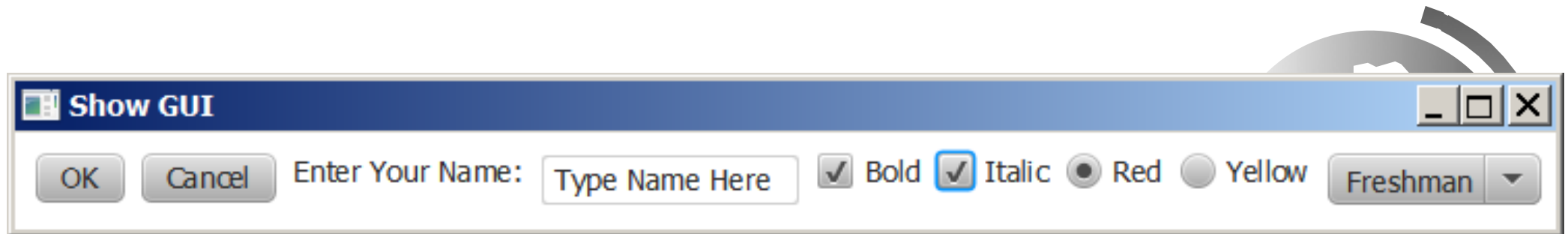


Chapter 9 Objects and Classes



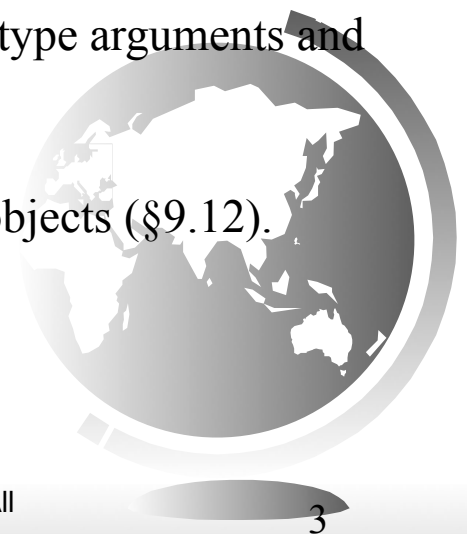
Motivations

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?



Objectives

- ❑ To describe objects and classes, and use classes to model objects (§9.2).
- ❑ To use UML graphical notation to describe classes and objects (§9.2).
- ❑ To demonstrate how to define classes and create objects (§9.3).
- ❑ To create objects using constructors (§9.4).
- ❑ To access objects via object reference variables (§9.5).
- ❑ To define a reference variable using a reference type (§9.5.1).
- ❑ To access an object's data and methods using the object member access operator (.) (§9.5.2).
- ❑ To define data fields of reference types and assign default values for an object's data fields (§9.5.3).
- ❑ To distinguish between object reference variables and primitive data type variables (§9.5.4).
- ❑ To use the Java library classes **Date**, **Random**, and **Point2D** (§9.6).
- ❑ To distinguish between instance and static variables and methods (§9.7).
- ❑ To define private data fields with appropriate **get** and **set** methods (§9.8).
- ❑ To encapsulate data fields to make classes easy to maintain (§9.9).
- ❑ To develop methods with object arguments and differentiate between primitive-type arguments and object-type arguments (§9.10).
- ❑ To store and process objects in arrays (§9.11).
- ❑ To create immutable objects from immutable classes to protect the contents of objects (§9.12).
- ❑ To determine the scope of variables in the context of a class (§9.13).
- ❑ To use the keyword **this** to refer to the calling object itself (§9.14).

