Objects and Classes

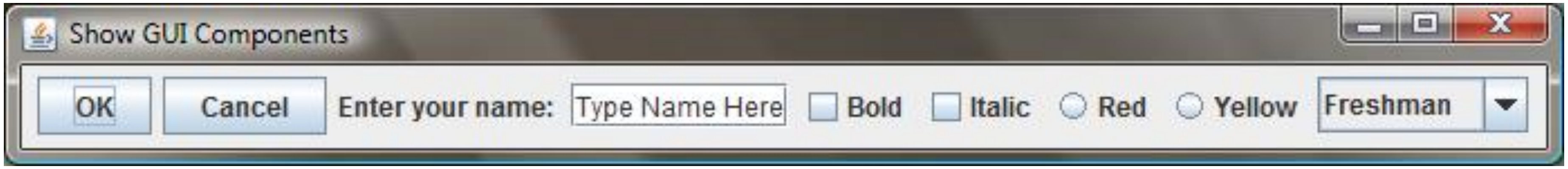
CSE 114, Computer Science 1

Stony Brook University

http://www.cs.stonybrook.edu/~cse114

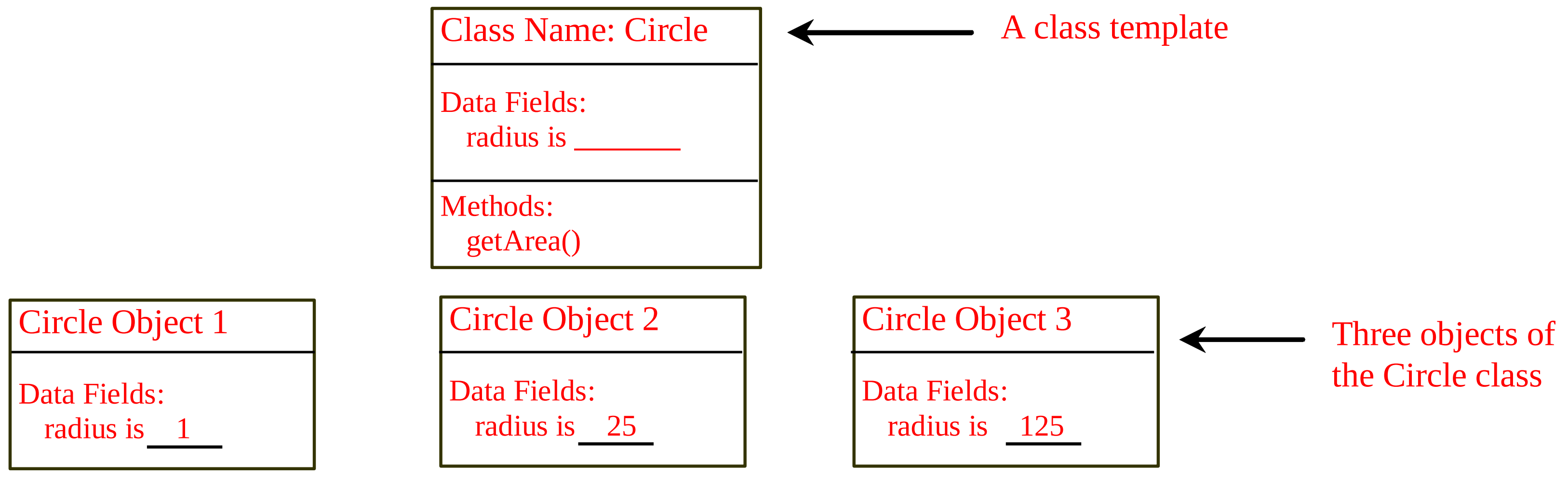
Opening Problem

- Develop a Graphical User Interface (GUIs)
  - need of multiple object instances of classes
    - 2 buttons
    - input fields
    - 2 check boxes
    - 2 choice boxes
    - lists



