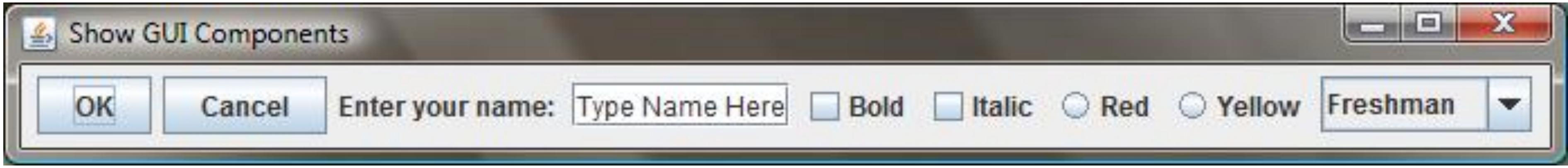
Chapter 9
Objects and Classes

Motivations

- After learning the preceding chapters, you are capable of solving many programming problems using selections, loops, methods, and arrays.
- However, these Java features are not sufficient for developing graphical user interfaces and large scale software systems.
- Suppose you want to develop a graphical user interface as shown below. How do you program it?

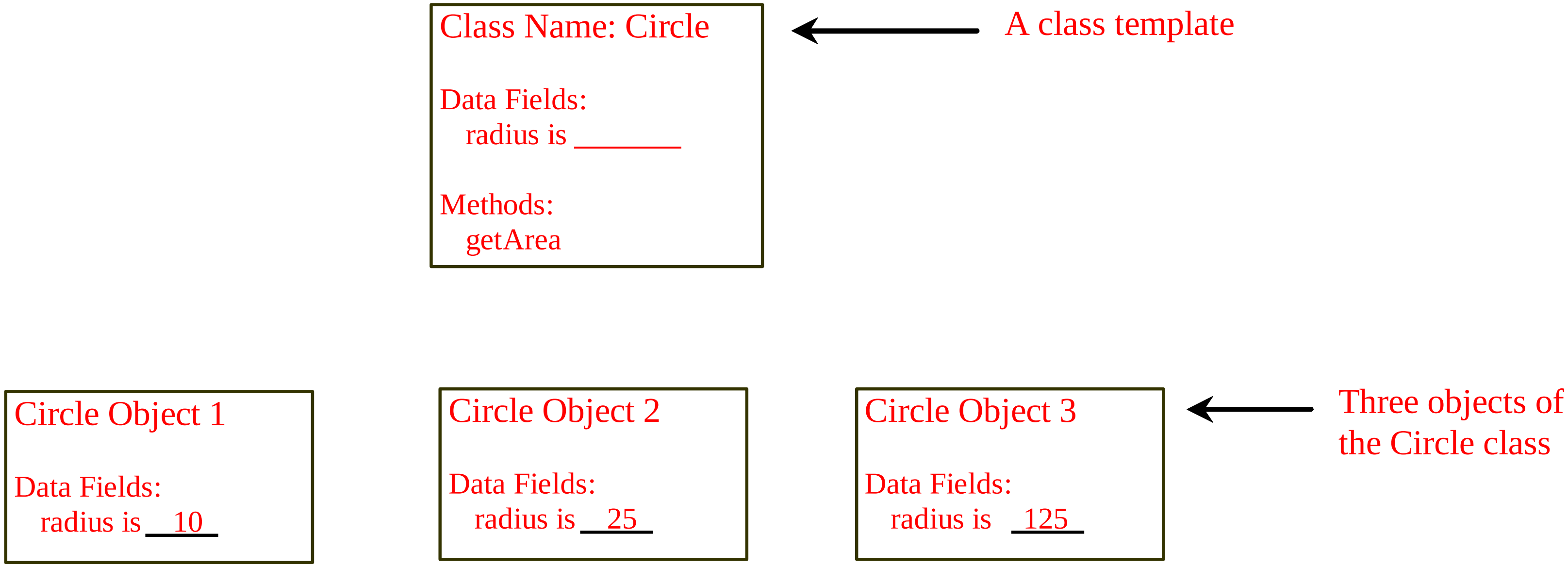


Show GUI Components

OO Programming Concepts

- Object-oriented programming (OOP) involves programming using objects.
- An *object* represents an entity in the real world that can be distinctly identified.
- For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
- An object has a unique identity, state, and behaviors.
- The *state* of an object consists of a set of *data fields* (also known as *properties*) with their current values.
- The *behavior* of an object is defined by a set of methods.