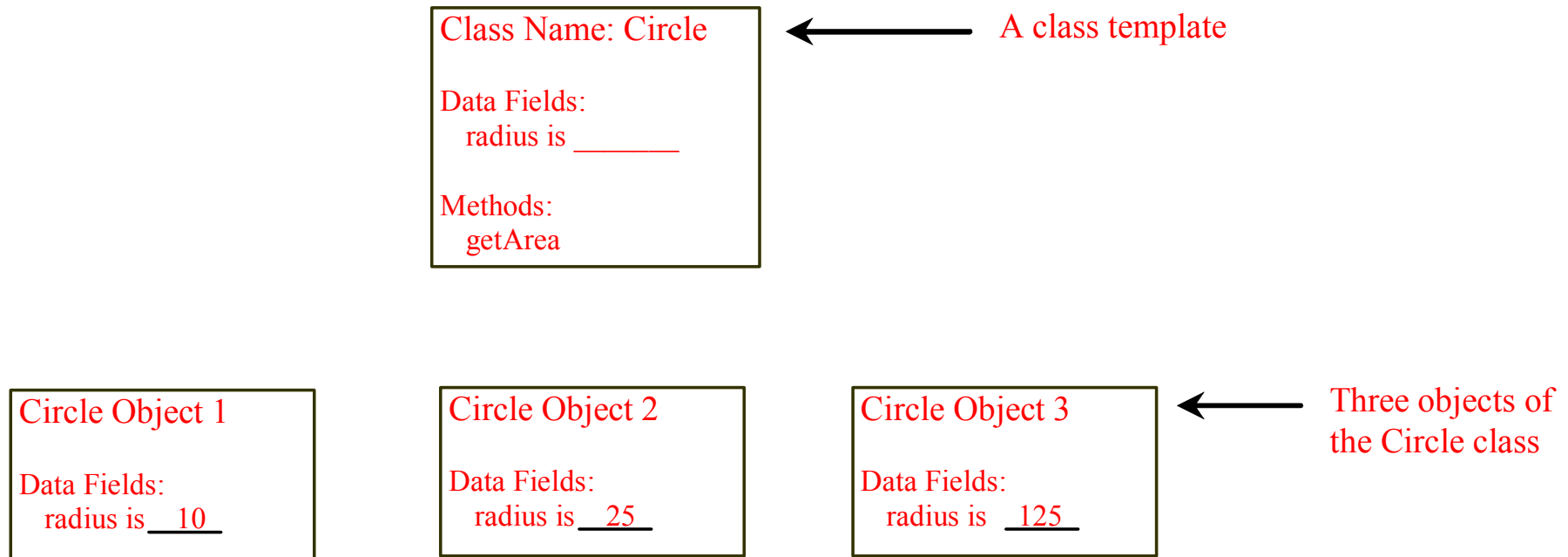


OO Programming Concepts

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



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;  
    }  
}
```

← Data field

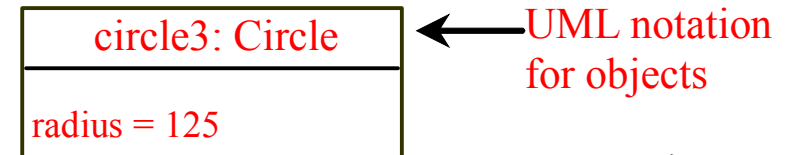
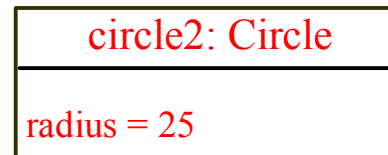
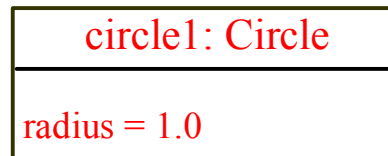
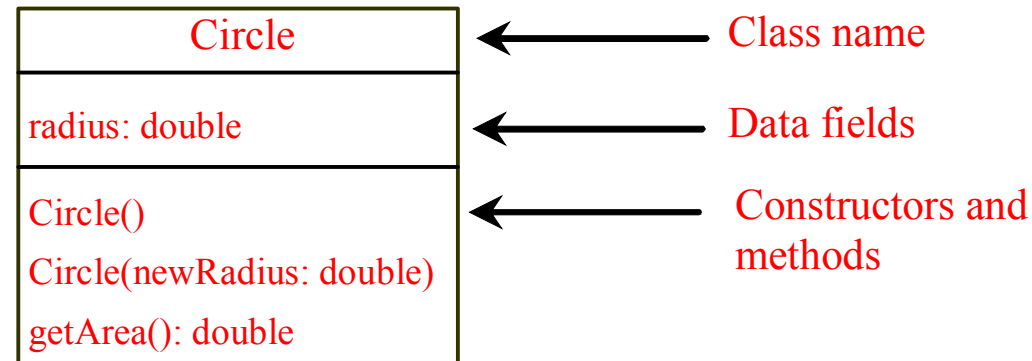
← Constructors

← Method



UML Class Diagram

UML Class Diagram



Example: Defining Classes and Creating Objects

Objective: Demonstrate creating objects, accessing data, and using methods.

TestSimpleCircle.java

