Lecture 2: Objects and Classes (Ch 8)

Adapted by Fangzhen Lin for COMP3021 from Y. Danial Liang's PowerPoints for Introduction to Java Programming, Comprehensive Version, 9/E, Pearson, 2013.

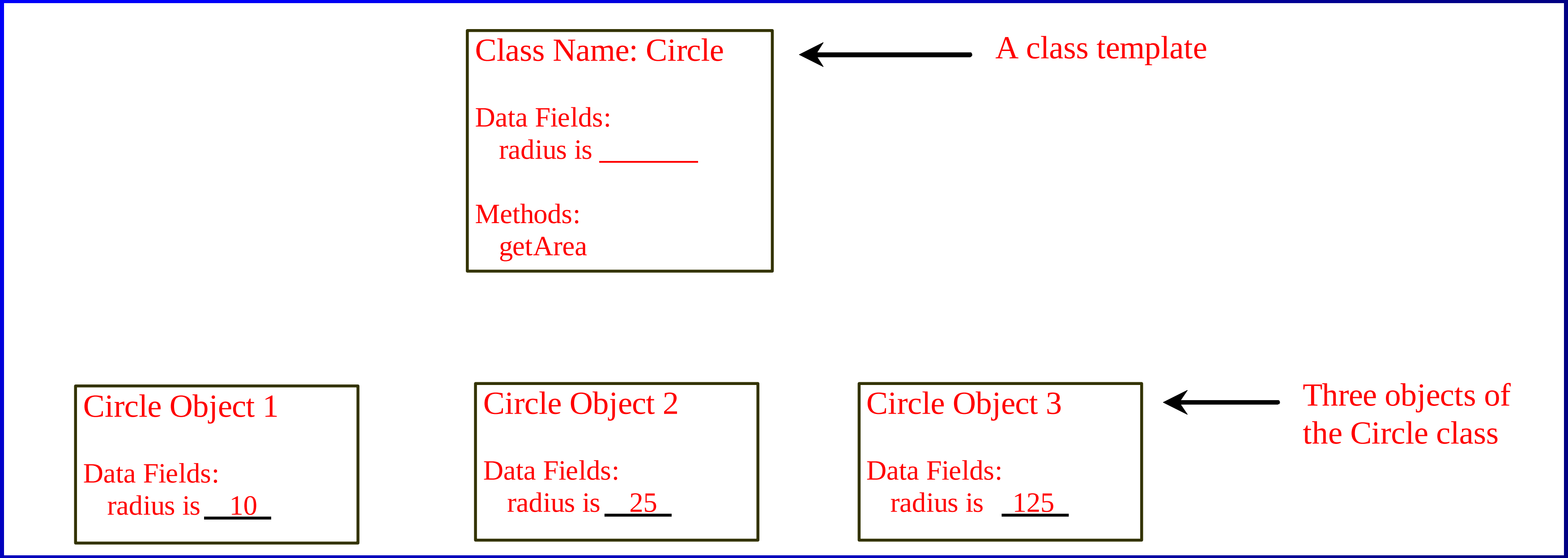
Objectives

- To describe objects and classes, and use classes to model objects (§8.2).
- To use UML graphical notation to describe classes and objects (§8.2).
- To demonstrate how to define classes and create objects (§8.3).
- To create objects using constructors (§8.4).
- To access objects via object reference variables (§8.5).
- To define a reference variable using a reference type (§8.5.1).
- To access an object's data and methods using the object member access operator (**.**) (§8.5.2).
- To define data fields of reference types and assign default values for an object's data fields (§8.5.3).
- To distinguish between object reference variables and primitive data type variables (§8.5.4).
- To use the Java library classes **Date**, **Random**, and **JFrame** (§8.6).
- To distinguish between instance and static variables and methods (§8.7).
- To define private data fields with appropriate **get** and **set** methods (§8.8).
- To encapsulate data fields to make classes easy to maintain (§8.9).
- To develop methods with object arguments and differentiate between primitive-type arguments and object-type arguments (§8.10).
- To store and process objects in arrays (§8.11).

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__

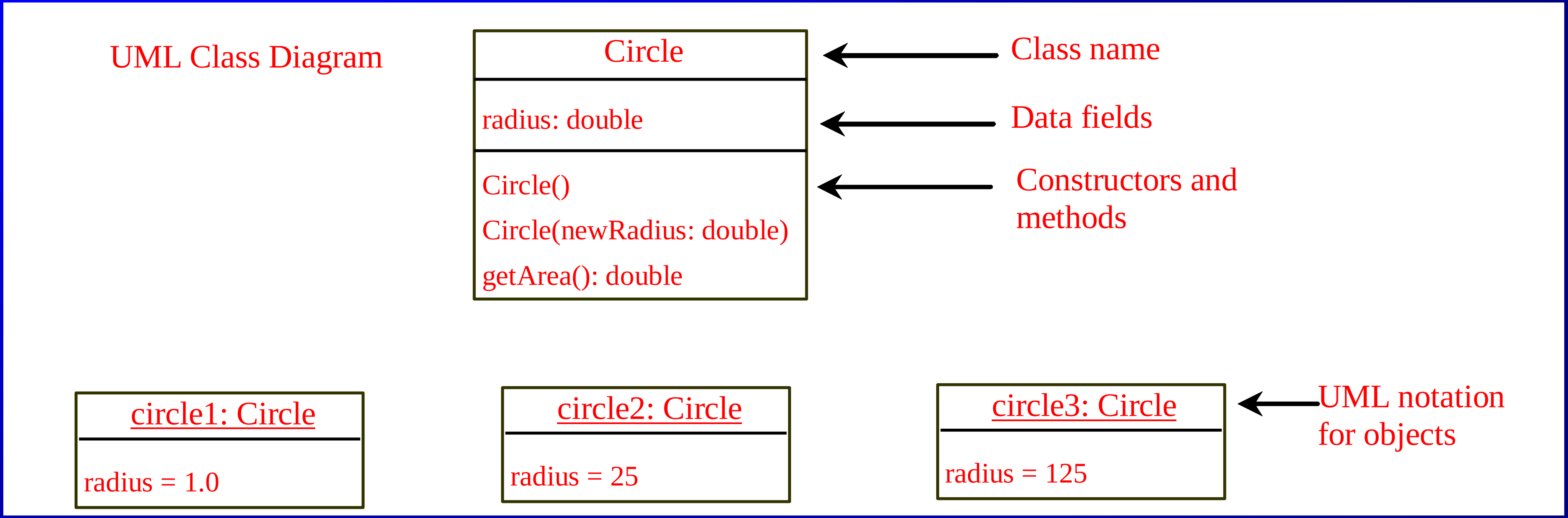
Three objects of the Circle class

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.