Objects



Class Name: Circle

Data Fields:
radius is _______

Methods:
getArea

A class template

Circle Object 1

Data Fields:
radius is 10

Circle Object 2

Data Fields:
radius is 25

Circle Object 3

Data Fields:
radius is 125

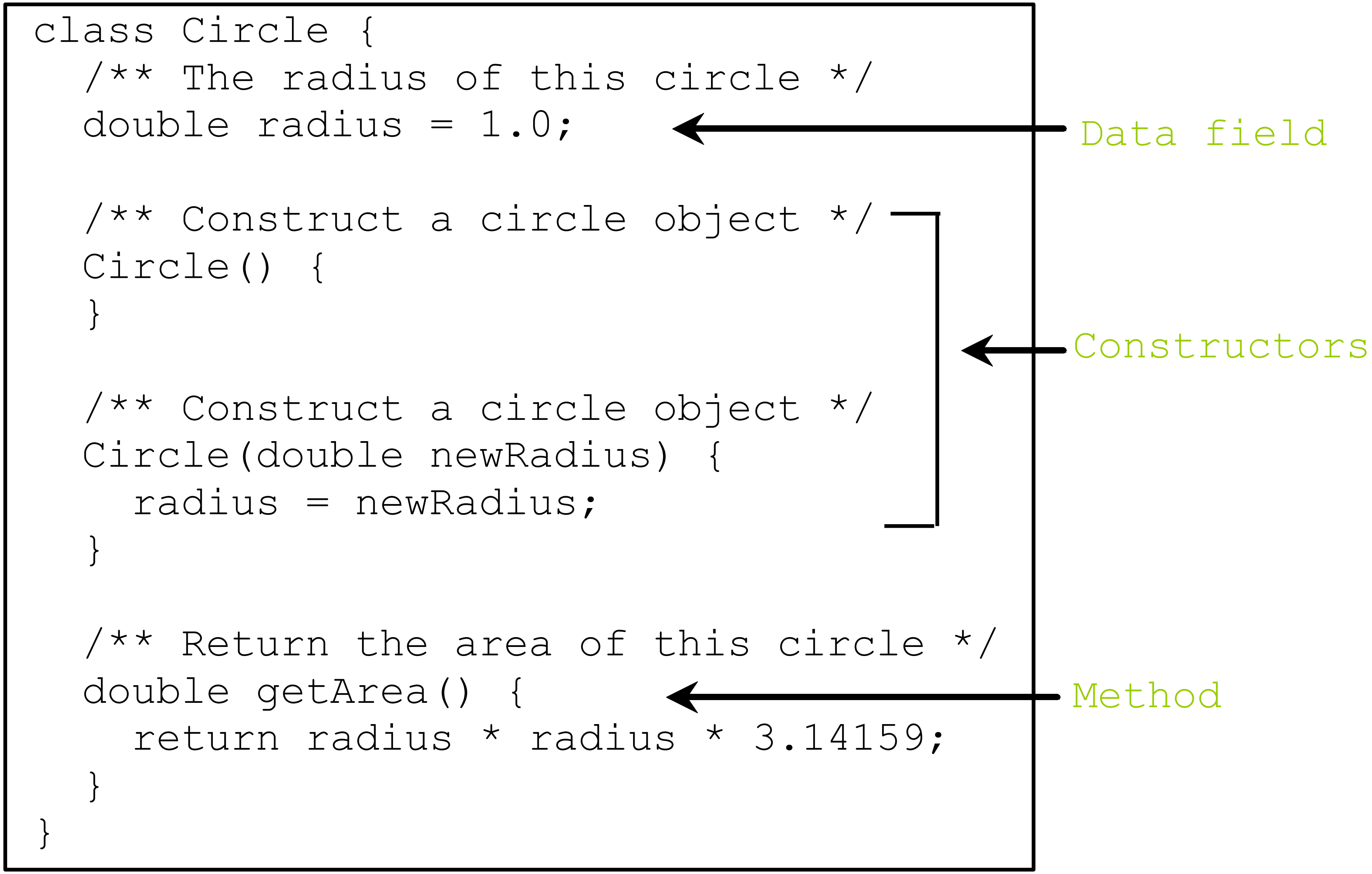
Three objects of
the Circle class

An object has both a state and behavior. The state defines the object, and the behavior defines what the object does.

Classes

- *Classes* are constructs that define objects of the same type.
- A Java class uses variables to define data fields and methods to define behaviors.
- Additionally, a class provides a special type of methods, known as constructors, which are invoked to construct objects from the class.

Classes



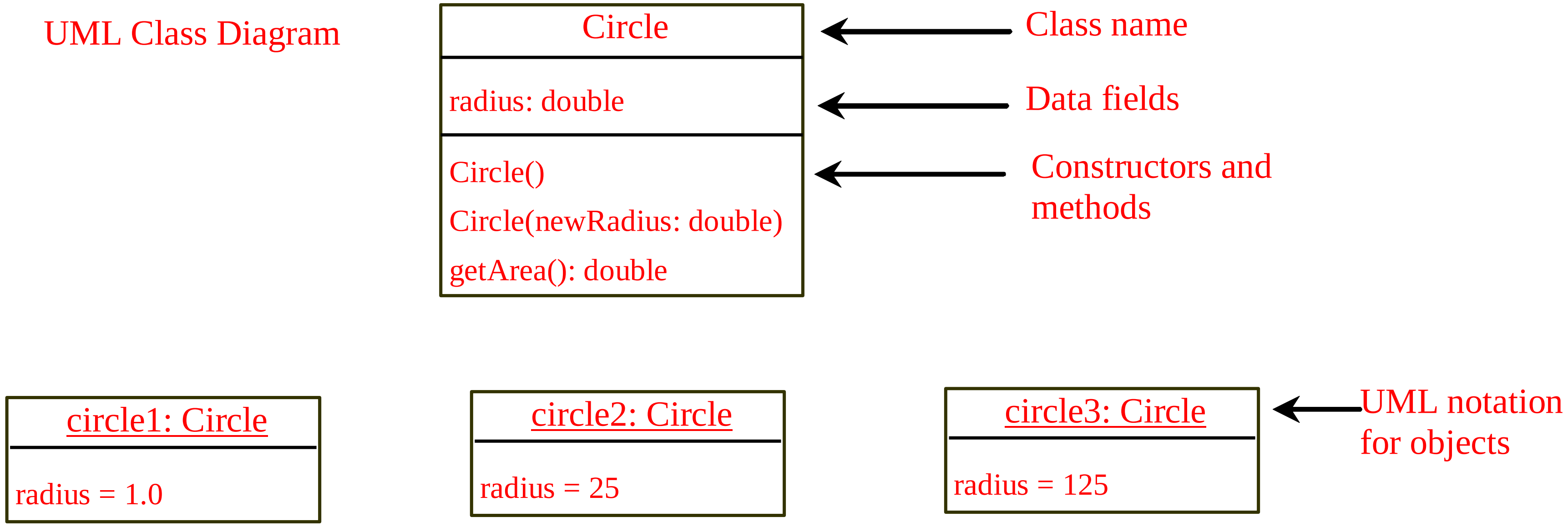
```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

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

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

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

UML Class Diagram



UML Class Diagram

Circle

radius: double

Circle()

Circle(newRadius: double)

getArea(): double

Class name

Data fields

Constructors and methods

circle1: Circle

radius = 1.0

circle2: Circle

radius = 25

circle3: Circle

radius = 125

UML notation for objects

Constructors