```
public class TestSimpleCircle {  
    /** Main method */  
    public static void main(String[] args) {  
        // Create a circle with radius 1  
        SimpleCircle circle1 = new SimpleCircle();  
        System.out.println("The area of the circle of radius "  
            + circle1.radius + " is " + circle1.getArea());  
  
        // Create a circle with radius 25  
        SimpleCircle circle2 = new SimpleCircle(25);  
        System.out.println("The area of the circle of radius "  
            + circle2.radius + " is " + circle2.getArea());  
  
        // Create a circle with radius 125  
        SimpleCircle circle3 = new SimpleCircle(125);  
        System.out.println("The area of the circle of radius "  
            + circle3.radius + " is " + circle3.getArea());  
  
        // Modify circle radius  
        circle2.radius = 100; // or circle2.setRadius(100)  
        System.out.println("The area of the circle of radius "  
            + circle2.radius + " is " + circle2.getArea());  
    }  
}
```

Example: Defining Classes and Creating Objects

TV	
channel: int	
volumeLevel: int	
on: boolean	
<hr/>	
+TV()	
+turnOn(): void	
+turnOff(): void	
+setChannel(newChannel: int): void	
+setVolume(newVolumeLevel: int): void	
+channelUp(): void	
+channelDown(): void	
+volumeUp(): void	
+volumeDown(): void	

The + sign indicates
a public modifier. →

The current channel (1 to 120) of this TV.
The current volume level (1 to 7) of this TV.
Indicates whether this TV is on/off.

Constructs a default TV object.

Turns on this TV.

Turns off this TV.

Sets a new channel for this TV.

Sets a new volume level for this TV.

Increases the channel number by 1.

Decreases the channel number by 1.

Increases the volume level by 1.

Decreases the volume level by 1.

TV.java

TestTV.java



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 defined 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();
```

Assign object reference

Create an object

Example:

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



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 ()`

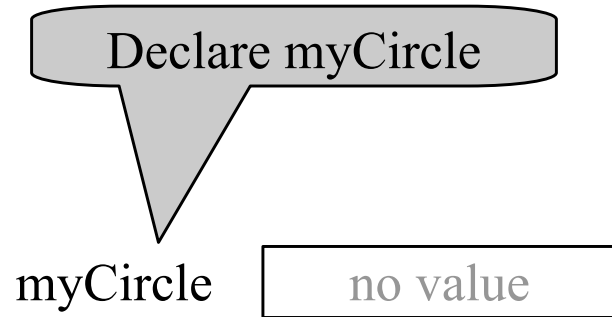


Trace Code

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

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```



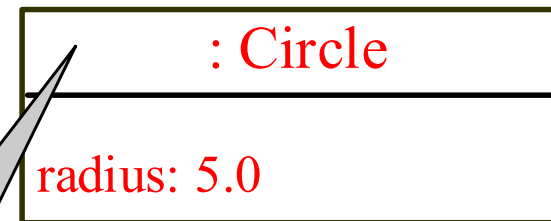
Trace Code, cont.