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

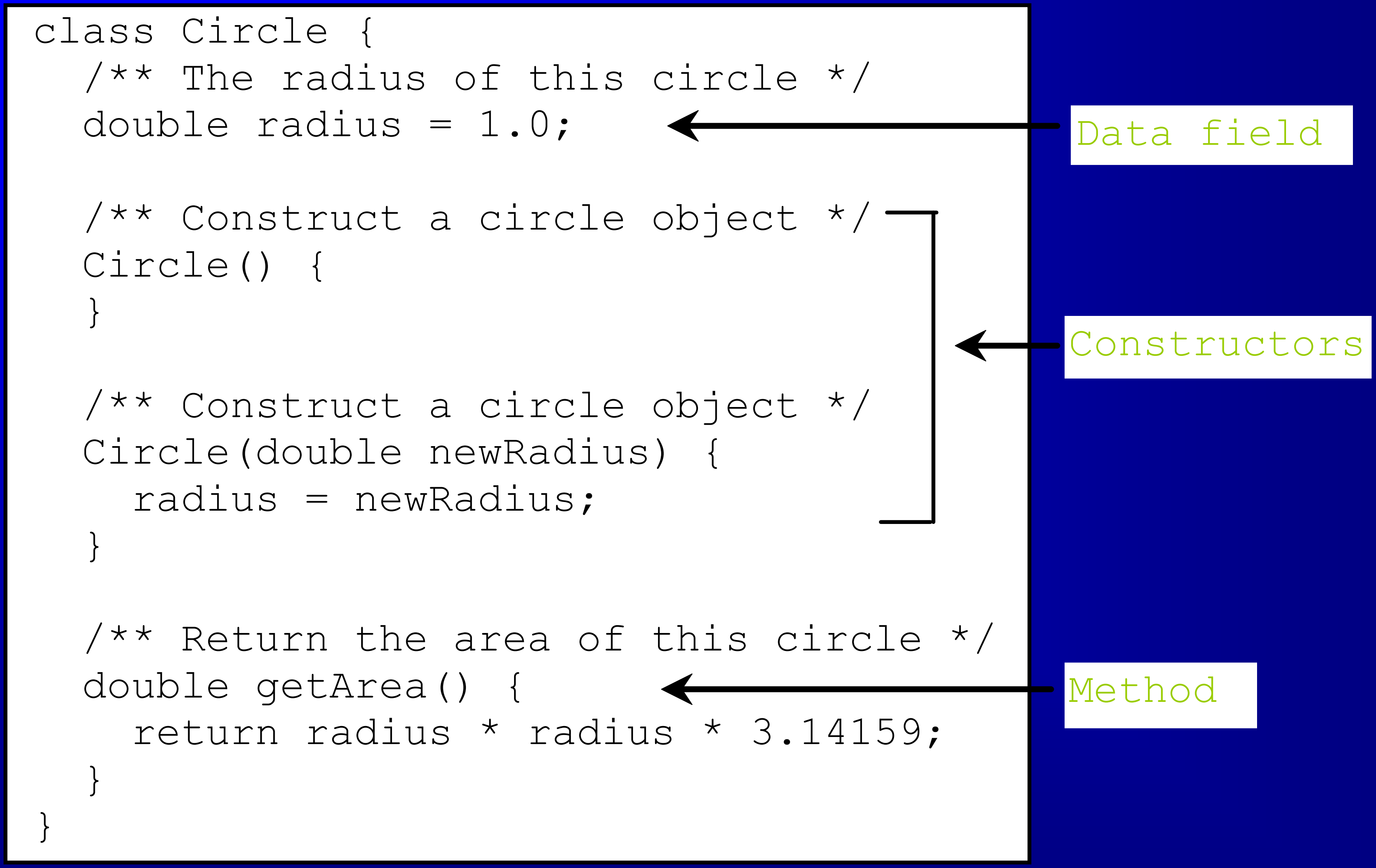
radius = 25

circle3: Circle

radius = 125

UML notation for objects

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;
  }
}
```

Data field

Constructors

Method

Constructors

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

```
Circle() {
}

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

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 defined 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 defined 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;
```

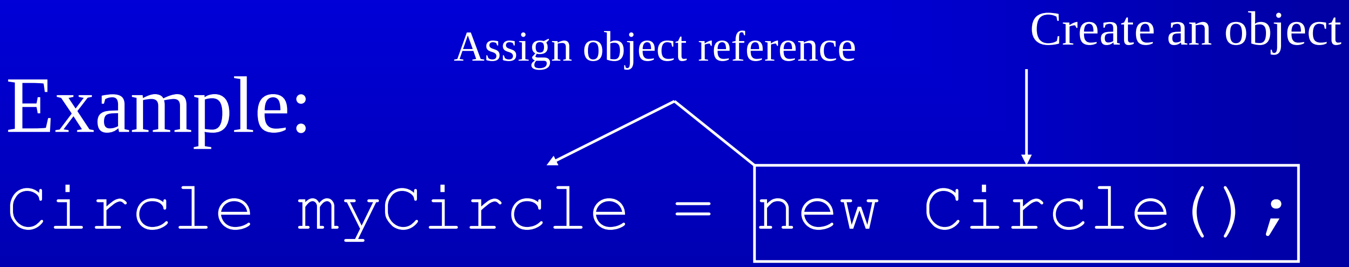
Example:

```
Circle myCircle;
```

Declaring/Creating Objects in a Single Step

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

Example:



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

Assign object reference

Create an object

Accessing Object’s Members

- ☞ Referencing the object’s data:

  `objectRefVar.data`

  e.g., `myCircle.radius`

- ☞ Invoking the object’s method:

  `objectRefVar.methodName(arguments)`

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

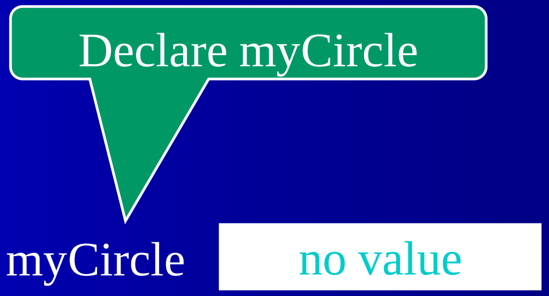
animation

Trace Code

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

SCircle yourCircle = new Circle();

yourCircle.radius = 100;
```



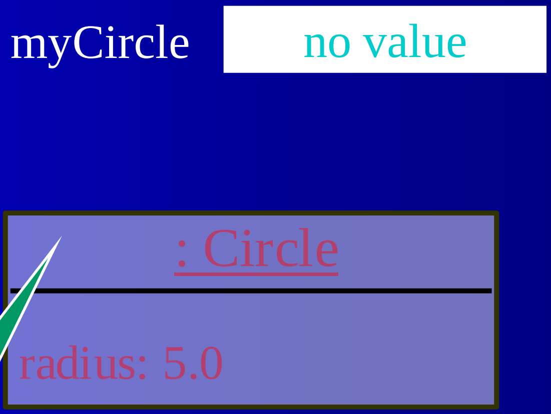
animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Create a circle

animation

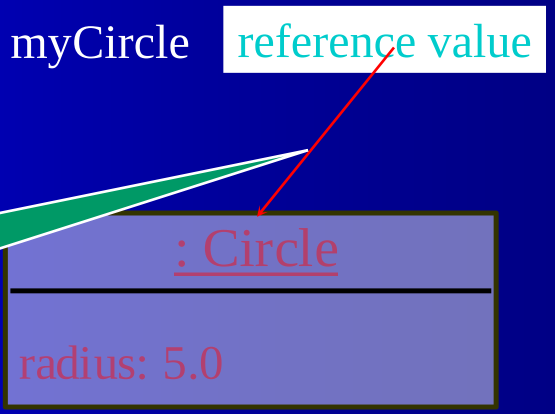
Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

Assign object reference to myCircle



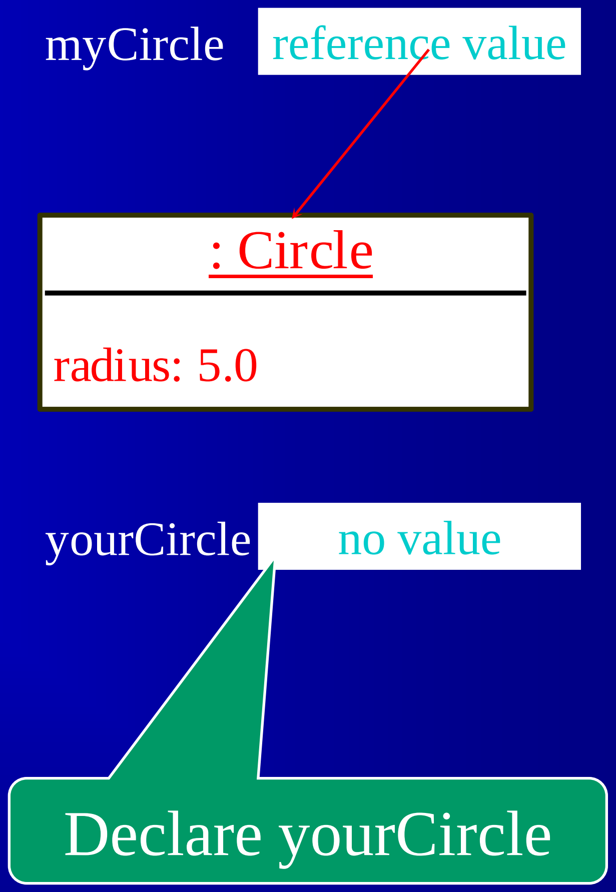
animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