```
Circle() {
}

Circle(double newRadius) {
  radius = newRadius;
}
```

Constructors are a special kind of methods that are invoked to construct objects.

Constructors, cont.

- A constructor with no parameters is referred to as a *no-arg constructor*.
- 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. Constructors play the role of initializing objects.

Creating Objects Using Constructors

```
new ClassName();
```

Example:

```
new Circle();

new Circle(5.0);
```

Default Constructor

- A class may be declared without constructors.
- In this case, a no-arg constructor with an empty body is implicitly declared in the class.
- This constructor, called *a default constructor*, is provided automatically *only if no constructors are explicitly declared in the class*.

Declaring Object Reference Variables

- To reference an object, assign the object to a reference variable.

- To declare a reference variable, use the syntax:

```
ClassName objectRefVar;
```

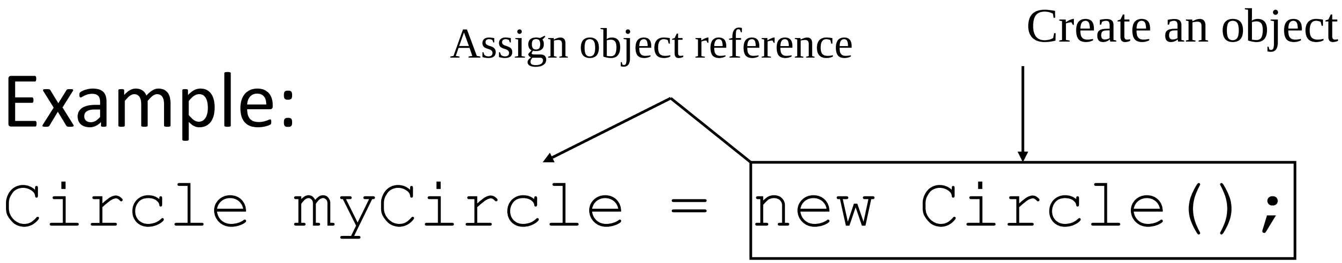
Example:

```
Circle myCircle;
```

Declaring/Creating Objects in a Single Step

```
ClassName objectRefVar = new ClassName();
```

Example:



Assign object reference

Create an object

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

Accessing Objects

- Referencing the object’s data:

```
objectRefVar.data
```

e.g., `myCircle.radius`

- Invoking the object’s method:

```
objectRefVar.methodName(arguments)
```

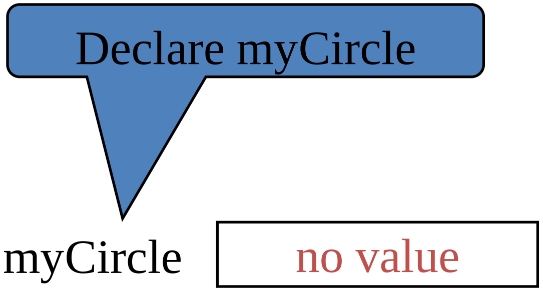
e.g., `myCircle.getArea()`

Trace Code

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

SCircle yourCircle = new Circle();

yourCircle.radius = 100;
```

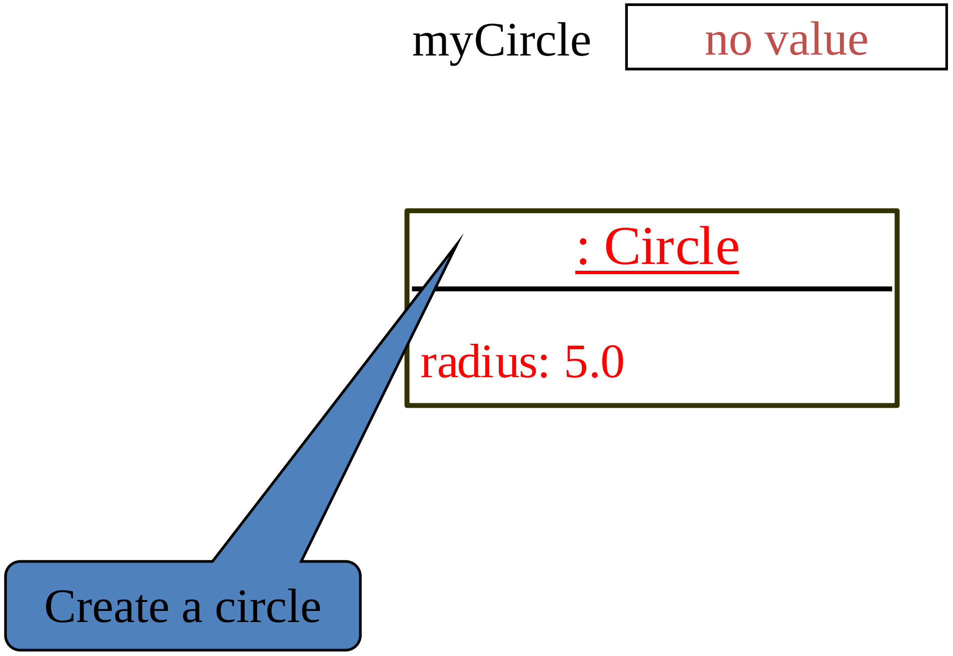


Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



myCircle

no value

: Circle

radius: 5.0

Create a circle

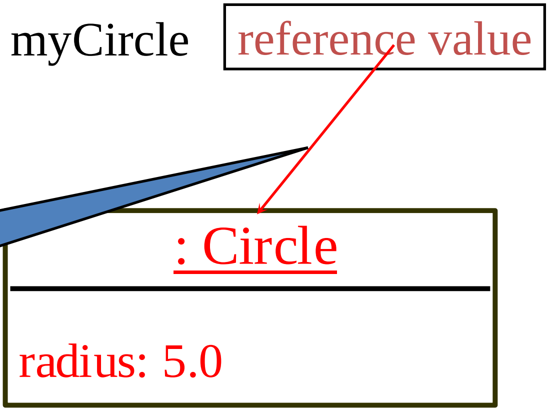
Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