Circle myCircle = new Circle(5.0);

myCircle no value

Circle yourCircle = new Circle();

yourCircle.radius = 100;



Create a circle



Trace Code, cont.

Circle myCircle  **new Circle(5.0);**

Circle yourCircle = new Circle();

yourCircle.radius = 100;

myCircle

reference value

Assign object reference
to myCircle

: Circle

radius: 5.0



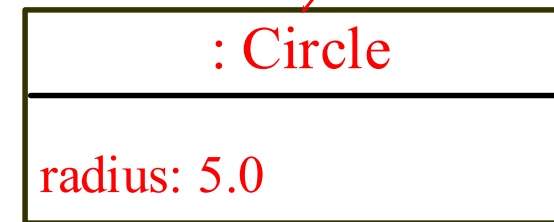
Trace Code, cont.

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

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle reference value



yourCircle no value

Declare yourCircle

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

Create a new
Circle object

: Circle

radius: 1.0

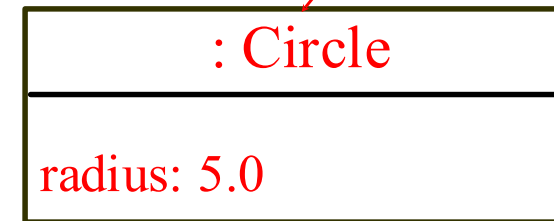
Trace Code, cont.

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

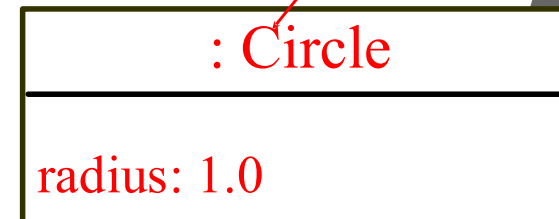
yourCircle.radius = 100;

myCircle **reference value**



yourCircle **reference value**

Assign object reference
to yourCircle



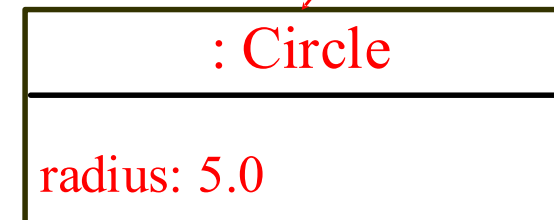
Trace Code, cont.

Circle myCircle = new Circle(5.0);

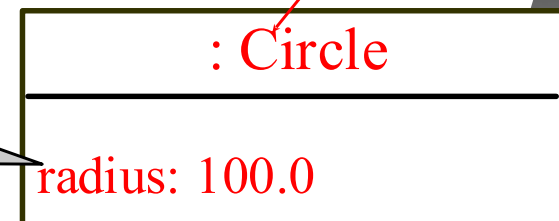
Circle yourCircle = new Circle();

yourCircle.radius = 100;