OO Programming Concepts

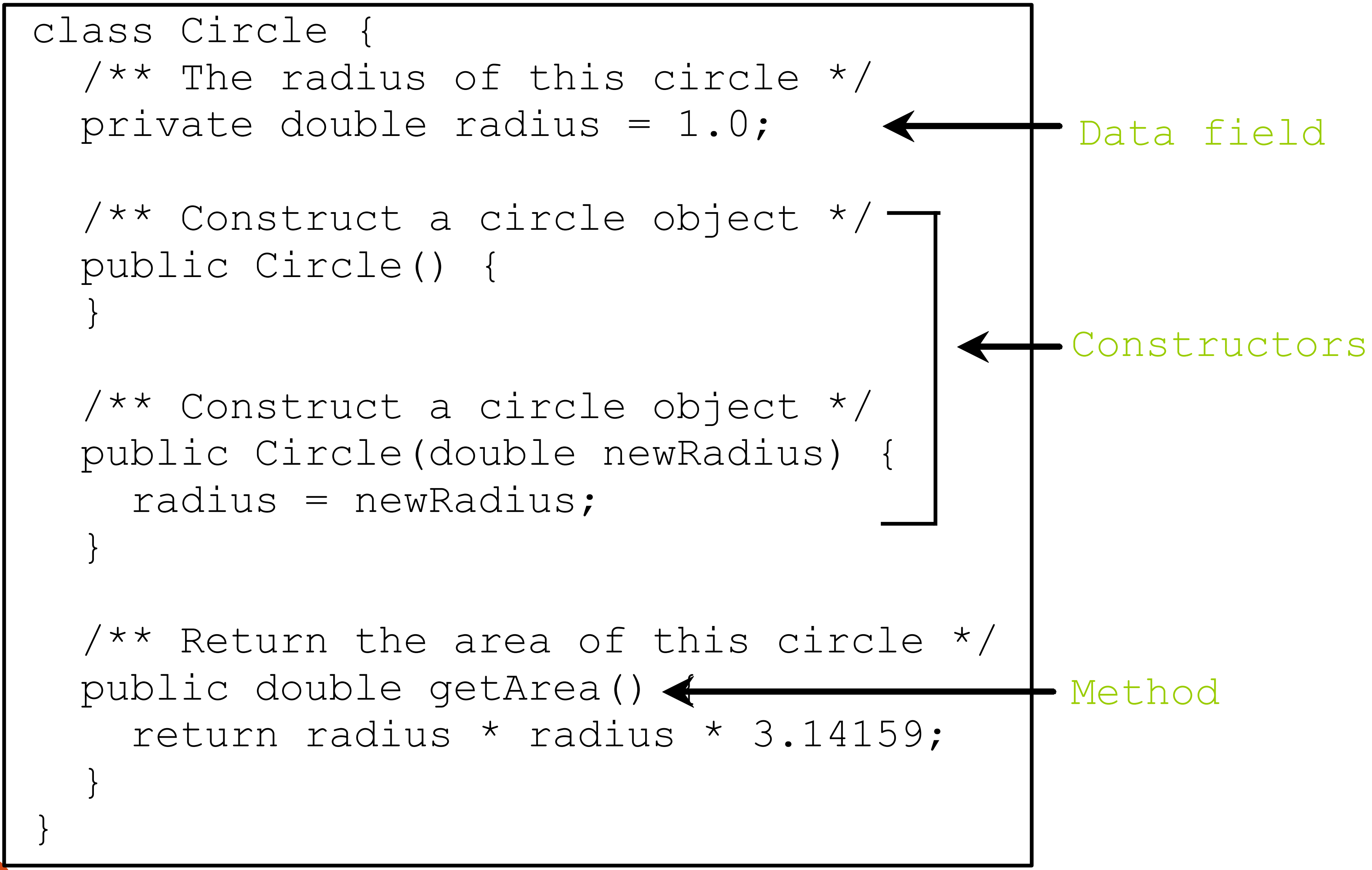
- An object represents an entity in the real world that can be distinctly identified.
- An object has a unique state and behaviors
  - the state of an object consists of a set of data fields (properties) with their current values
  - the behavior of an object is defined by a set of methods



Classes

- Classes are templates that define objects of the same type.
- A Java class uses:
  - variables to define data fields and
  - methods to define behaviors
- A class provides a special type of methods called **constructors** which are invoked to construct objects from the class

Classes

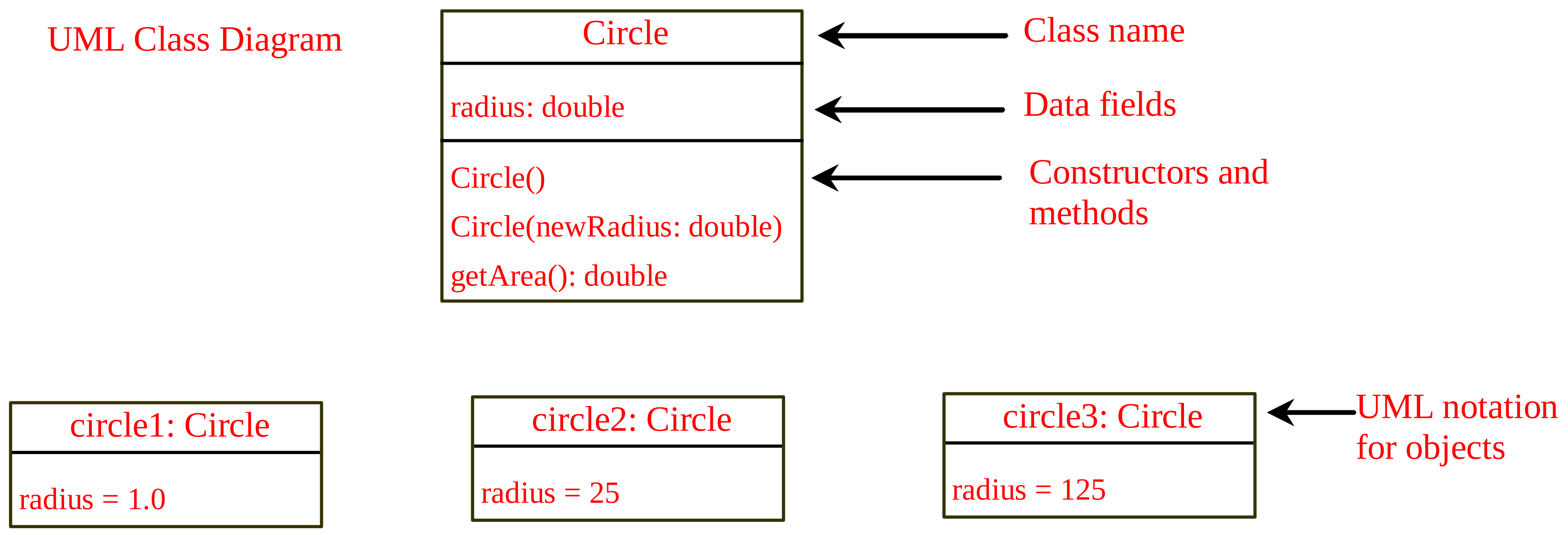


```java
class Circle {
  /** The radius of this circle */
  private double radius = 1.0;

  /** Construct a circle object */
  public Circle() {
  }

  /** Construct a circle object */
  public Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  public double getArea() {
    return radius * radius * 3.14159;
  }
}
```