myCircle reference value

: Circle

radius: 5.0

yourCircle no value

Declare yourCircle

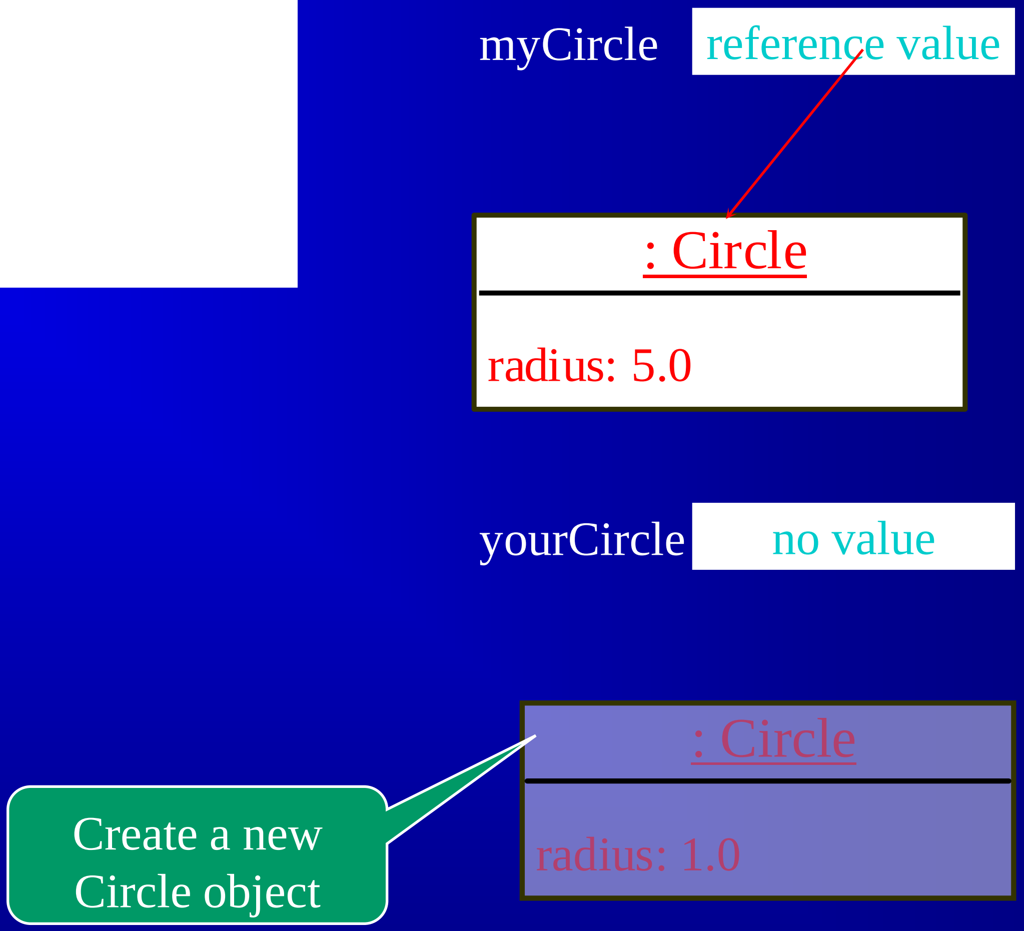
animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Create a new Circle object

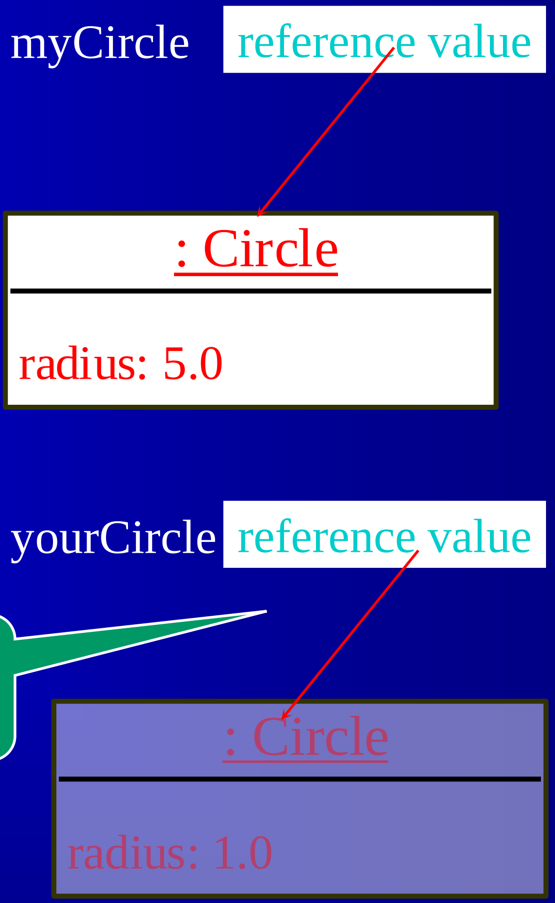
animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Assign object reference to yourCircle