Assign object reference to myCircle

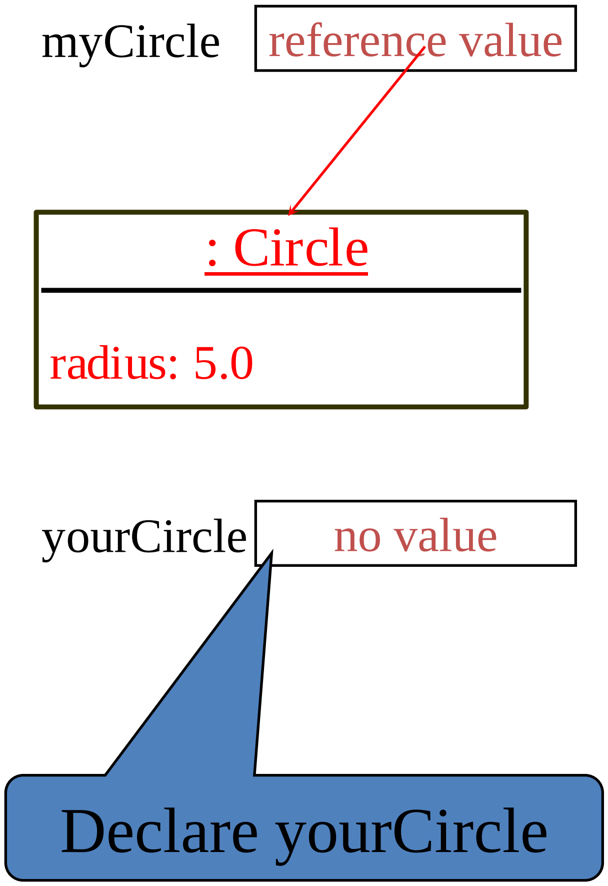


Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



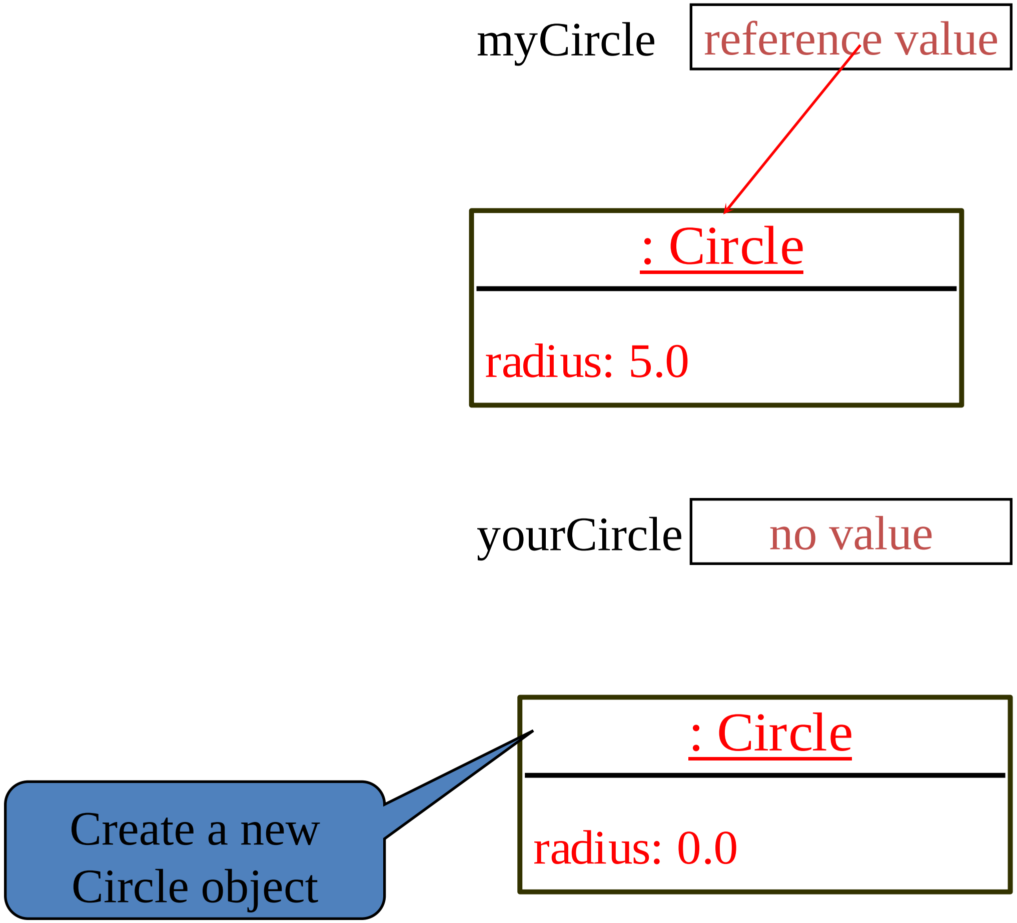
Declare yourCircle

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



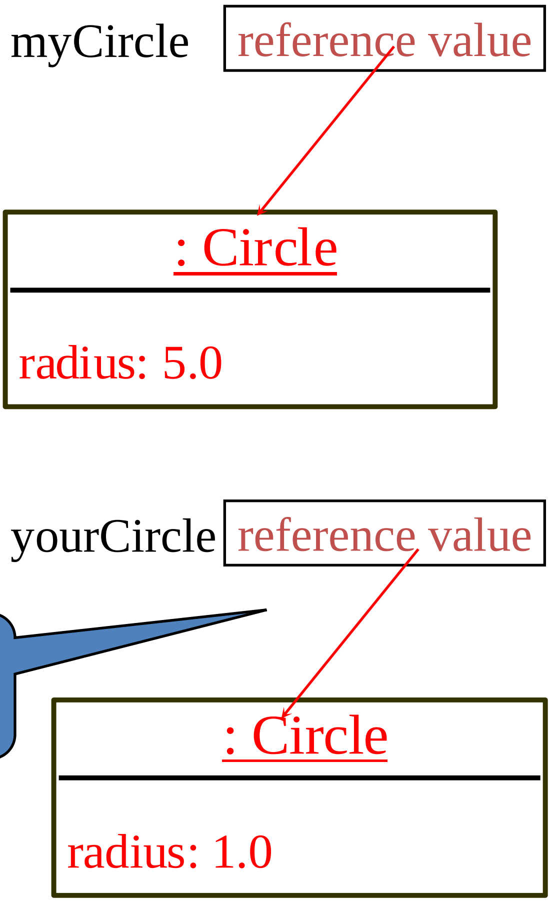
Create a new Circle object

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



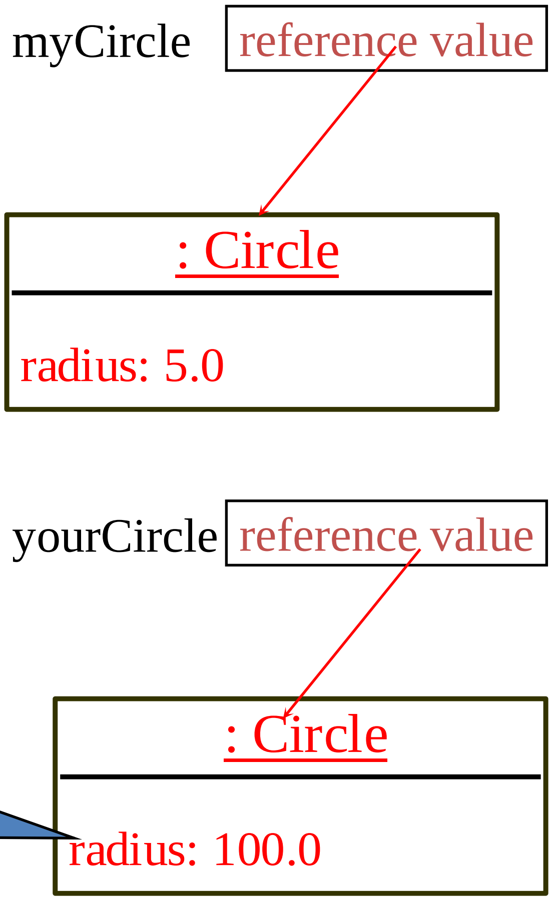
Assign object reference to yourCircle

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Change radius in yourCircle

Caution

Recall that you use

Math.methodName(arguments) (e.g., Math.pow(3, 2.5))

to invoke a method in the Math class. Can you invoke getArea() using Circle1.getArea()? The answer is no. All the methods used before this chapter are static methods, which are defined using the static keyword. However, getArea() is non-static. It must be invoked from an object using

objectRefVar.methodName(arguments) (e.g., myCircle.getArea()).