Classes

```
public class TestCircle {

  public static void main(String[] args) {
    Circle circle1 = new Circle();
    Circle circle2 = new Circle(25);
    Circle circle3 = new Circle(125);

    System.out.println( circle1.getArea() );
    System.out.println( circle2.getArea() );
    System.out.println( circle3.getArea() );

    //System.out.println( circle1.radius );
    //System.out.println( circle2.radius );
    //System.out.println( circle3.radius );
  }

}
```

UML Class Diagram



Constructors

- Constructors must have the same name as the class itself.
- Constructors do not have a return type—not even void.
- Constructors are invoked using the new operator when an object is created – they initialize objects to **reference variables**:

```
ClassName o = new ClassName();
```

  - Example:

```
Circle myCircle = new Circle(5.0);
```

- A class may be declared without constructors: a no-arg **default constructor** with an empty body is implicitly declared in the class

Accessing Objects

- Referencing the object's data:

  `objectRefVar.data`

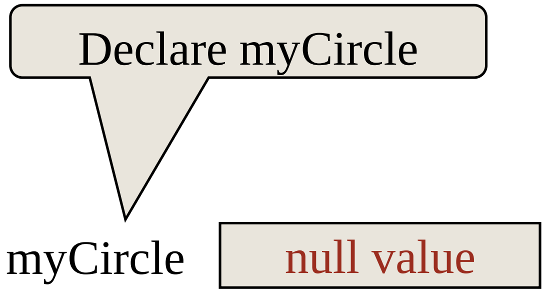
  - Example: `myCircle.radius`
- Invoking the object's method:

`objectRefVar.methodName(arguments)`

  - Example: `myCircle.getArea()`

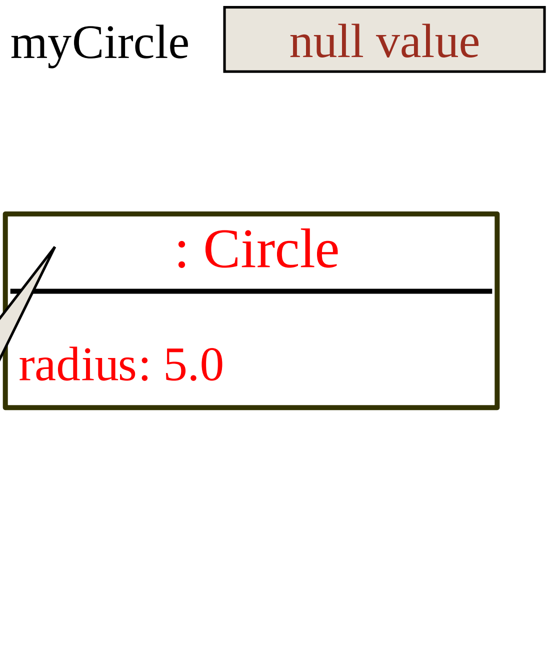
Using classes

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



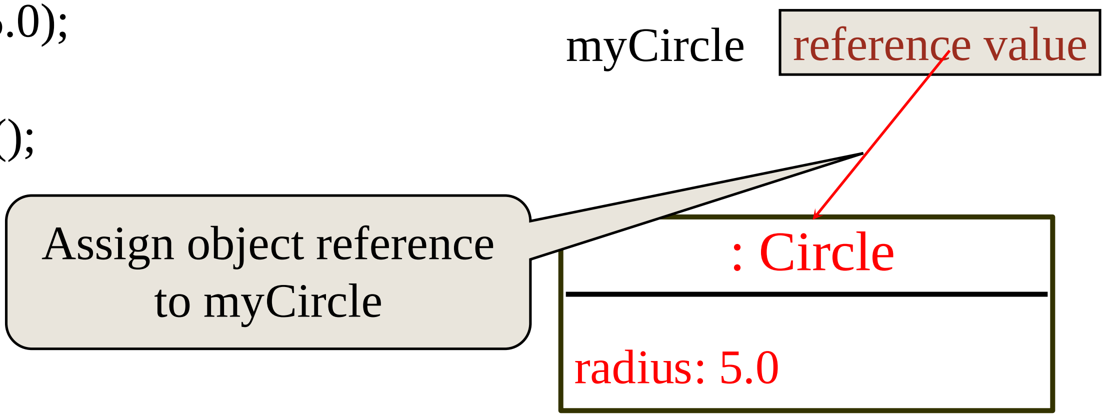
Using classes

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



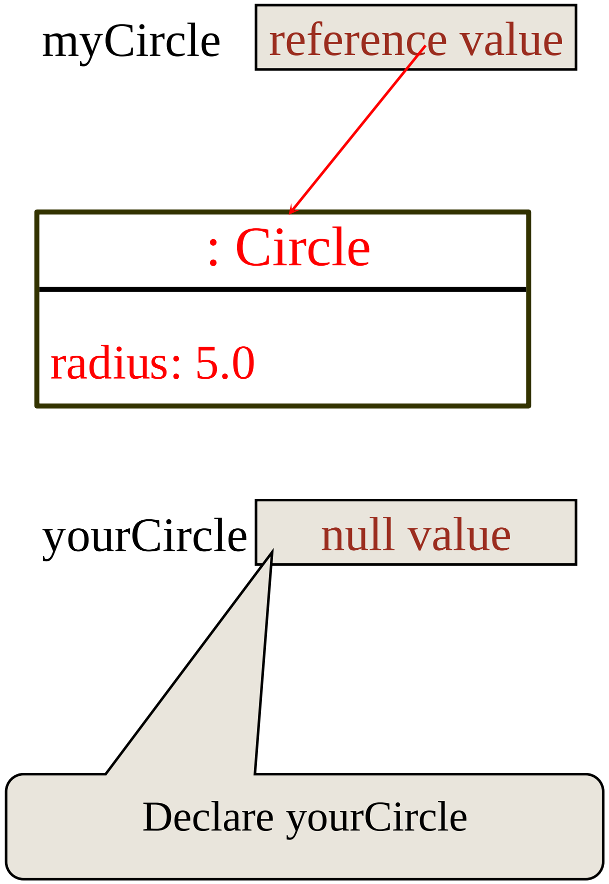
Using classes

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



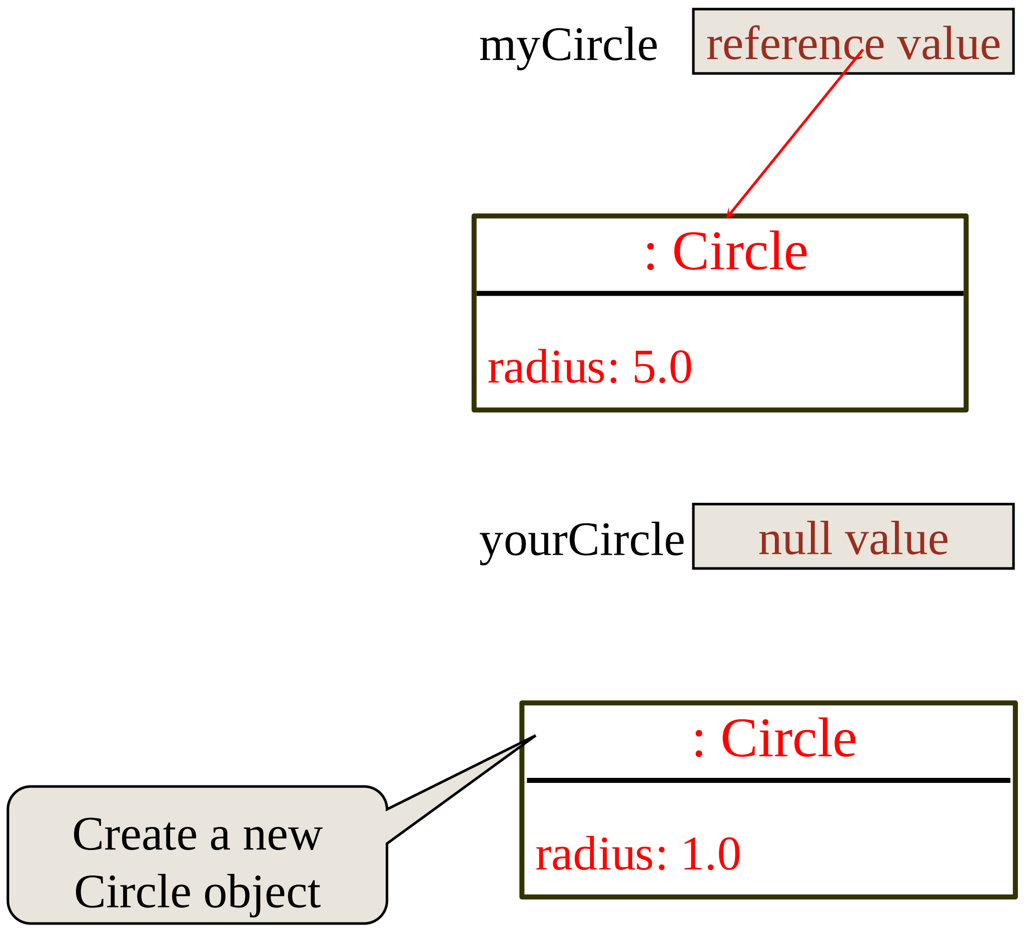
Using classes

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



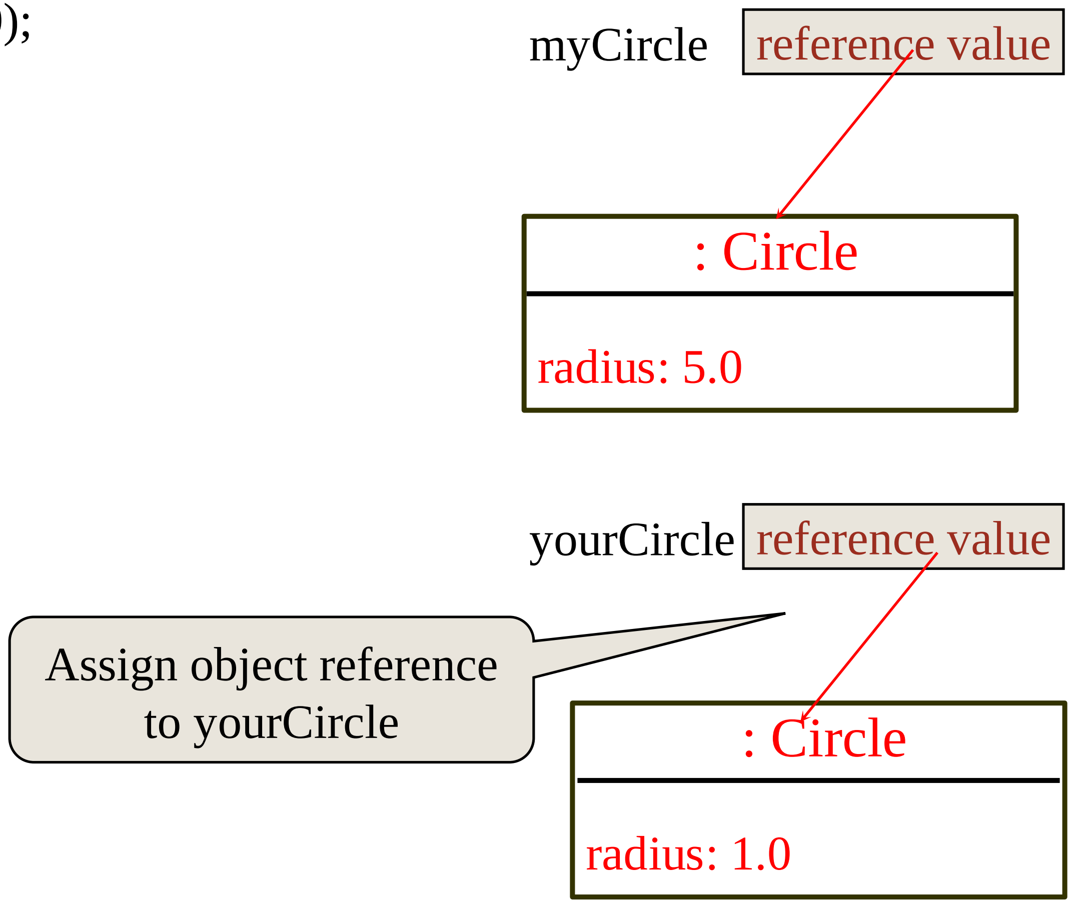
Using classes

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



