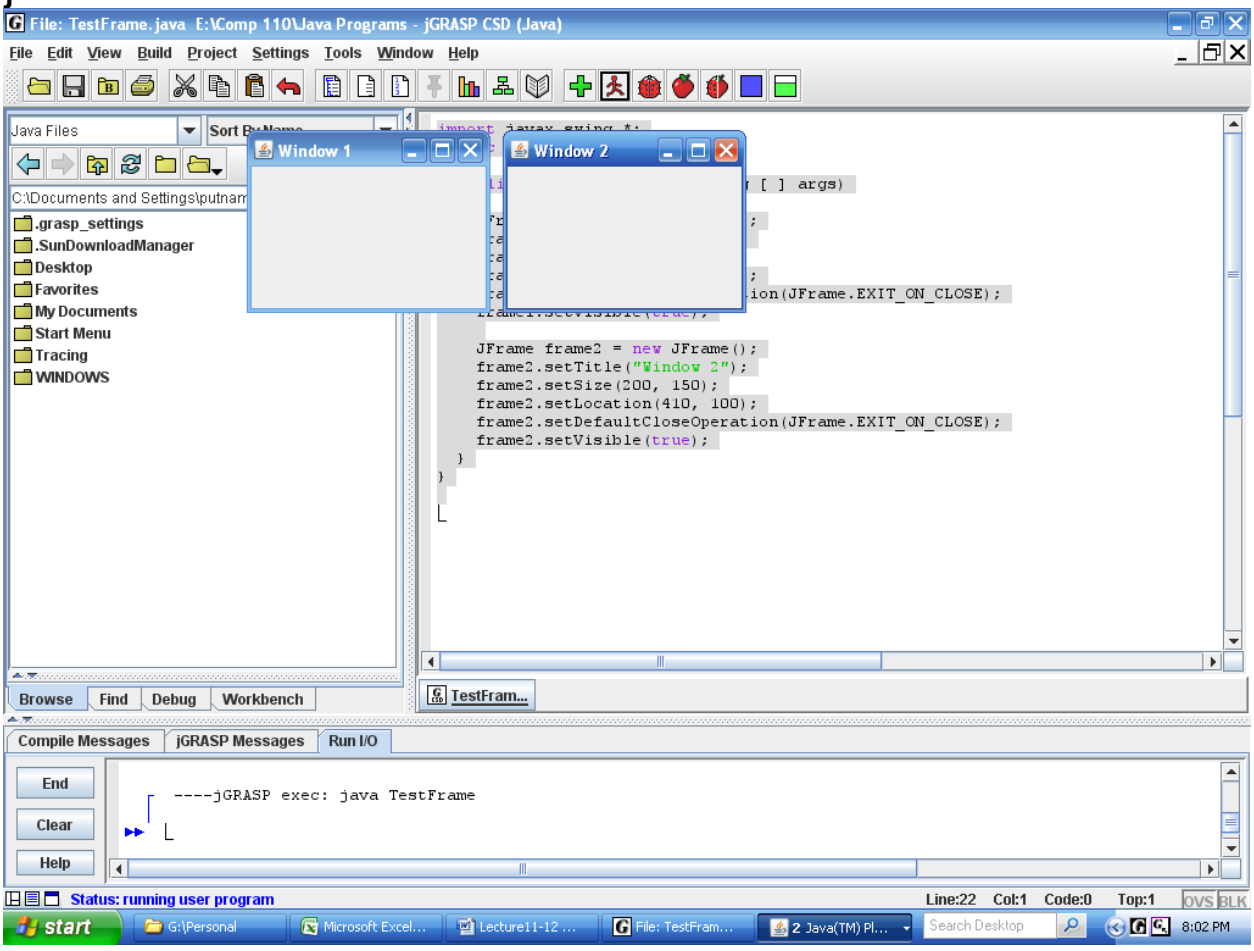
```
java.util.Date date = new java.util.Date( );
date.getTime( );        //returns time
date.setTime(150);      //sets time from 1 Jan 1970 GMT
date.toString( );
```

java.util.Random class **Math.random( );**

```
java.util.Random random1 = new java.util.Random( );
random1.nextInt(10);
random1.nextInt( );
random1.nextLong(10);
random1.nextDouble(10);
random1.nextFloat(10);
random1.nextBoolean(10);
```