More explanations will be given in the section on “Static Variables, Constants, and Methods.”

Reference Data Fields

The data fields can be of reference types. For example, the following Student class contains a data field name of the String type.

```
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'
}
```

The null Value

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

Default Value for a Data Field

- The default value of a data field is
  - null for a reference type,
  - 0 for a numeric type,
  - false for a boolean type, and
  - '\u0000' for a char type.
- However, Java assigns no default value to a local variable inside a method.

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

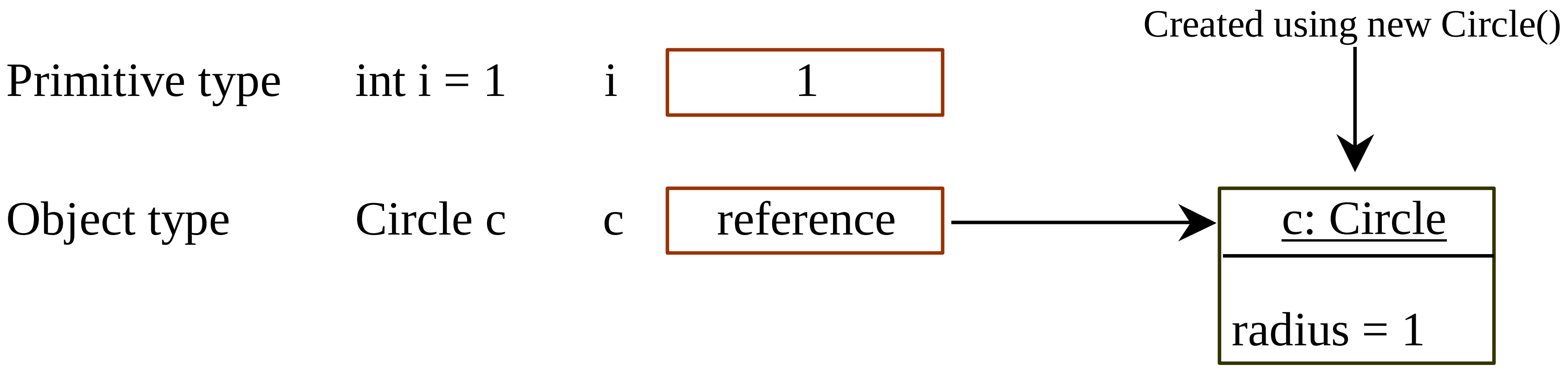
Example

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: variables not initialized

Differences between Variables of Primitive Data Types and Object Types



Primitive type int i = 1 i 1

Object type Circle c c reference

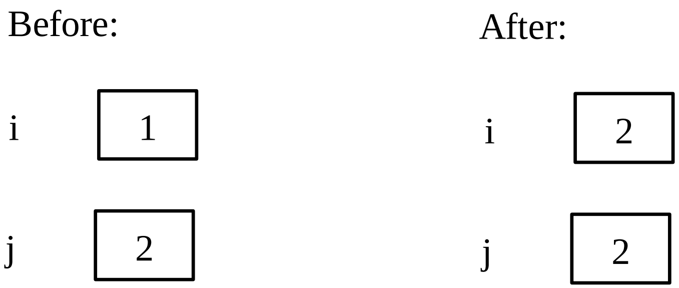
Created using new Circle()