myCircle reference value



yourCircle reference value



Change radius in
yourCircle

Caution

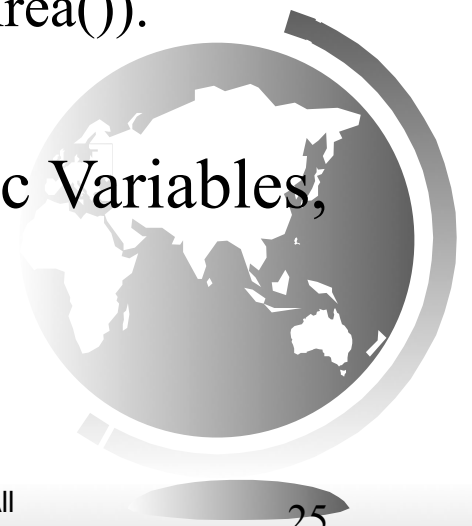
Recall that you use

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

to invoke a method in the `Math` class. Can you invoke `getArea()` using `SimpleCircle.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);  
    }  
}
```

name? null
age? 0
isScienceMajor? false
gender?



Example

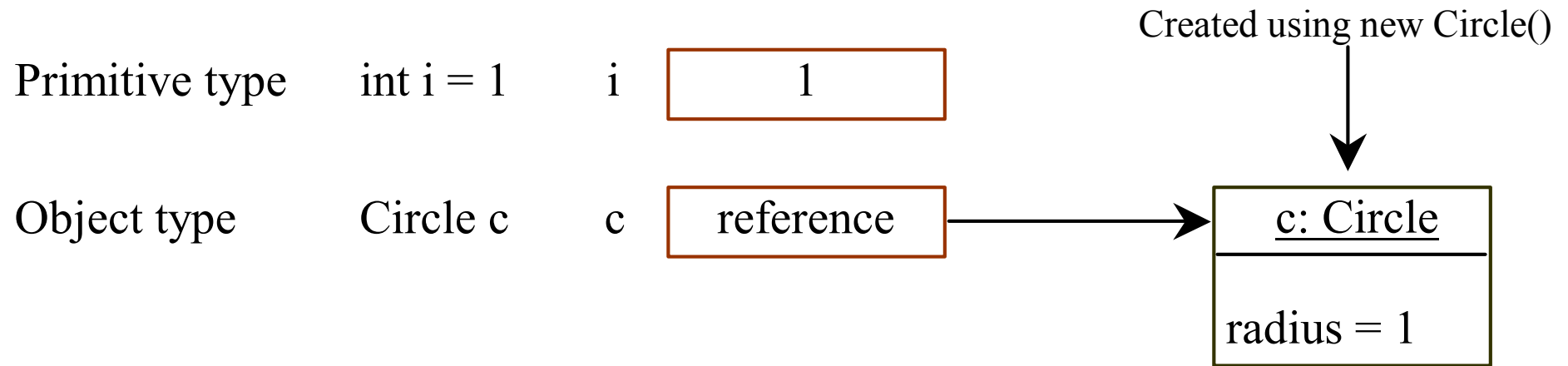
Java assigns no default value to a local variable inside a method.

```
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);  
    }  
}
```

Compile error: variable not initialized



Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types

Primitive type assignment $i = j$

Before:

i

1

j

2

After:

i

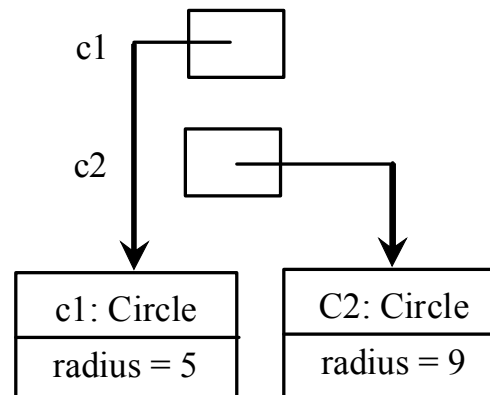
2

j

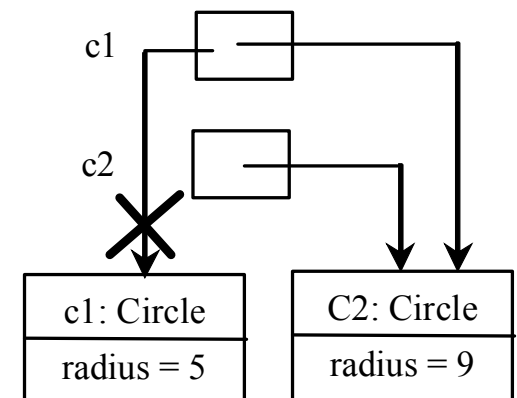
2

Object type assignment $c1 = c2$

Before:



After:



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

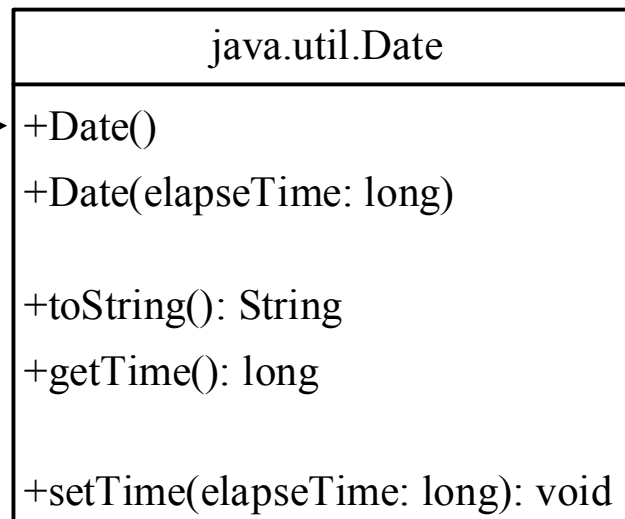
TIP: If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object. The JVM will automatically collect the space if the object is not referenced by any variable.



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 modifier



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.



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



The **Point2D** Class

Java API has a convenient **Point2D** class in the **javafx.geometry** package for representing a point in a two-dimensional plane.

javafx.geometry.Point2D