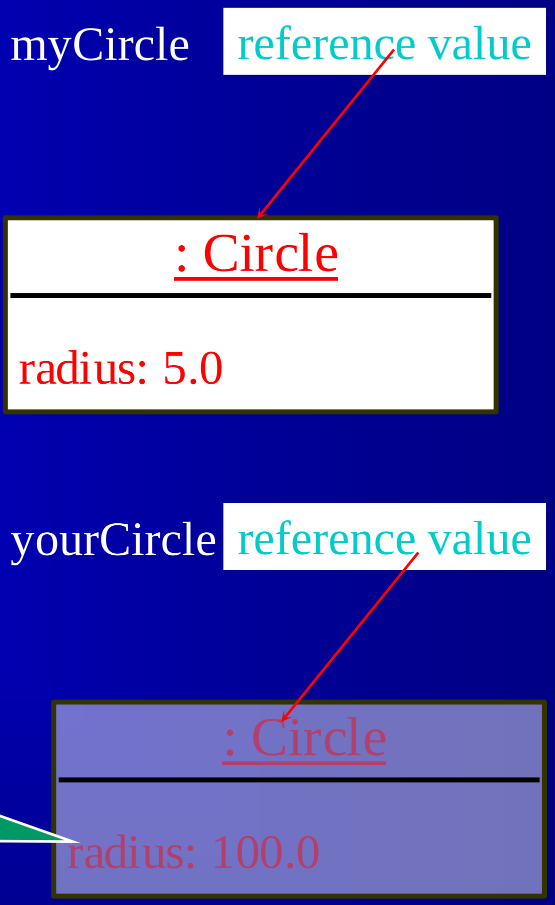
animation

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. Only static methods, which are defined using the static keyword can be called this way. 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);
  }
}
```

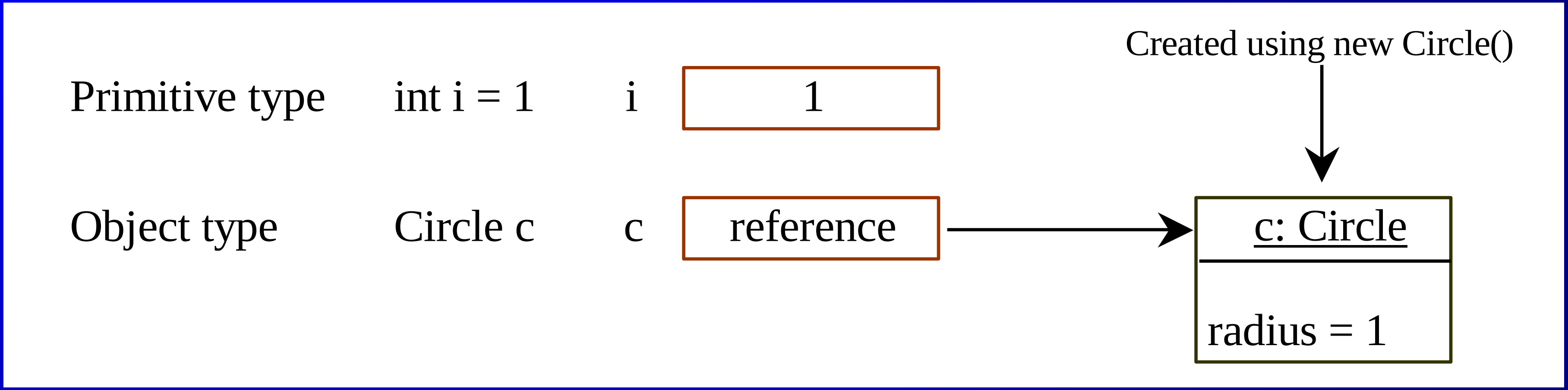
Example

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

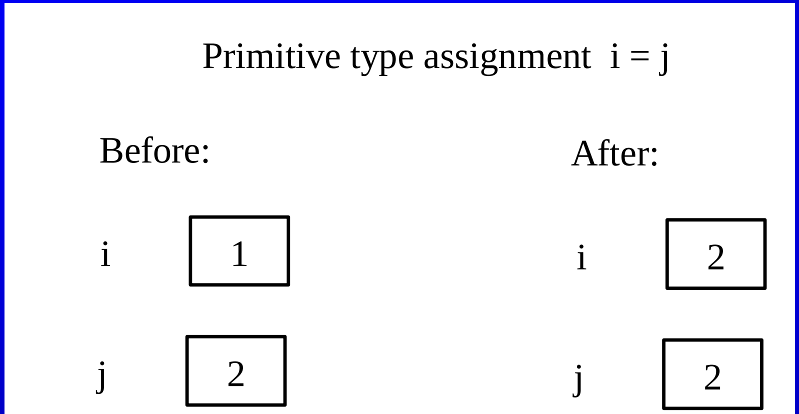
Object type Circle c c reference

Created using new Circle()

c: Circle

radius = 1

Copying Variables of Primitive Data Types and Object Types



Primitive type assignment i = j

Before:

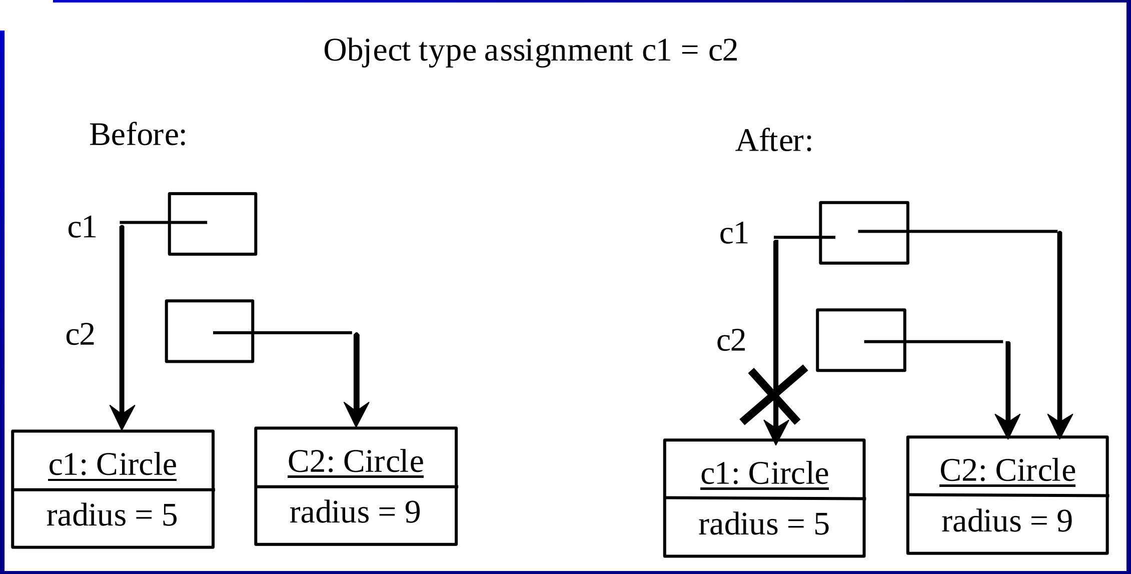
i 1

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

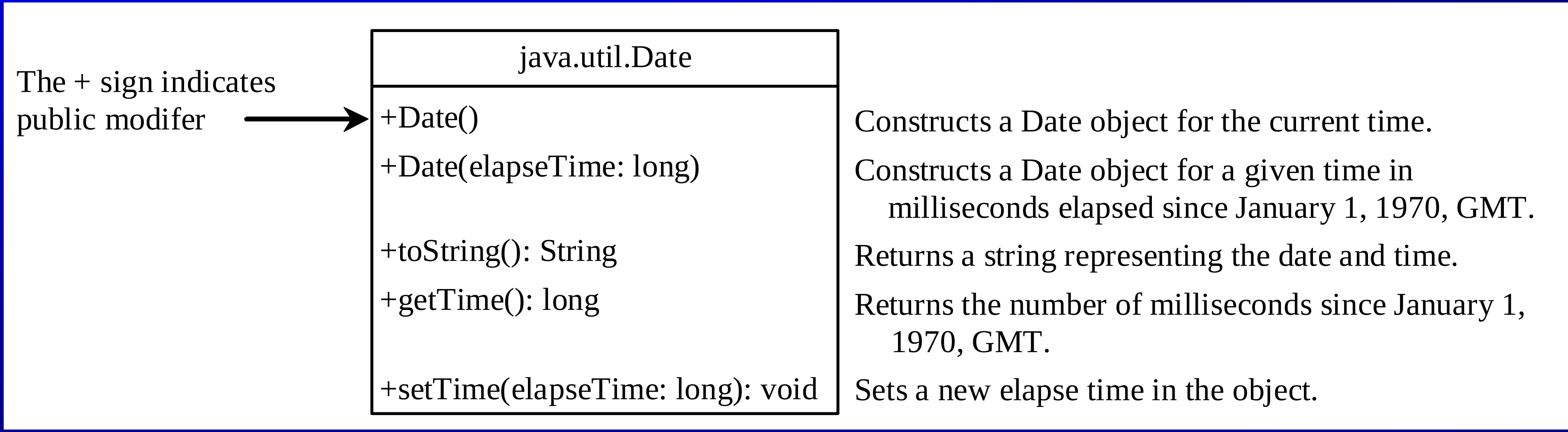
As shown in the previous figure, after the assignment statement c1 = c2, c1 points to the same object referenced by c2. The object previously referenced by c1 is no longer referenced. This object is known as garbage. Garbage is automatically collected by JVM.

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 .

The Date Class

Java provides a system-independent encapsulation of date and time in the java.util.Date class. You can use the Date class to create an instance for the current date and time and use its toString method to return the date and time as a string.



The + sign indicates public modifer

java.util.Date

+Date()

Constructs a Date object for the current time.

+Date(elapseTime: long)

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

+toString(): String

Returns a string representing the date and time.

+getTime(): long

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

+setTime(elapseTime: long): void

Sets a new elapse time in the object.

The Date Class Example

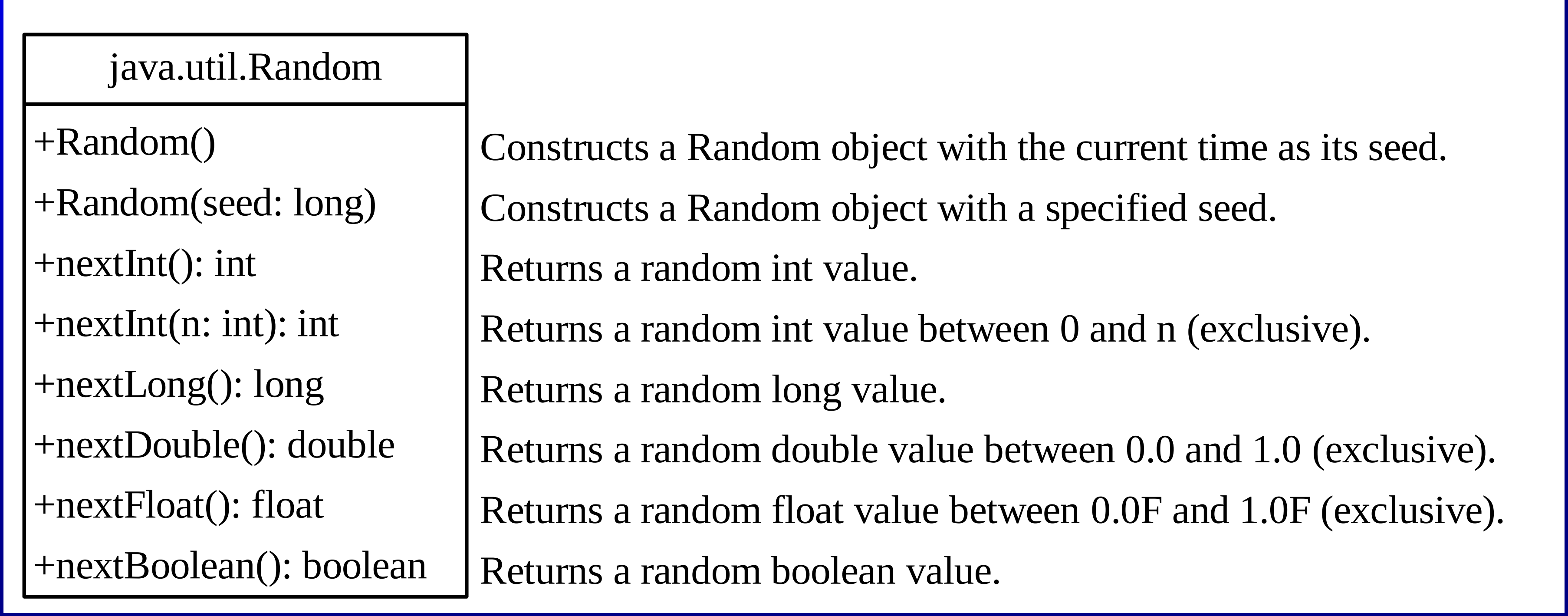
For example, the following code

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

displays a string like `Sun Mar 09 13:50:19 EST 2003`.

The Random Class

You have used Math.random() to obtain a random double value between 0.0 and 1.0 (excluding 1.0). A more useful random number generator is provided in the java.util.Random class.



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.

The Random Class Example

If two Random objects have the same seed, they will generate identical sequences of numbers. For example, the following code creates two Random objects with the same seed 3.