c: Circle

radius = 1

Copying Variables of Primitive Data Types and Object Types

Primitive type assignment i = j


Before:

i 1

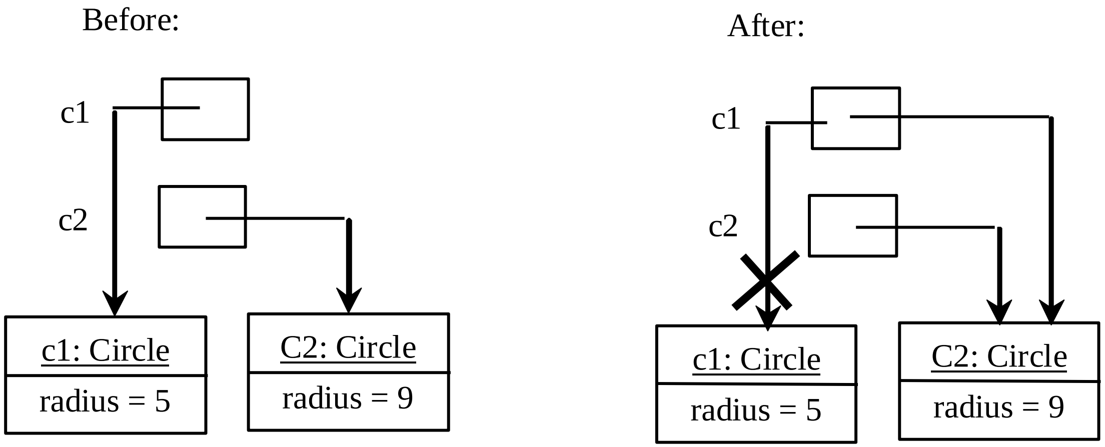
j 2

After:

i 2

j 2

Object type assignment c1 = c2


Before:

c1

c2

c1: Circle
radius = 5

C2: Circle
radius = 9

After:

c1

c2

c1: Circle
radius = 5

C2: Circle
radius = 9

Garbage Collection

- As shown in the previous figure, after the assignment statement c1 = c2, c1 points to the same object referenced by c2.
- The object previously referenced by c1 is no longer referenced. This object is known as garbage.
- Garbage is automatically collected by JVM.

Garbage Collection, cont

- TIP: If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object.
- The JVM will automatically collect the space if the object is not referenced by any variable.

Instance
Variables, and Methods

- Instance variables belong to a specific instance.
- Instance methods are invoked by an instance of the class.

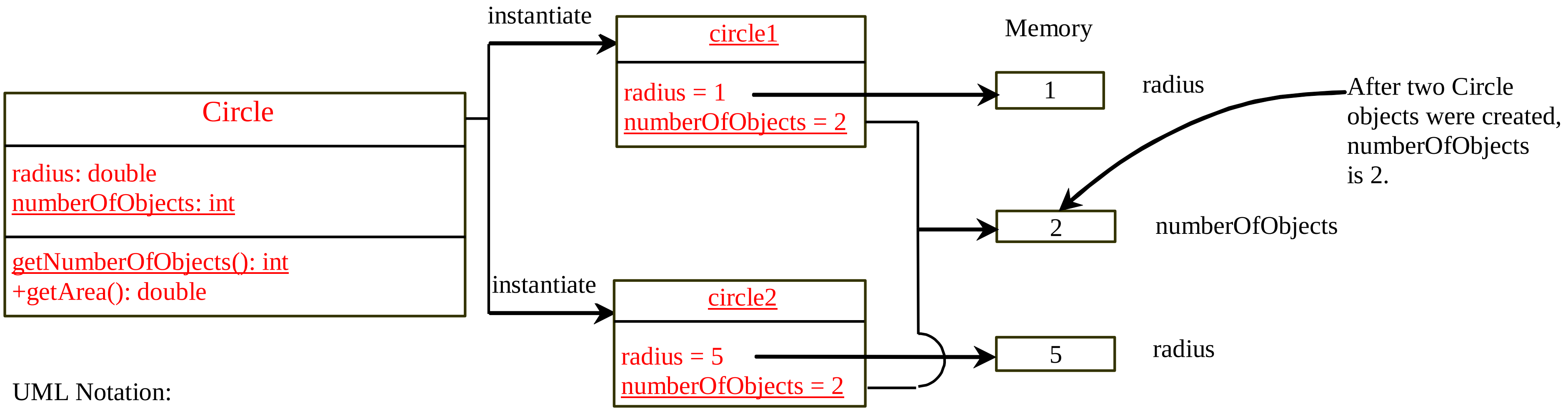
Static Variables, Constants, and Methods

- Static variables are shared by all the instances of the class.
- Static methods are not tied to a specific object.
- Static constants are final variables shared by all the instances of the class.

Static Variables, Constants, and Methods, cont.

To declare static variables, constants, and methods, use the *static* modifier.

Static Variables, Constants, and Methods, cont.



After two Circle objects were created, numberOfObjects is 2.

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.