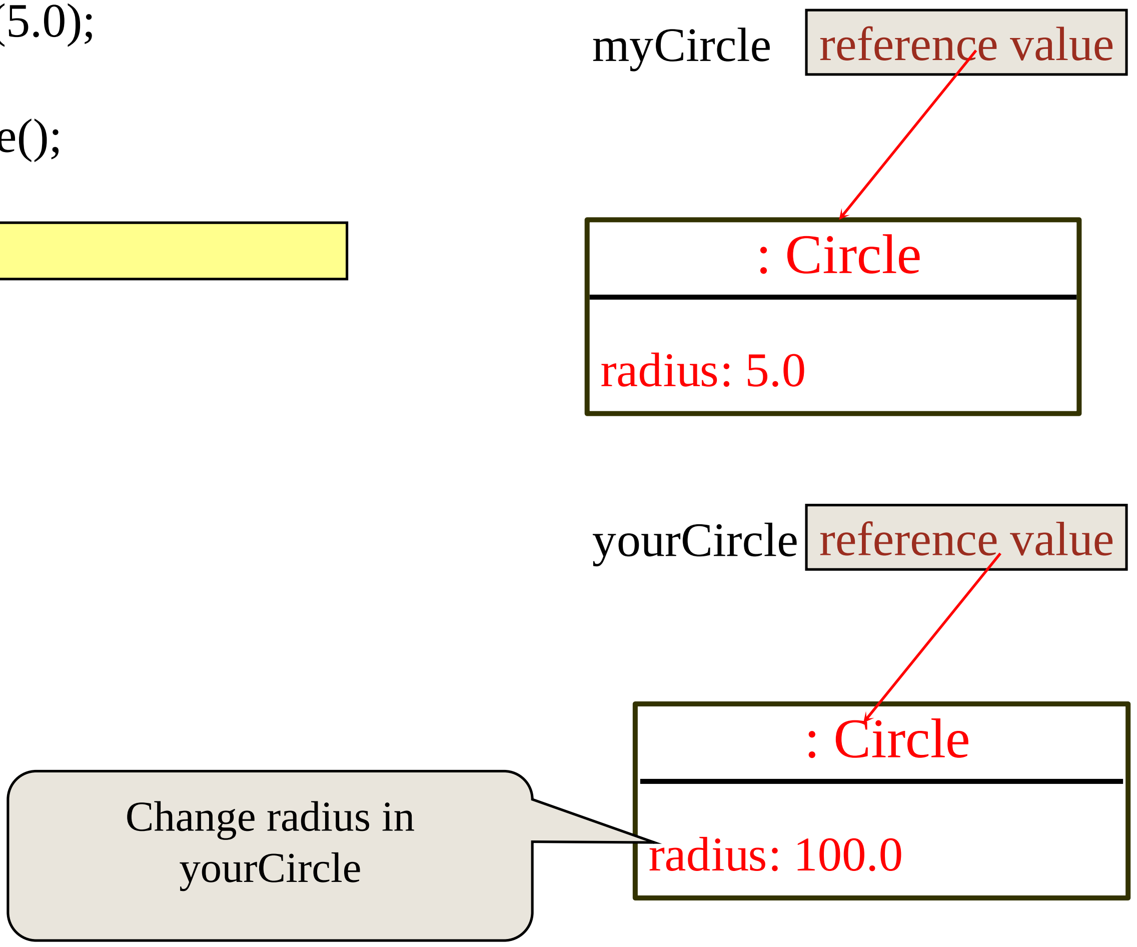
Using classes

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Using classes

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Static vs. Non-static methods

- Static methods:
  - Shared by all the instances of the class - not tied to a specific object.

```
double d = Math.pow(3, 2.5);
```

  - Static constants are final variables shared by all the instances of the class.
- Non-static methods must be invoked from an object:
  - Instance variables belong to a specific instance.
  - Instance methods are invoked by an instance of the class.

```
double d1 = myCircle.getArea();
double d2 = yourCircle.getArea();
```