```
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
  System.out.print(random1.nextInt(1000) + " ");
Random random2 = new Random(3);
System.out.print("\nFrom random2: ");
for (int i = 0; i < 10; i++)
  System.out.print(random2.nextInt(1000) + " ");
```

```
From random1: 734 660 210 581 128 202 549 564 459 961
From random2: 734 660 210 581 128 202 549 564 459 961
```

Displaying GUI Components

When you develop programs to create graphical user interfaces, you will use Java classes such as JFrame, JButton, JRadioButton, JComboBox, and JList to create frames, buttons, radio buttons, combo boxes, lists, and so on. Here is an example that creates two windows using the JFrame class.

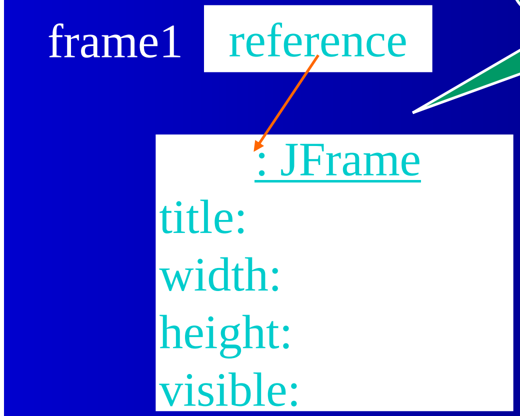
TestFrame

animation

Trace Code

```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true); JFrame
frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```

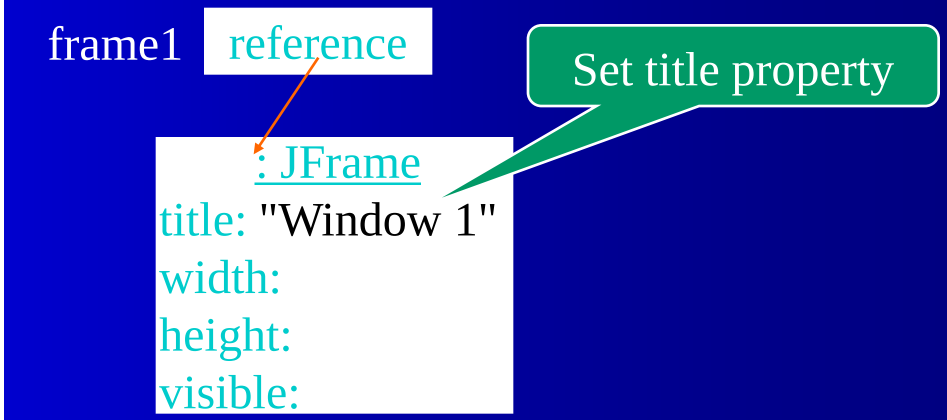
Declare, create, and assign in one statement



animation

Trace Code

```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true); JFrame
frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```


frame1 reference

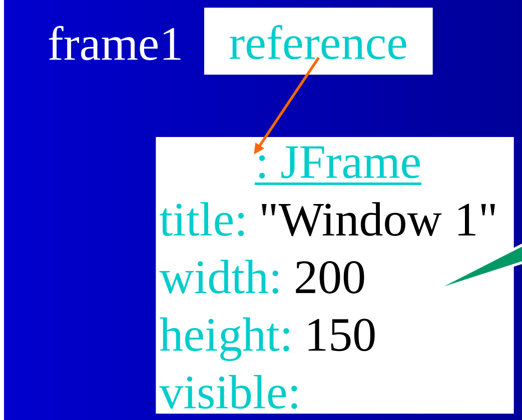
: JFrame
title: "Window 1"
width:
height:
visible:

Set title property

animation

Trace Code

```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```

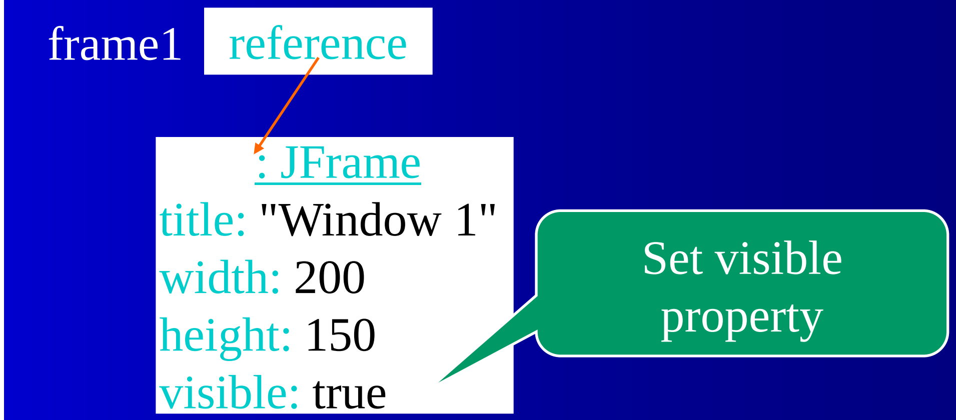


Set size property

animation

Trace Code

```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```



frame1 reference

: JFrame

title: "Window 1"

width: 200

height: 150

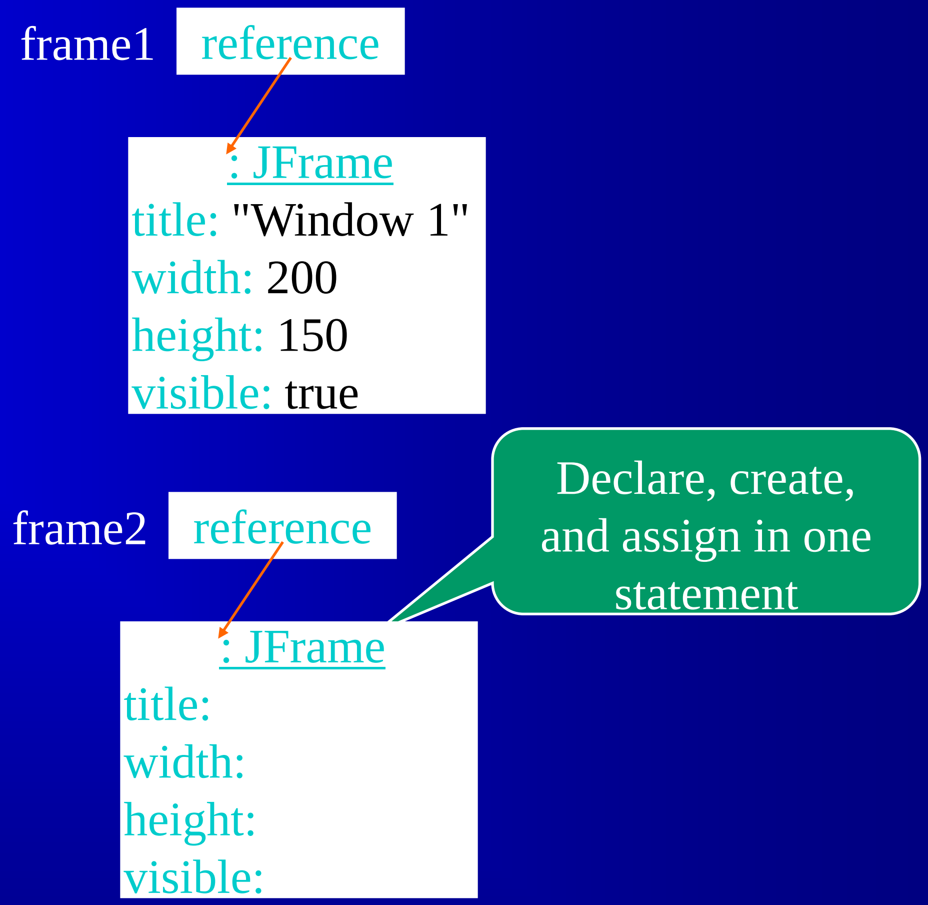
visible: true

Set visible property

animation

Trace Code