17. Displaying GUI Components

```
import javax.swing.*;
public class TestFrame
{
  public static void main(String [ ] args)
  { JFrame frame1 = new JFrame();
    frame1.setTitle("Window 1");
    frame1.setSize(200, 150);
    frame1.setLocation(200, 100);
    frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame1.setVisible(true);

    JFrame frame2 = new JFrame();
    frame2.setTitle("Window 2");
    frame2.setSize(200, 150);
    frame2.setLocation(410, 100);
    frame2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame2.setVisible(true);
  }
}
```



```
import javax.swing.*;
public class GUIComponents
{
  public static void main(String [ ] args)
  {
    JButton jbtOK = new JButton("OK");
    JLabel jlblName = new JLabel("Enter Name: ");
    JTextField jtfName = new JTextField("Type Name Here: ");
    JCheckBox jchkBold = new JCheckBox("Bold");
    JRadioButton jrbRed = new JRadioButton("Red");
    JComboBox jcboColor = new JComboBox(new String[ ] {"Red", "Green", "Blue"});

    JPanel panel = new JPanel();
    panel.add(jbtOK);
    panel.add(jlblName);
    panel.add(jtfName);
    panel.add(jchkBold);
    panel.add(jrbRed);
    panel.add(jcboColor);

    JFrame frame = new JFrame();
    frame.setTitle("Show GUI Components");
    frame.setSize(450,450);
    frame.setLocation(200, 100);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

18. Static Variables, Constants & Methods

a. **Instance Variables**
   - depend upon a particular instance of a class
   - are not shared by objects of the same class

b. **Static (Class) Variables**
   - common storage (memory) location
   - all objects of the same class share the common value
   - any object of the same class may change the value

c. **Instance Methods**
   - depend upon the existence of an object for execution

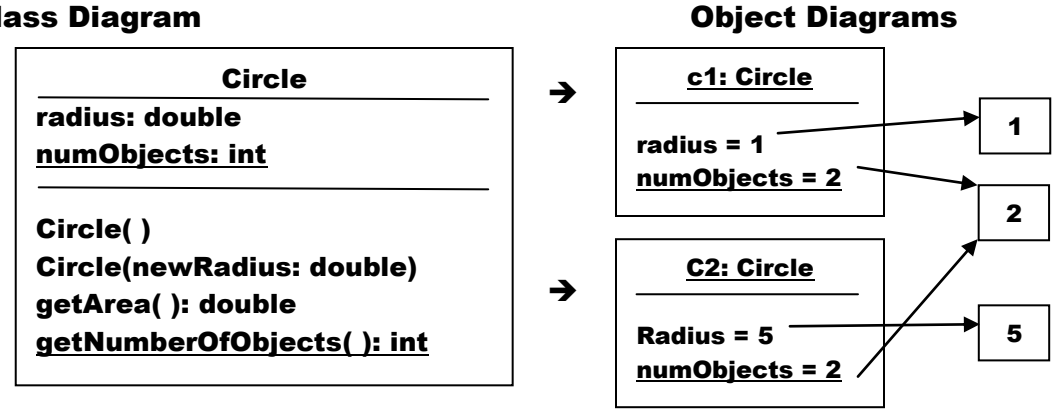
d. **Static (Class) Methods**
   - may be called without creating an instance of the class

```
public class TestCircle2
{
  public static void main(String [ ] args)
    {
      Circle2  c1 = new Circle2( );
      System.out.println("Before c2 creation" );
      System.out.println("c1.radius:  " + c1.radius + "Object #:  " +
                                                    c1.numObjects);
      Circle2  c2 = new Circle2(5);
      C1.radius = 9;
      System.out.println("After c2 creation with c1.radius = 9" );
      System.out.println("c1.radius:  " + c1.radius + "Object #:  " +
                                                    c1.numObjects);
      System.out.println("c2.radius:  " + c2.radius + "Object #:  " +
                                                    c2.numObjects);
    }
}
```

```
class Circle2
{
  double radius;                    <-- instance variable
  static int numObjects = 0;        <-- class (static) variable
  Circle2( ) { radius = 1.0; numObjects++; }
  Circle2(double newRadius)
  {
    radius = newRadius;
    numObjects++;
  }
  static int getNumberOfObjects( )  <-- class (static) method
  {
     return numObjects;
  }
  double getArea( )                 <-- instance method
  {
    return radius*radius*Math.PI;
  }
}
```

Remark: The class variable **numObjects** can be referenced by a reference variable, e.g., c1.numObjects and c2.numObjects or by a class name, e.g., Circle2.numObjects. **Best Practices:** Use class names to reference class (static) variables and methods; this makes it easier to recognize that we are not referencing instance items.

e. UML Diagrams



f. Declaring Constants

- The "static" modifier allows the constant to be used without creating an instance of the container class
- The "final" modifier prohibits the changing of the value after declaration, i.e., creates a constant value