Default values

Java assigns no default value to a local variable inside a method.

```
public class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```

Compilation error: the variables are not initialized

BUT it assigns default values to data fields!

Reference Data Fields

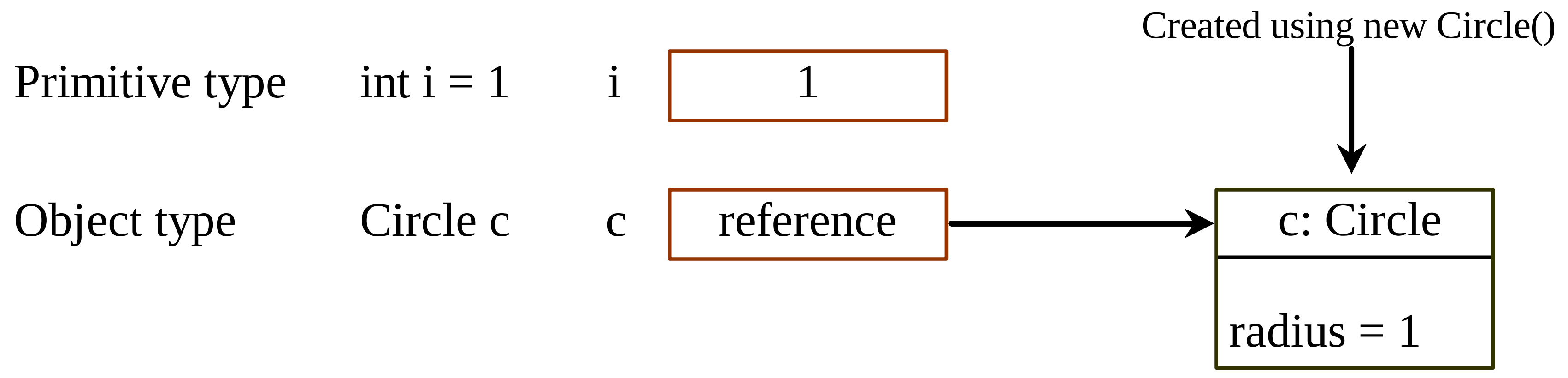
- The data fields can also be of reference types
- Example:

```
public class Student {
   String name; // name has default value null
   int age; // age has default value 0
   boolean isScienceMajor; // isScienceMajor has default value false
   char gender; // c has default value '\u0000'
}
```

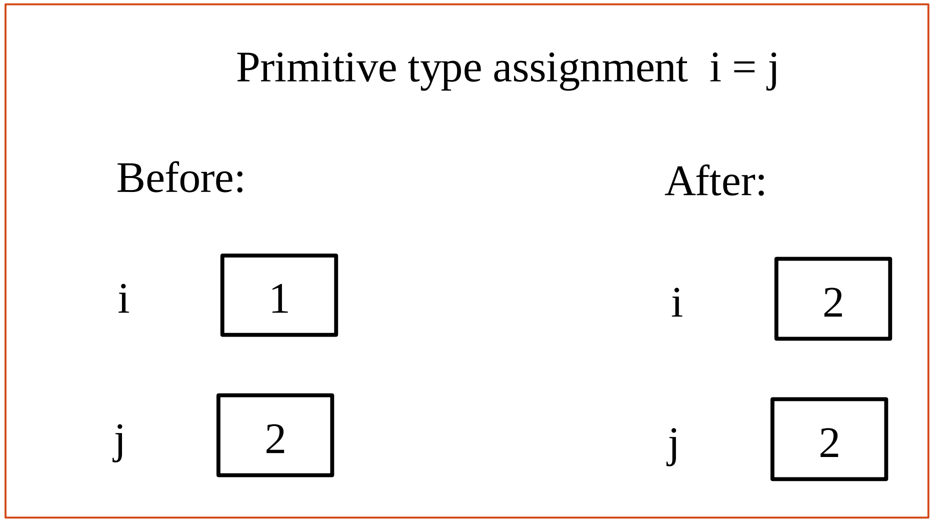
- If a data field of a reference type does not reference any object, the data field holds a special literal value: **null**.

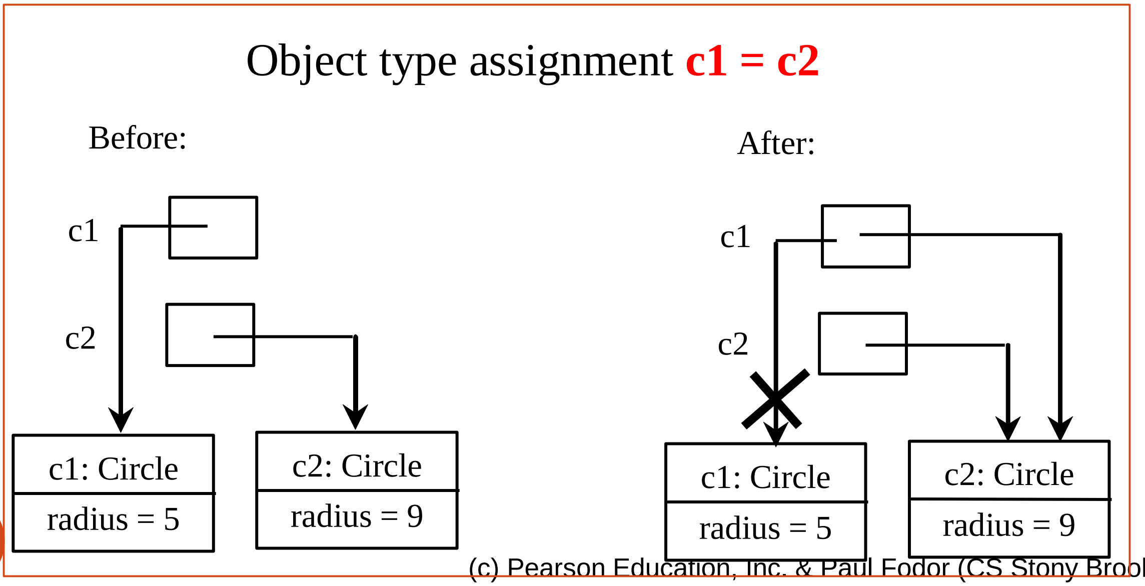
```
public class Test {
  public static void main(String[] args) {
    Student student = new Student();
    System.out.println("name? " + student.name);              // null
    System.out.println("age? " + student.age);                // 0
    System.out.println("isScienceMajor? " + student.isScienceMajor);
                                                              // false
    System.out.println("gender? " + student.gender);          //
  }
}
```

Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types





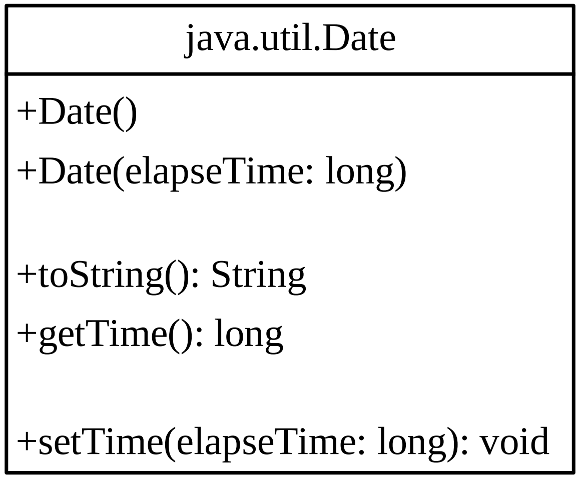
- The object previously referenced by c1 is no longer referenced – it is called *garbage*
- Garbage is automatically collected by JVM = *garbage collection*

The Date Class

Java provides a system-independent encapsulation of date and time in the java.util.Date class.

The toString method returns the date and time as a string

The + sign indicates public modifer



Constructs a Date object for the current time.

Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.

Returns a string representing the date and time.

Returns the number of milliseconds since January 1, 1970, GMT.

Sets a new elapse time in the object.

January 1, 1970, GMT is called the Unix time (or Unix epoch time)