```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```

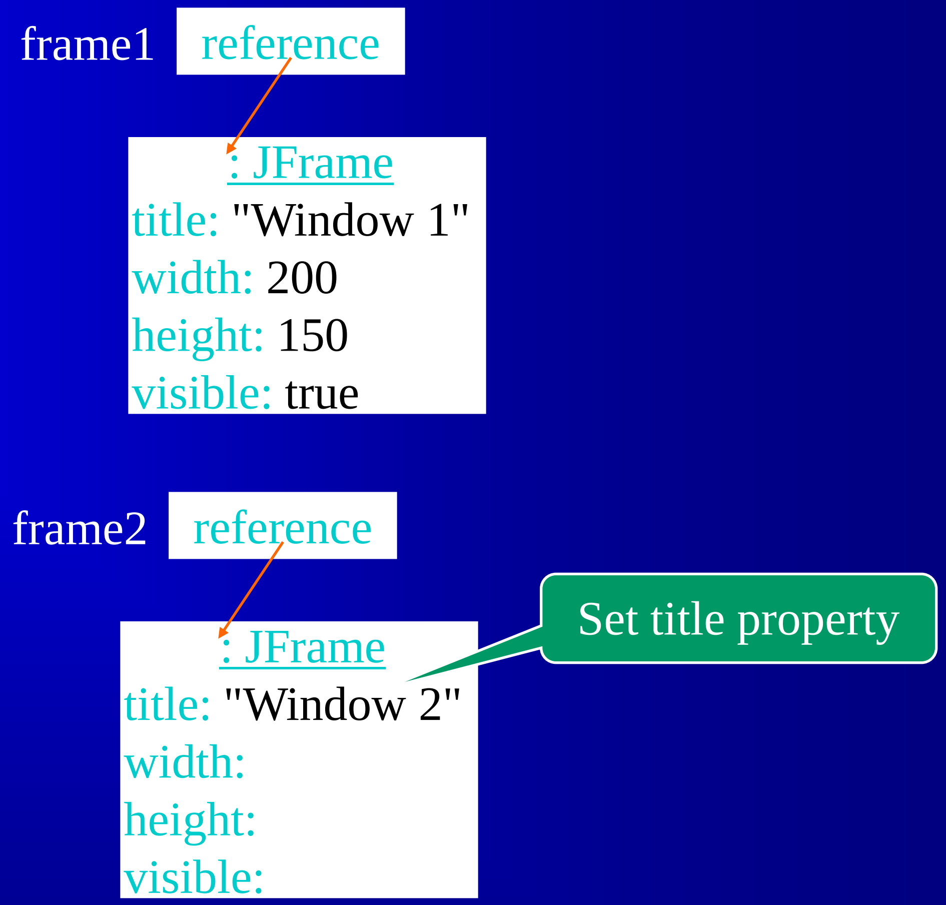


Declare, create, and assign in one statement

animation

Trace Code

```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```



frame1 reference

: JFrame
title: "Window 1"
width: 200
height: 150
visible: true

frame2 reference

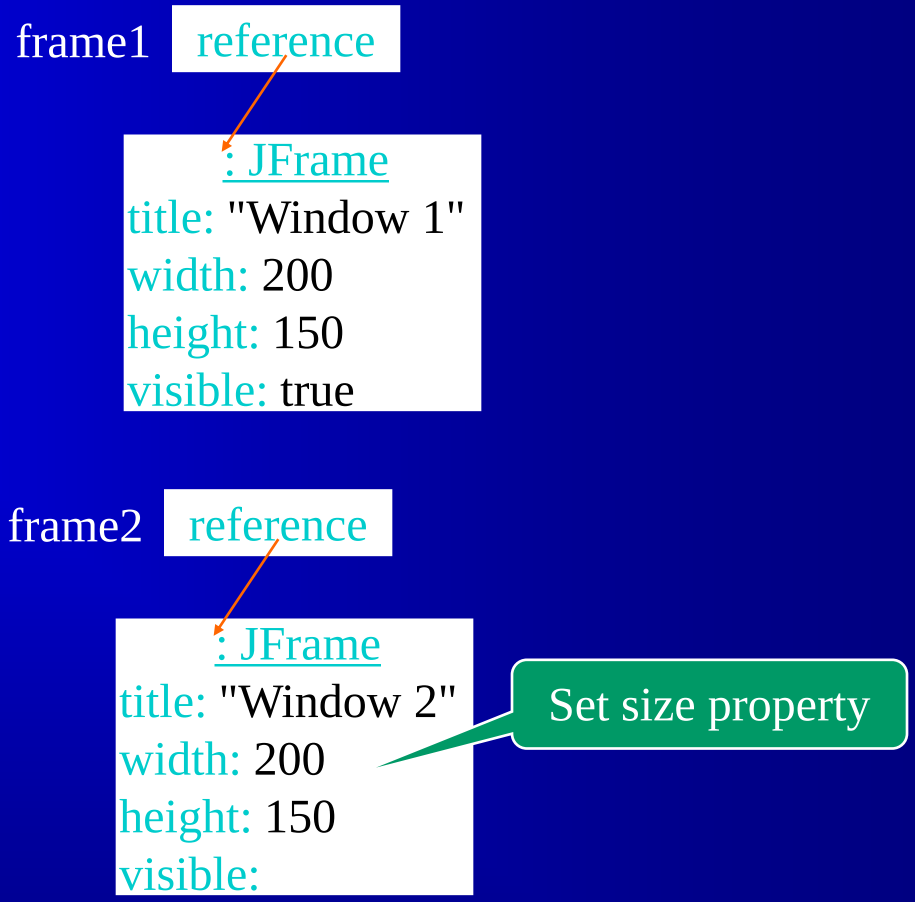
: JFrame
title: "Window 2"
width:
height:
visible:

Set title property

animation

Trace Code

```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```

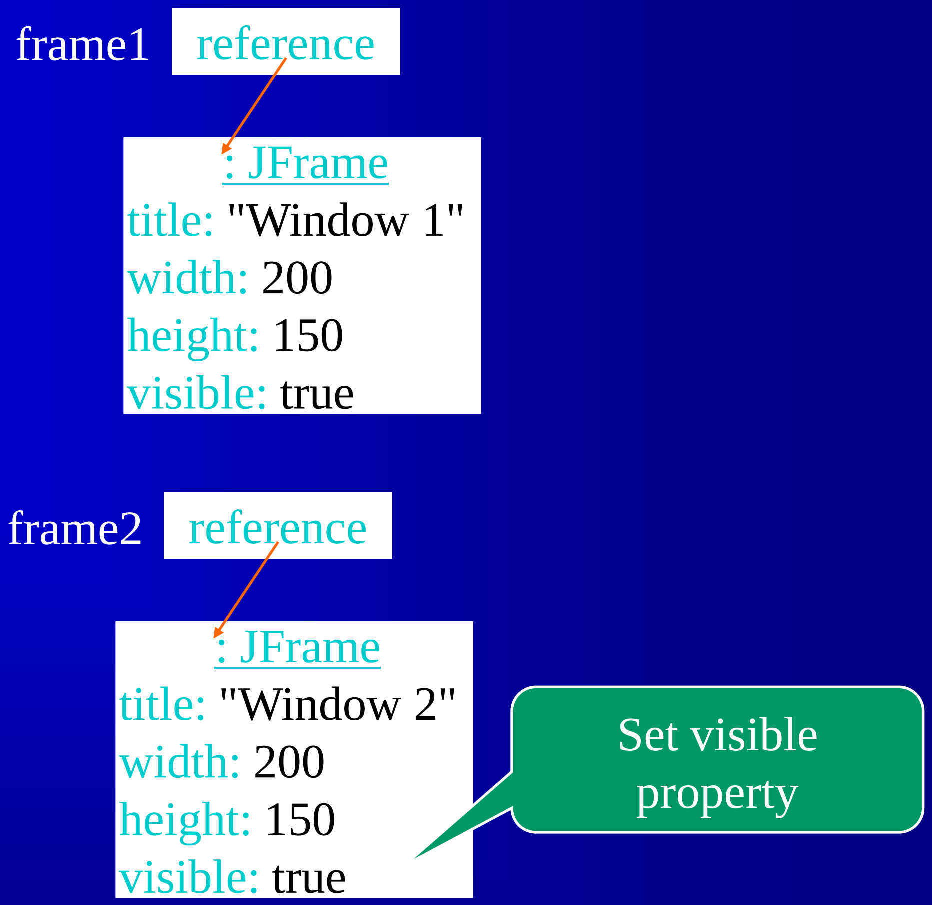


Set size property

animation

Trace Code

```
JFrame frame1 = new JFrame();
frame1.setTitle("Window 1");
frame1.setSize(200, 150);
frame1.setVisible(true);
JFrame frame2 = new JFrame();
frame2.setTitle("Window 2");
frame2.setSize(200, 150);
frame2.setVisible(true);
```



frame1 reference

: JFrame
title: "Window 1"
width: 200
height: 150
visible: true

frame2 reference

: JFrame
title: "Window 2"
width: 200
height: 150
visible: true

Set visible property

Adding GUI Components to Window

You can add graphical user interface components, such as buttons, labels, text fields, combo boxes, lists, and menus, to the window. The components are defined using classes. Here is an example to create buttons, labels, text fields, check boxes, radio buttons, and combo boxes.

GUIComponents

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.