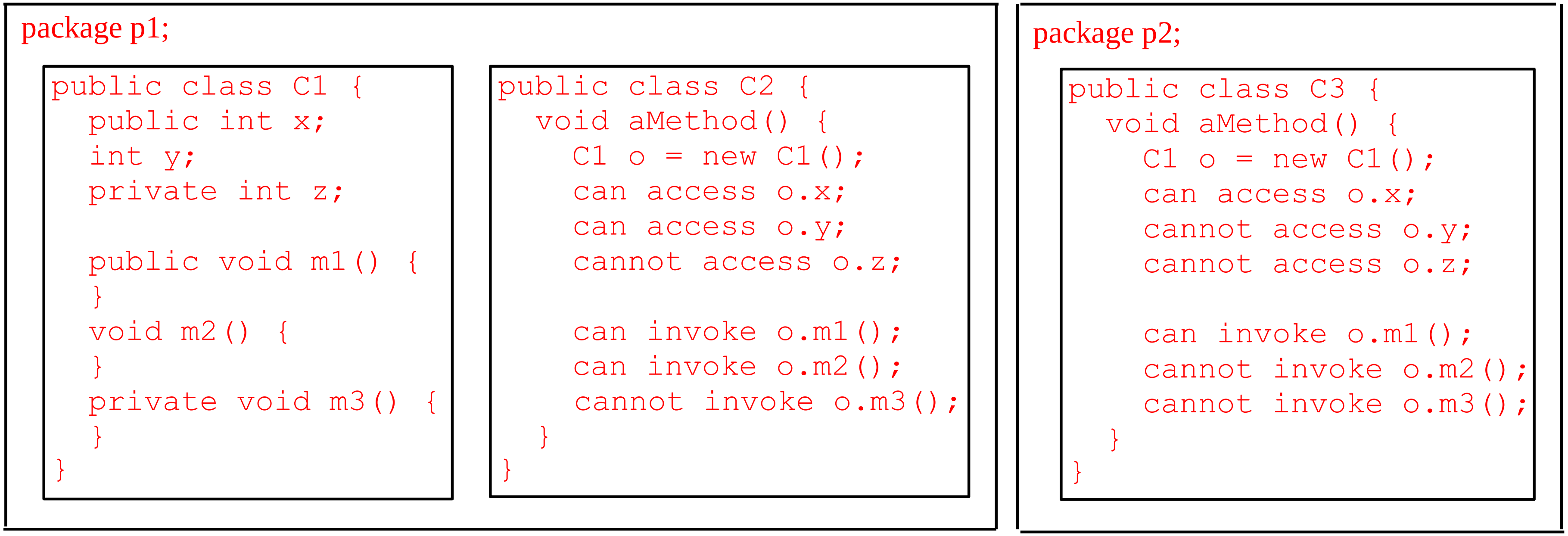
- `public`

  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.

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



package p1;

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

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

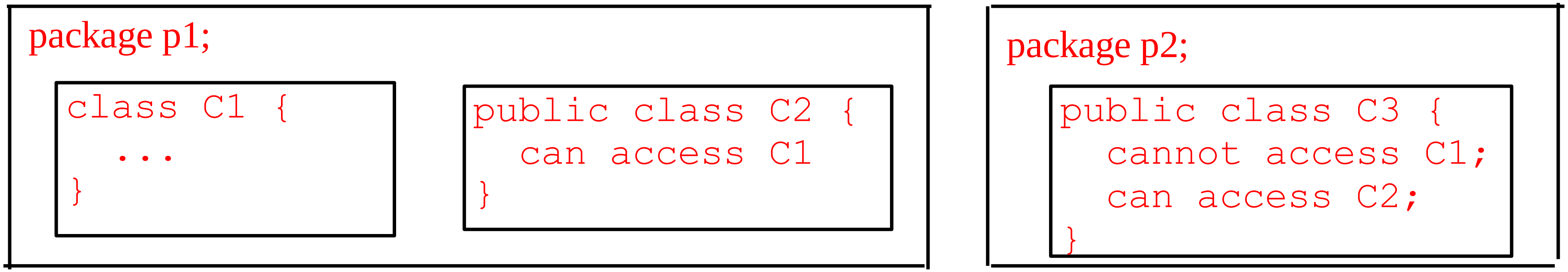
```
public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;

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

package p2;

```
public class C3 {
  void aMethod() {
    C1 o = new 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;
}
```

The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.

NOTE

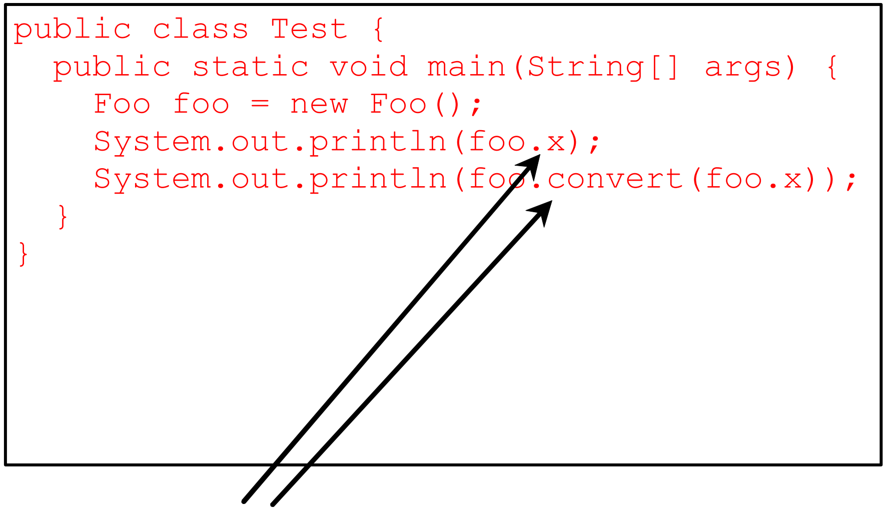
An object cannot access its private members, as shown in (b). It is OK, however, if the object is declared in its own class, as shown in (a).