```
+Point2D(x: double, y: double)
+distance(x: double, y: double): double
+distance(p: Point2D): double
+getX(): double
+getY(): double
+toString(): String
```

Constructs a **Point2D** object with the specified *x*- and *y*-coordinates.
Returns the distance between this point and the specified point (*x*, *y*).
Returns the distance between this point and the specified point *p*.
Returns the *x*-coordinate from this point.
Returns the *y*-coordinate from this point.
Returns a string representation for the point.

TestPoint2D.java



```
package chapter9;

import java.util.Scanner;
import javafx.geometry.Point2D;

public class TestPoint2D {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter point1's x-, y-coordinates: ");
        double x1 = input.nextDouble();
        double y1 = input.nextDouble();
        System.out.print("Enter point2's x-, y-coordinates: ");
        double x2 = input.nextDouble();
        double y2 = input.nextDouble();

        Point2D p1 = new Point2D(x1, y1);
        Point2D p2 = new Point2D(x2, y2);
        System.out.println("p1 is " + p1.toString());
        System.out.println("p2 is " + p2.toString());
        System.out.println("The distance between p1 and p2 is " + p1.distance(p2));
        System.out.println("The midpoint between p1 and p2 is " + p1.midpoint(p2).toString());
    }
}
```

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.

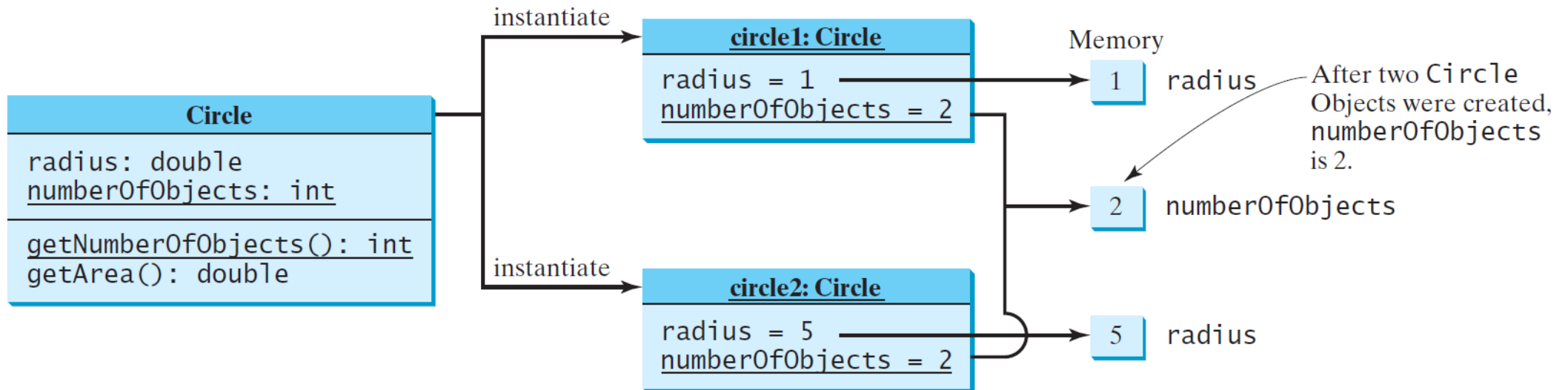
To declare static variables, constants, and methods,
use the static modifier.



Static Variables, Constants, and Methods, cont.

UML Notation:

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`