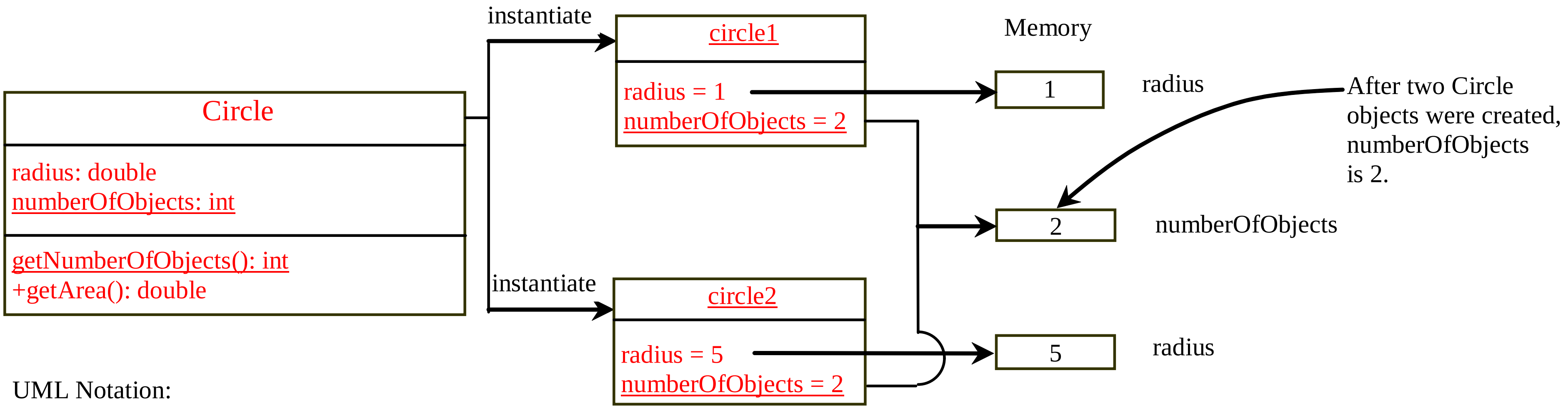
Static methods are not tied to a specific object.

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.



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

Example of Using Instance and Class Variables and Method

Objective: Demonstrate the roles of instance and class variables and their uses. This example adds a class variable numberOfObjects to track the number of Circle objects created.

CircleWithStaticMembers

TestCircleWithStaticMembers

Visibility Modifiers and Accessor/Mutator Methods

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

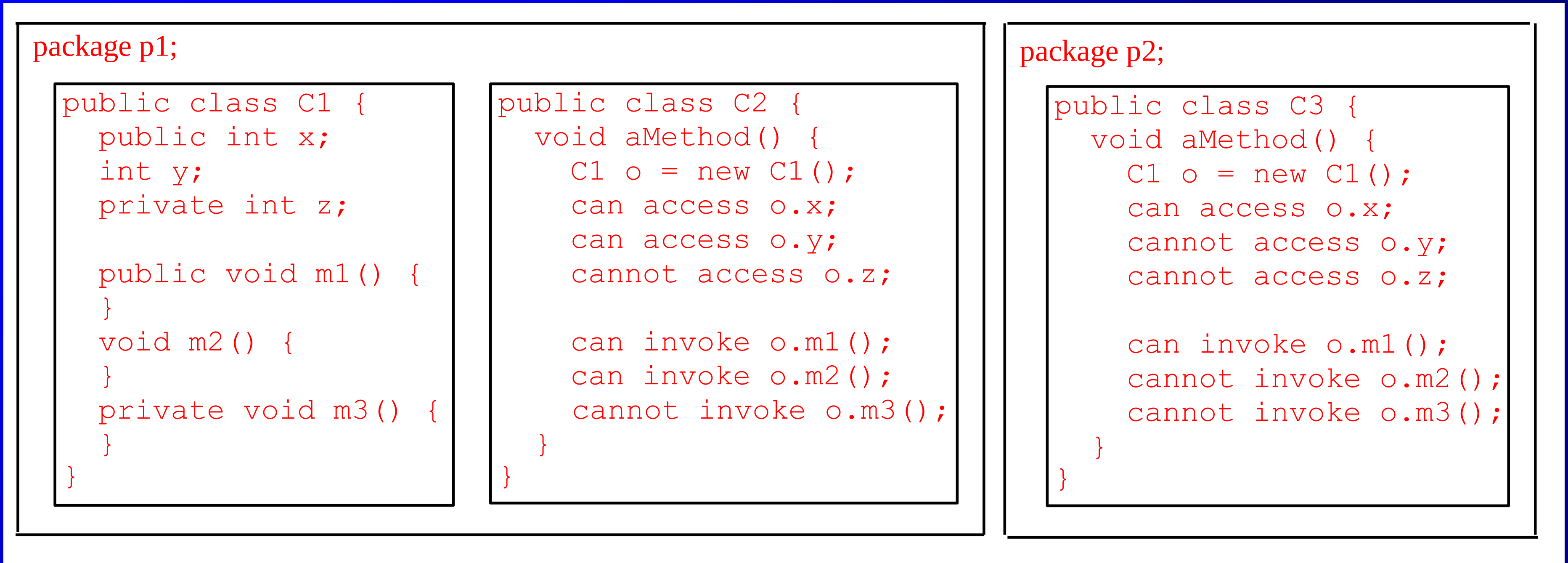
- ☞ `public`

  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() {
  }
}
```

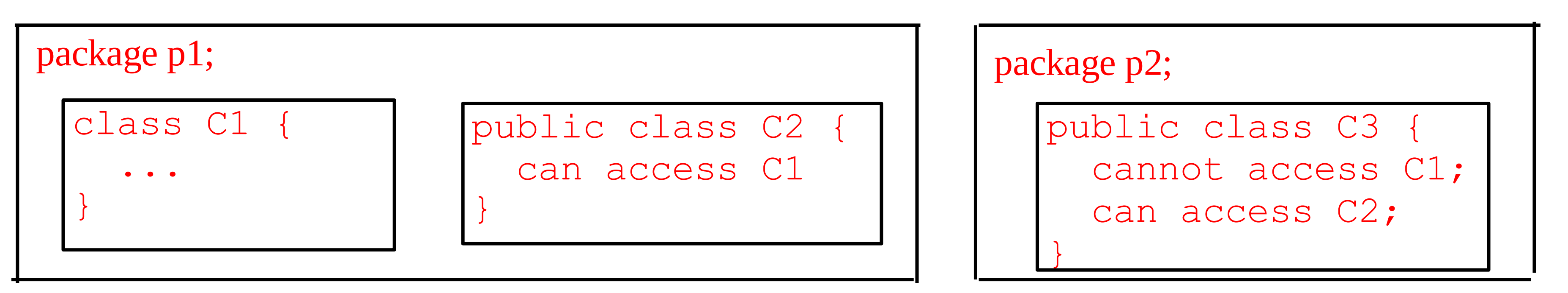
```
public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;

    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
}
```

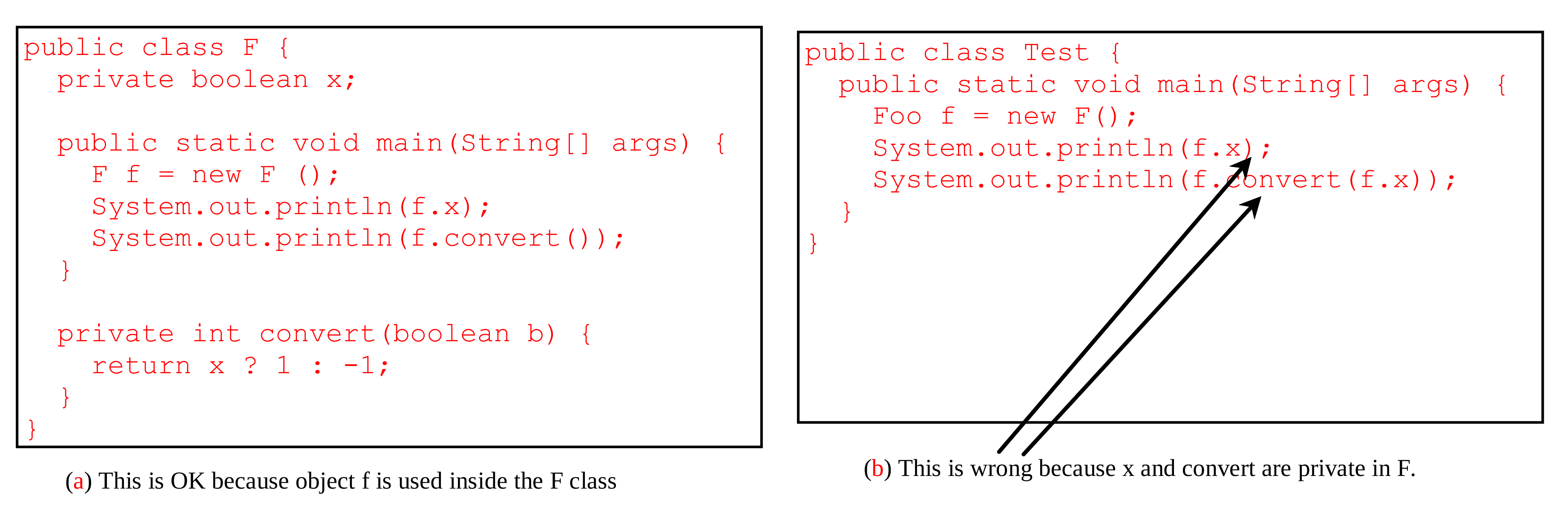
package p2;

```
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 F {
  private boolean x;