```
public class Foo {
  private boolean x;

  public static void main(String[] args) {
    Foo foo = new Foo();
    System.out.println(foo.x);
    System.out.println(foo.convert());
  }

  private int convert(boolean b) {
    return x ? 1 : -1;
  }
}
```

(a) This is OK because object foo is used inside the Foo class



```
public class Test {
  public static void main(String[] args) {
    Foo foo = new Foo();
    System.out.println(foo.x);
    System.out.println(foo.convert(foo.x));
  }
}
```

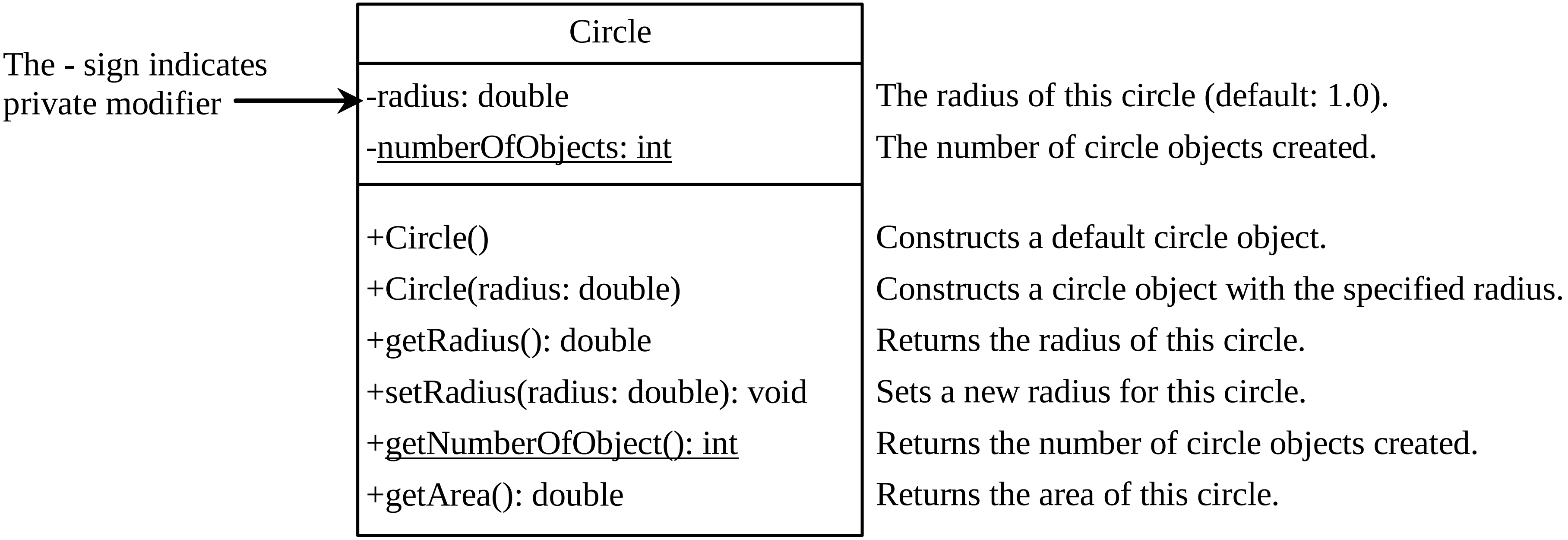
(b) This is wrong because x and convert are private in Foo.

Why Data Fields Should Be private?

To protect data.

To make class easy to maintain.

Example of Data Field Encapsulation



Circle

The - sign indicates private modifier

-radius: double

-numberOfObjects: int

+Circle()

+Circle(radius: double)

+getRadius(): double

+setRadius(radius: double): void

+getNumberOfObject(): int

+getArea(): double

The radius of this circle (default: 1.0).

The number of circle objects created.

Constructs a default circle object.

Constructs a circle object with the specified radius.

Returns the radius of this circle.

Sets a new radius for this circle.

Returns the number of circle objects created.

Returns the area of this circle.

Passing Objects to Methods

- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)

Array of Objects

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

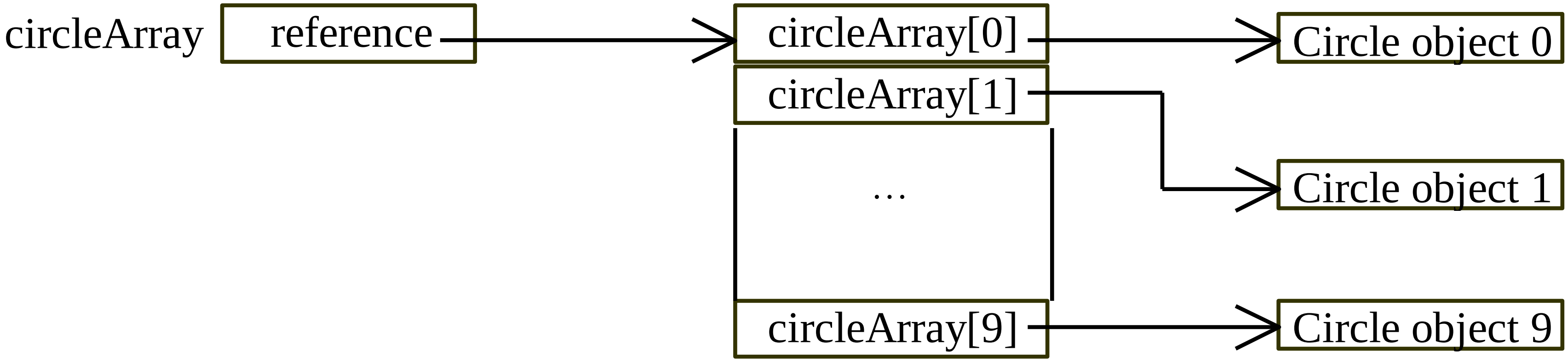
An array of objects is actually an *array of reference variables*.

So invoking circleArray[1].getArea() involves two levels of referencing as shown in the next figure.

circleArray references to the entire array. circleArray[1] references to a Circle object.

Array of Objects, cont.

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


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