```

public class CircleWithStaticMembers {
    /** The radius of the circle */
    double radius;

    /** The number of the objects created */
    static int numberOfObjects = 0;

    /** Construct a circle with radius 1 */
    CircleWithStaticMembers() {
        radius = 1.0;
        numberOfObjects++;
    }

    /** Construct a circle with a specified radius */
    CircleWithStaticMembers(double newRadius) {
        radius = newRadius;
        numberOfObjects++;
    }

    /** Return numberOfObjects */
    static int getNumberOfObjects() {
        return numberOfObjects;
    }

    /** Return the area of this circle */
    double getArea() {
        return radius * radius * Math.PI;
    }
}

```

```

public class TestCircleWithStaticMembers {
    /** Main method */
    public static void main(String[] args) {
        System.out.println("Before creating objects");
        System.out.println("The number of Circle objects is " +
            CircleWithStaticMembers.numberOfObjects);

        // Create c1
        CircleWithStaticMembers c1 = new CircleWithStaticMembers();

        // Display c1 BEFORE c2 is created
        System.out.println("\nAfter creating c1");
        System.out.println("c1: radius (" + c1.radius +
            ") and number of Circle objects (" +
            c1.numberOfObjects + ")");

        // Create c2
        CircleWithStaticMembers c2 = new CircleWithStaticMembers(5);

        // Modify c1
        c1.radius = 9;

        // Display c1 and c2 AFTER c2 was created
        System.out.println("\nAfter creating c2 and modifying c1");
        System.out.println("c1: radius (" + c1.radius +
            ") and number of Circle objects (" +
            c1.numberOfObjects + ")");
        System.out.println("c2: radius (" + c2.radius +
            ") and number of Circle objects (" +
            c2.numberOfObjects + ")");
    }
}

```

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() {
    }
}

```

```

package p1;

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 {
    ...
}

```

```

package p1;

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.

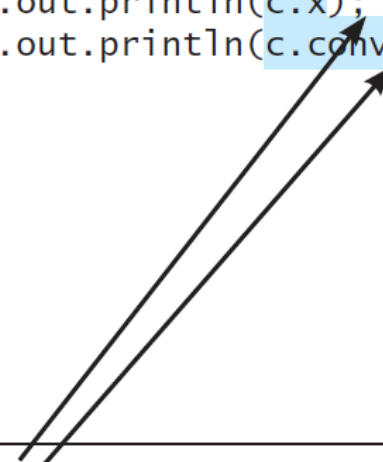
NOTE

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 C {  
    private boolean x;  
  
    public static void main(String[] args) {  
        C c = new C();  
        System.out.println(c.x);  
        System.out.println(c.convert());  
    }  
  
    private int convert() {  
        return x ? 1 : -1;  
    }  
}
```

(a) This is okay because object `c` is used inside the class `C`.

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



(b) This is wrong because `x` and `convert` are private in class `C`.

Why Data Fields Should Be private?