```
java.util.Date date = new java.util.Date();
System.out.println(date.toString());
```

The Random Class

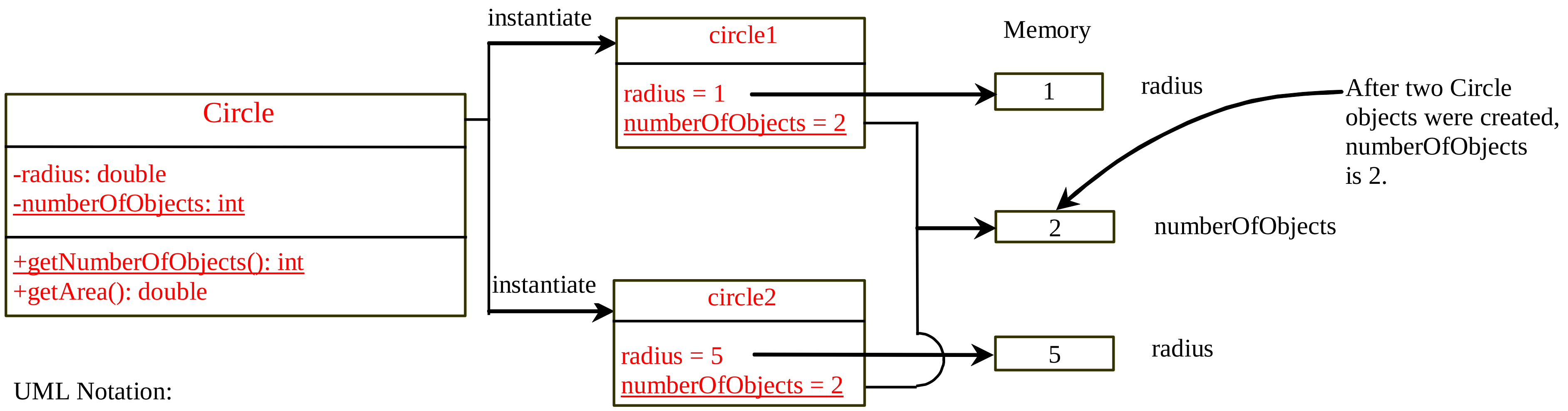
java.util.Random

java.util.Random	
+Random()	Constructs a Random object with the current time as its seed.
+Random(seed: long)	Constructs a Random object with a specified seed.
+nextInt(): int	Returns a random int value.
+nextInt(n: int): int	Returns a random int value between 0 and n (exclusive).
+nextLong(): long	Returns a random long value.
+nextDouble(): double	Returns a random double value between 0.0 and 1.0 (exclusive).
+nextFloat(): float	Returns a random float value between 0.0F and 1.0F (exclusive).
+nextBoolean(): boolean	Returns a random boolean value.

```
Random random1 = new Random(3);
for (int i = 0; i < 10; i++)
  System.out.print(random1.nextInt(1000) + " ");
```

```
734 660 210 581 128 202 549 564 459 961
```

Static Variables, Constants and Methods



UML Notation:
+: public variables or methods
underline: static variables or methods

Visibility Modifiers and Accessor/Mutator Methods

- By default, the class, variable, or method can be accessed by any class in the same package.

`public`

The class, data, or method is visible to any class in any package.

`private`

The data or methods can be accessed only by the declaring class - To protect data!

The get and set methods are used to read and modify private properties.

Packages

- The private modifier restricts access to within a class
- The default modifier restricts access to within a package
- public – unrestricted access

package p1;

```
public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

```
public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;

    can invoke o.m1();
    can invoke o.m2();
    cannot invoke o.m3();
  }
}
```

package p2;

```
public class C3 {
  void aMethod() {
    p1.C1 o = new p1.C1();
    can access o.x;
    cannot access o.y;
    cannot access o.z;

    can invoke o.m1();
    cannot invoke o.m2();
    cannot invoke o.m3();
  }
}
```

package p1;