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

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

(a) This is OK because object f is used inside the F class

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

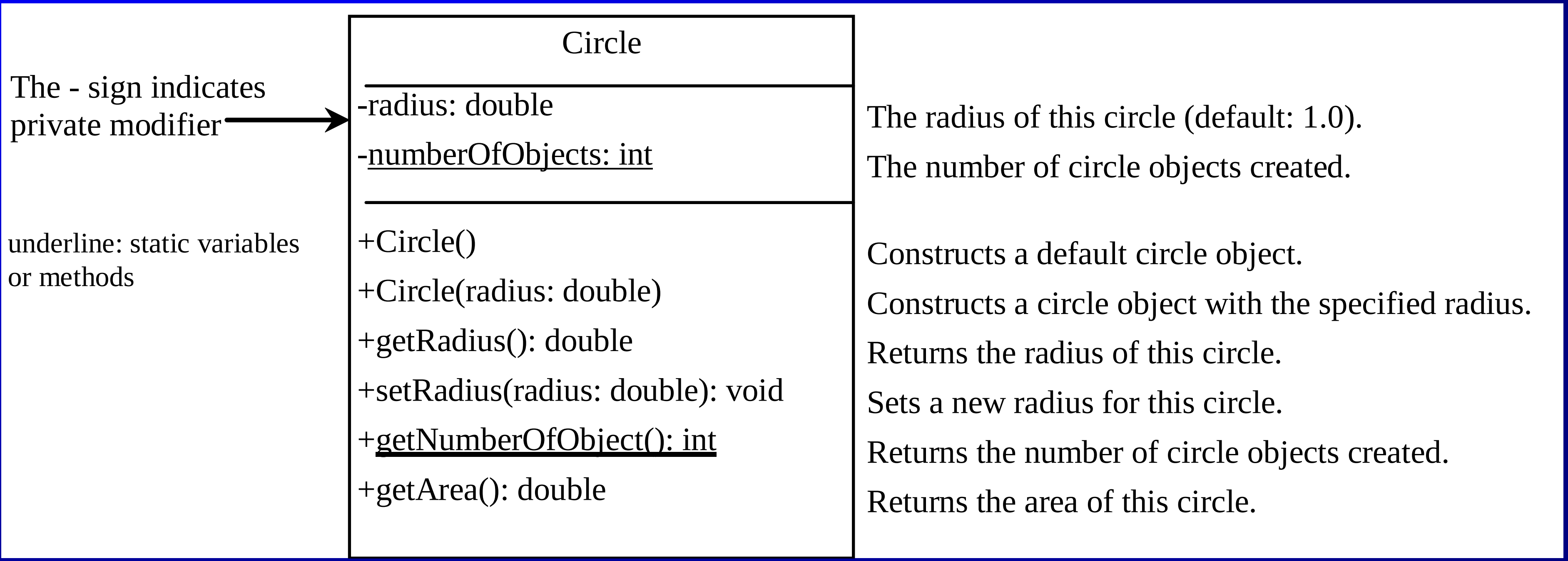
(b) This is wrong because x and convert are private in F.

Why Data Fields Should Be private?

To protect data.

To make class easy to maintain.

Example of
Data Field Encapsulation



The - sign indicates private modifier

underline: static variables or methods

Circle

-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.

CircleWithPrivateDataFields

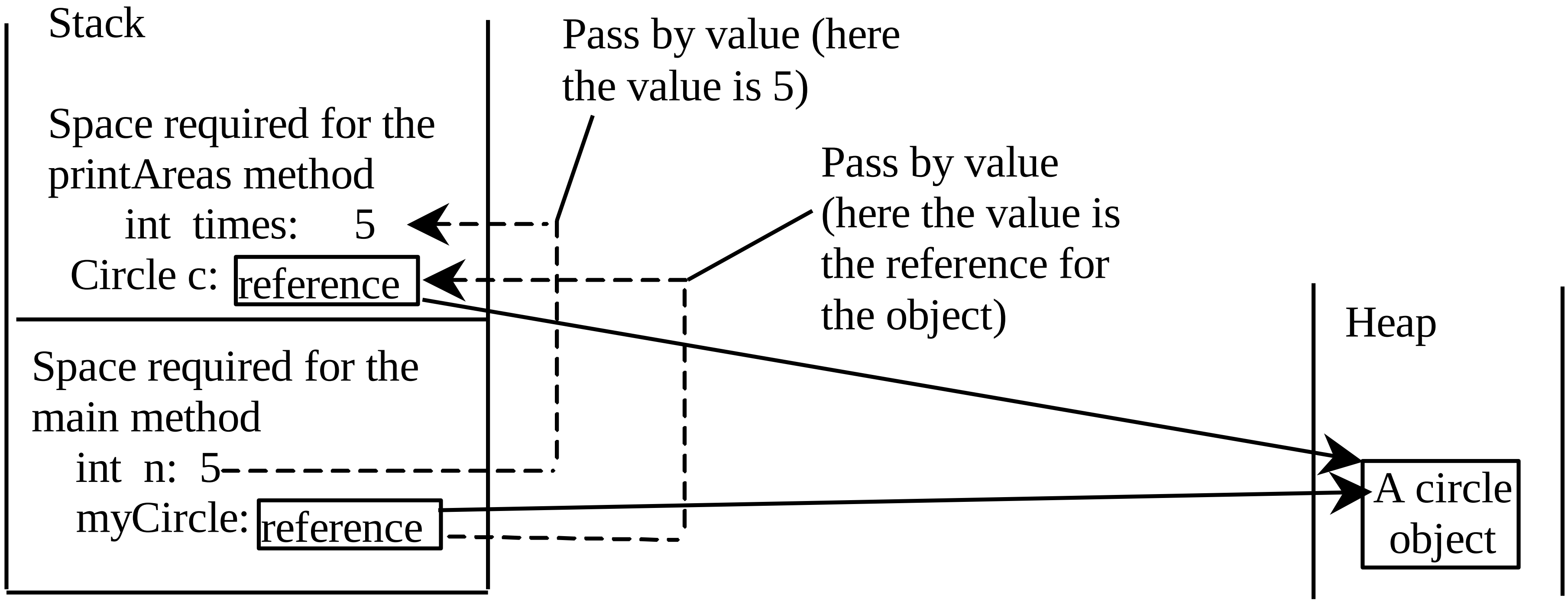
TestCircleWithPrivateDataFields

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)

TestPassObject

Passing Objects to Methods, cont.


Stack
Space required for the printAreas method
int times: 5
Circle c: reference
Space required for the main method
int n: 5
myCircle: reference
Pass by value (here the value is 5)
Pass by value (here the value is the reference for the object)
Heap
A circle object

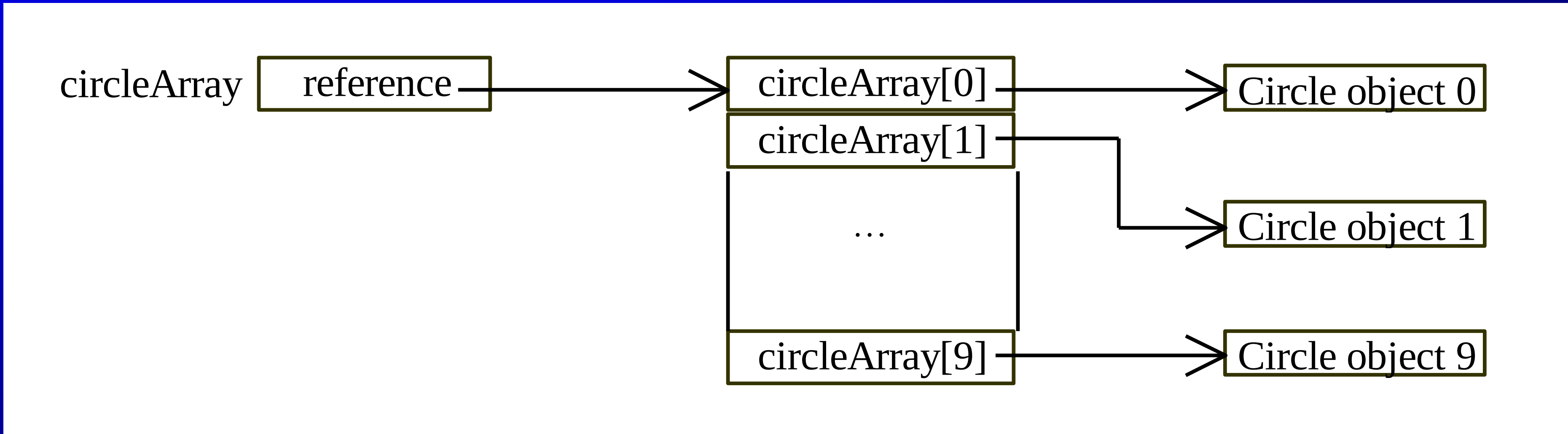
Array of Objects

```
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];
```



Array of Objects, cont.

Summarizing the areas of the circles

TotalArea