```
final static double PI = 3.14159265358979323846;
```

19. Visibility Modifiers

a. Public Modifier

- makes classes, methods & data fields accessible from any class

b. Private Modifier

- makes methods & data fields accessible only from within its own class
- cannot be used on a class definition, i.e., classes cannot be declared to be private

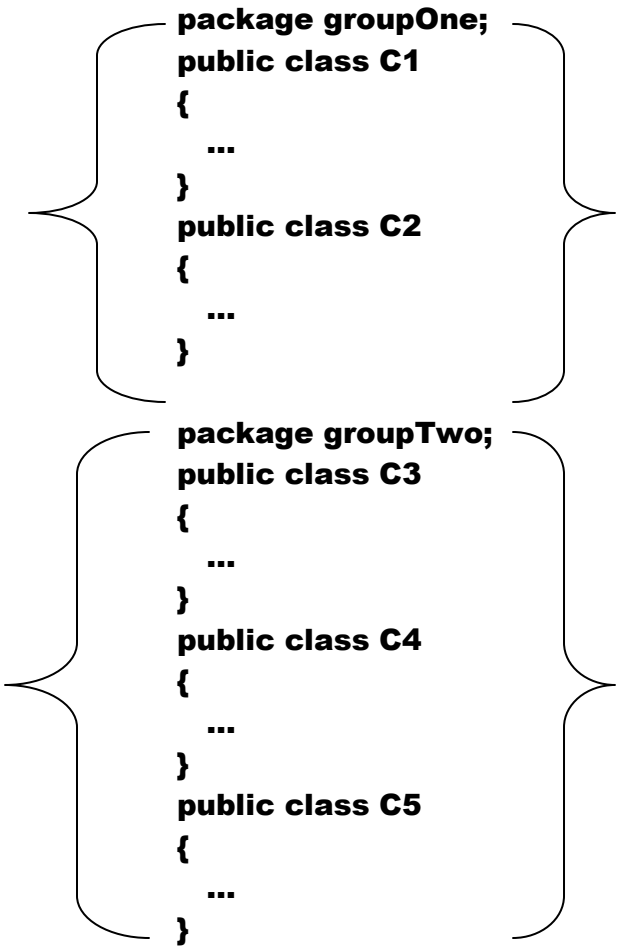
c. Default Modifier – not specified as either public nor private

- makes classes, methods & data fields accessible from any class in the same package
- known as package-access or package-private

Remark: Using the public/private modifiers on local variables i.e., variables declared within methods, will cause a compiler error.

d. Packages

- **Java mechanism used to organize classes**
- **the declaration of `package packageName;` must be on the first non-comment and first non-blank line of the Java program**
- **if a class is declared without a package statement, it is by default placed into a default package which consists of all classes in the same file**
- **see example in Liang pages 245-246**
- **a package declaration remains in force until another declaration is encountered, e.g.,**



```
package groupOne;
public class C1
{
   ...
}
public class C2
{
   ...
}
```

```
package groupTwo;
public class C3
{
   ...
}
public class C4
{
   ...
}
public class C5
{
   ...
}
```

e. Restricting Instantiation

In order to restrict the creation of objects from a class definition, create a private constructor that does nothing. For example the class java.lang.Math is designed to be a container class for a set of

methods, constants and class variables. Accordingly, it has incorporated in its definition the following constructor
Private Math( ) { }

20. Data Field Encapsulation
    a. private data fields -- precludes direct reference of data field from outside the class without ...
    b. public data manipulation methods
        i. get method – accessor – makes a private data field accessible
        ii. set method – mutator – allows the modification of a private data field

    c. get method (accessor) signatures
        i. public returnType getPropertyName( )
        ii. public boolean isPropertyName( )

    d. set method (mutator) signatures
        public void setpropertyName(dataType propertyValue)

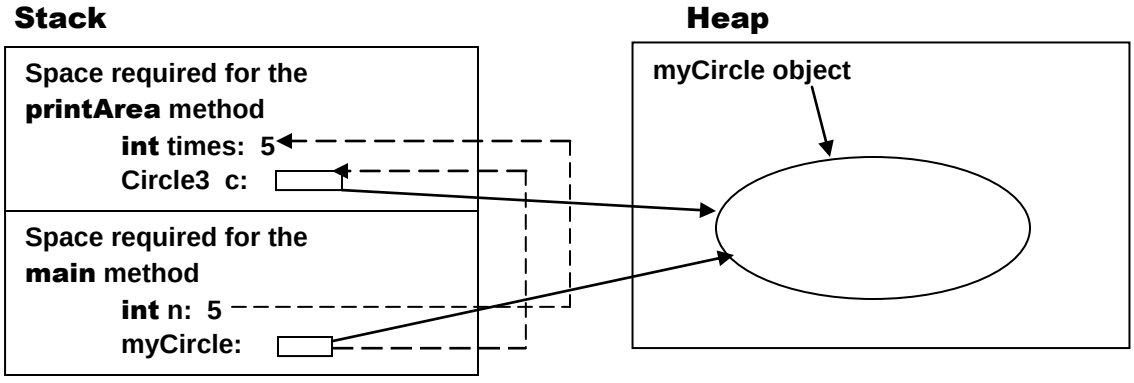
```
class Circle3
{
  private double radius;                    <-- encapsulated instance variable
  private static int numObjects = 0;        <-- encapsulated class (static) variable
  public Circle3( ) { numObjects++; }
  public Circle3(double newRadius)
  {
    radius = newRadius;
    numObjects++;
  }
  public double getRadius( ) { return radius; }   <-- public instance method