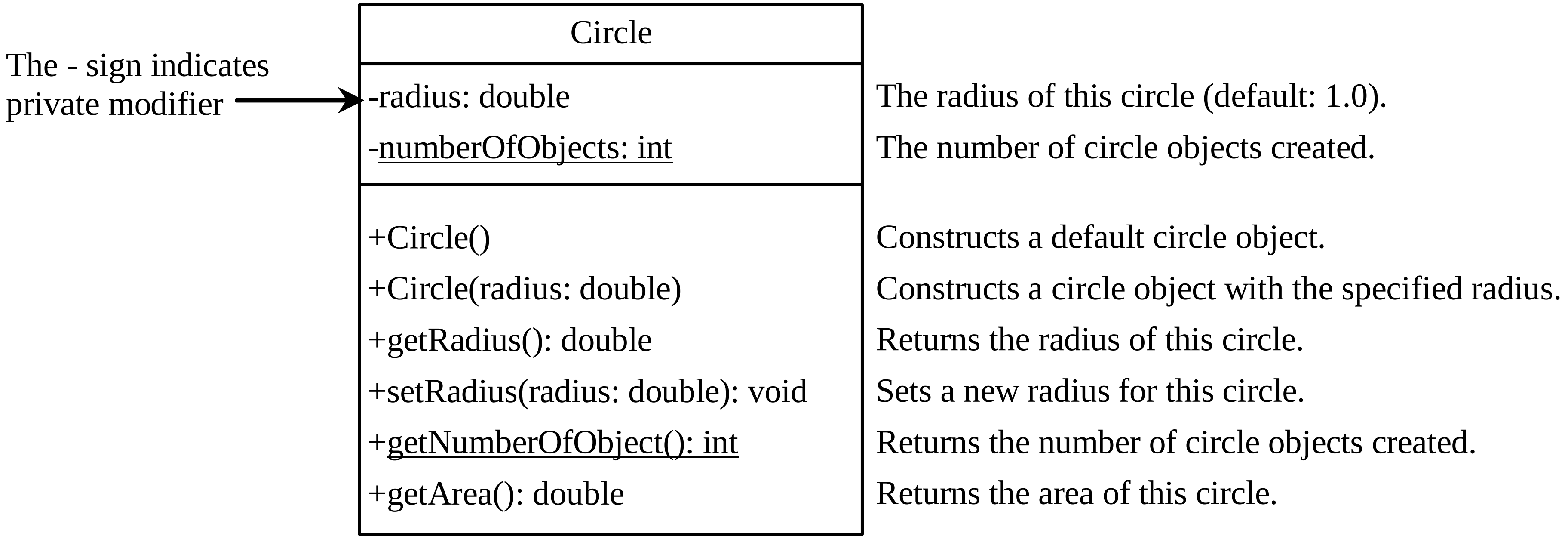
```
class C1 {
  ...
}
```

```
public class C2 {
  can access C1
}
```

package p2;

```
public class C3 {
  cannot access C1;
  can access C2;
}
```

UML: Data Field Encapsulation



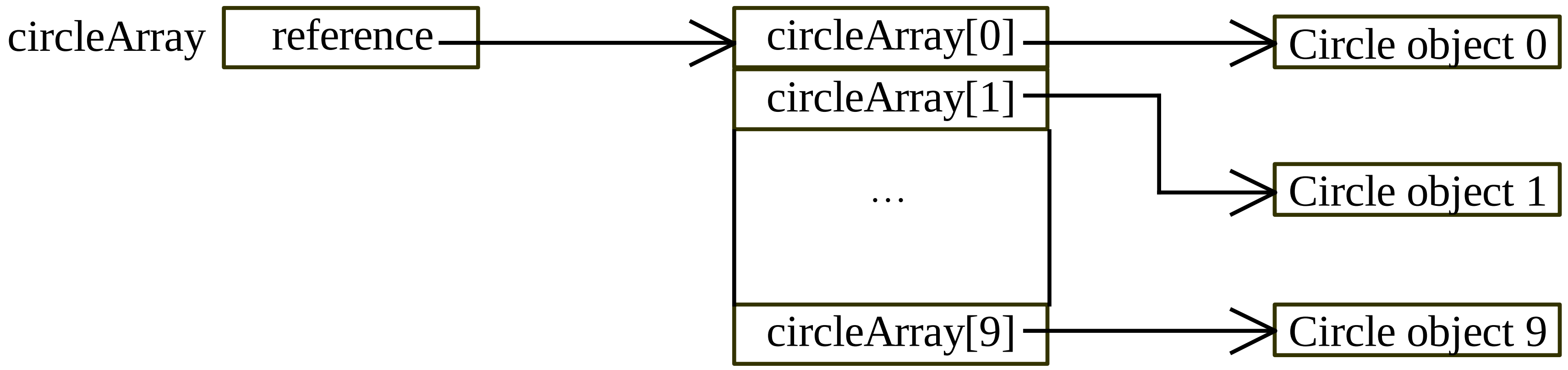
The - sign indicates private modifier

Circle	
-radius: double	The radius of this circle (default: 1.0).
-numberOfObjects: int	The number of circle objects created.
+Circle()	Constructs a default circle object.
+Circle(radius: double)	Constructs a circle object with the specified radius.
+getRadius(): double	Returns the radius of this circle.
+setRadius(radius: double): void	Sets a new radius for this circle.
+getNumberOfObject(): int	Returns the number of circle objects created.
+getArea(): double	Returns the area of this circle.

Array of Objects

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

- An array of objects is an *array of reference variables* (like the multi-dimensional arrays seen before)


circleArray
reference
circleArray[0]
circleArray[1]
…
circleArray[9]
Circle object 0
Circle object 1
Circle object 9