  public void setRadius(double newRadius)         <-- public instance method
  {
    radius = ( newRadius >= 0) ? newRadius : 0;
  }
  public static int getNumberOfObjects( ) { return numObjects; }   <-- public class (static) method

  public double getArea( ) { return radius*radius*Math.PI; }
}
```

public instance method

21. Passing Objects to Methods

Passing an Object to a Method
Is implemented by
Passing a Reference Variable of the Object to the Method

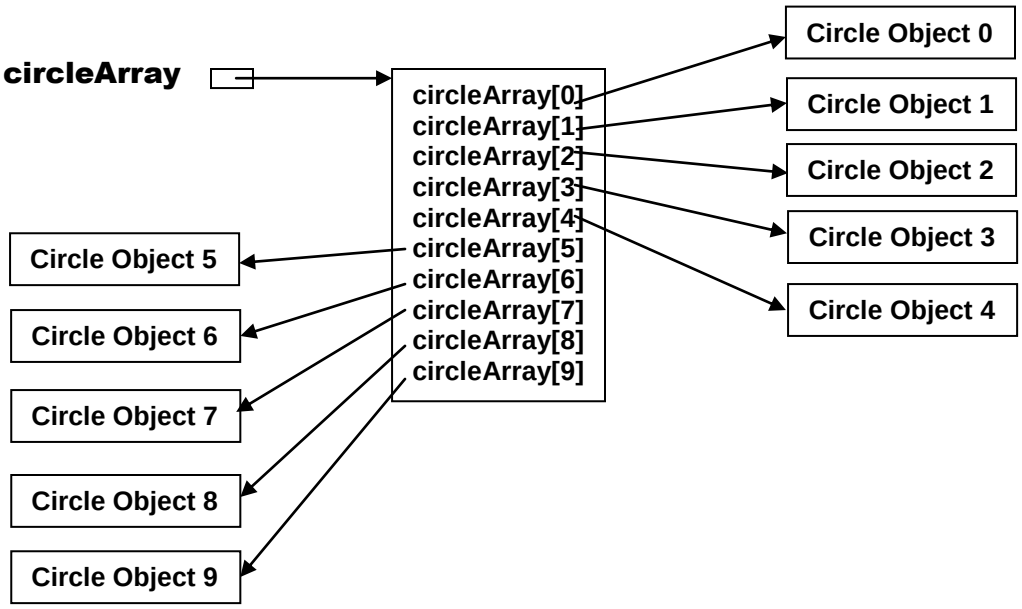
```
public class TPO
{
  public static void main (String [ ] args )
  {
    Circle3 myCircle = new Circle3(1);
    int n = 5;
    printAreas(myCircle, n);
    System.out.println("\n"  + Final Radius:  " + myCircle.getRadius( ) );
    System.out.println("Final n:  " + n);
  }
  public static void printAreas(Circle3 c, int times)
  {
    System.out.println("Radius:  \t\tArea" );
    while (times >= 1)
    {
       System.out.println(c.get.Radius() + "\t\t" + c.getArea());
       c.setRadius(c.getRadius( ) + 1);
       times- -;
    }
  }
}
```



pass by value of reference variable to an object
results in an effective
pass by sharing, i.e., it results in sharing the object

22. Array of Objects

```
Circle[ ] circleArray = new Circle[ 10 ];

for (int i = 0; i < circleArray.length; i++)
{
   circleArray[ i ] = new Circle( );
}
```



See Listing 7.10 page 251-252