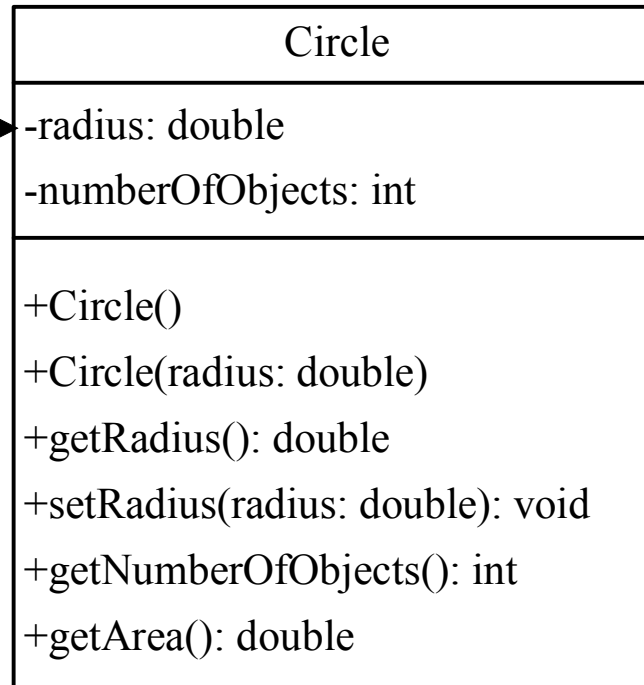
To protect data.

To make code easy to maintain.



Example of Data Field Encapsulation

The - sign indicates
private modifier



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



```

public class CircleWithPrivateDataFields {
    /** The radius of the circle */
    private double radius = 1;

    /** The number of the objects created */
    private static int numberOfObjects = 0;
    /** Construct a circle with radius 1 */
    public CircleWithPrivateDataFields() {
        numberOfObjects++;
    }
    /** Construct a circle with a specified radius */
    public CircleWithPrivateDataFields(double newRadius) {
        radius = newRadius;
        numberOfObjects++;
    }
    /** Return radius */
    public double getRadius() {
        return radius;
    }
    /** Set a new radius */
    public void setRadius(double newRadius) {
        radius = (newRadius >= 0) ? newRadius : 0;
    }
    /** Return numberOfObjects */
    public static int getNumberOfObjects() {
        return numberOfObjects;
    }
    /** Return the area of this circle */
    public double getArea() {
        return radius * radius * Math.PI;
    }
}

```

```

public class TestCircleWithPrivateDataFields {
    /** Main method */
    public static void main(String[] args) {
        // Create a Circle with radius 5.0
        CircleWithPrivateDataFields myCircle =
            new CircleWithPrivateDataFields(5.0);
        System.out.println("The area of the circle of radius "
            + myCircle.getRadius() + " is " + myCircle.getArea());

        // Increase myCircle's radius by 10%
        myCircle.setRadius(myCircle.getRadius() * 1.1);
        System.out.println("The area of the circle of radius "
            + myCircle.getRadius() + " is " + myCircle.getArea());
    }
}

```



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.java



```

public class TestPassObject {
    /** Main method */
    public static void main(String[] args) {
        // Create a Circle object with radius 1
        CircleWithPrivateDataFields myCircle =
            new CircleWithPrivateDataFields(1);

        // Print areas for radius 1, 2, 3, 4, and 5.
        int n = 5;
        printAreas(myCircle, n);

        // See myCircle.radius and times
        System.out.println("\n" + "Radius is " + myCircle.getRadius());
        System.out.println("n is " + n);
    }

    /** Print a table of areas for radius */
    public static void printAreas(
        CircleWithPrivateDataFields c, int times) {
        System.out.println("Radius \t\tArea");
        while (times >= 1) {
            System.out.println(c.getRadius() + "\t\t" + c.getArea());
            c.setRadius(c.getRadius() + 1);
            times--;
        }
    }
}

```

Radius	Area
1.0	3.141592653589793
2.0	12.566370614359172
3.0	28.274333882308138
4.0	50.26548245743669
5.0	78.53981633974483

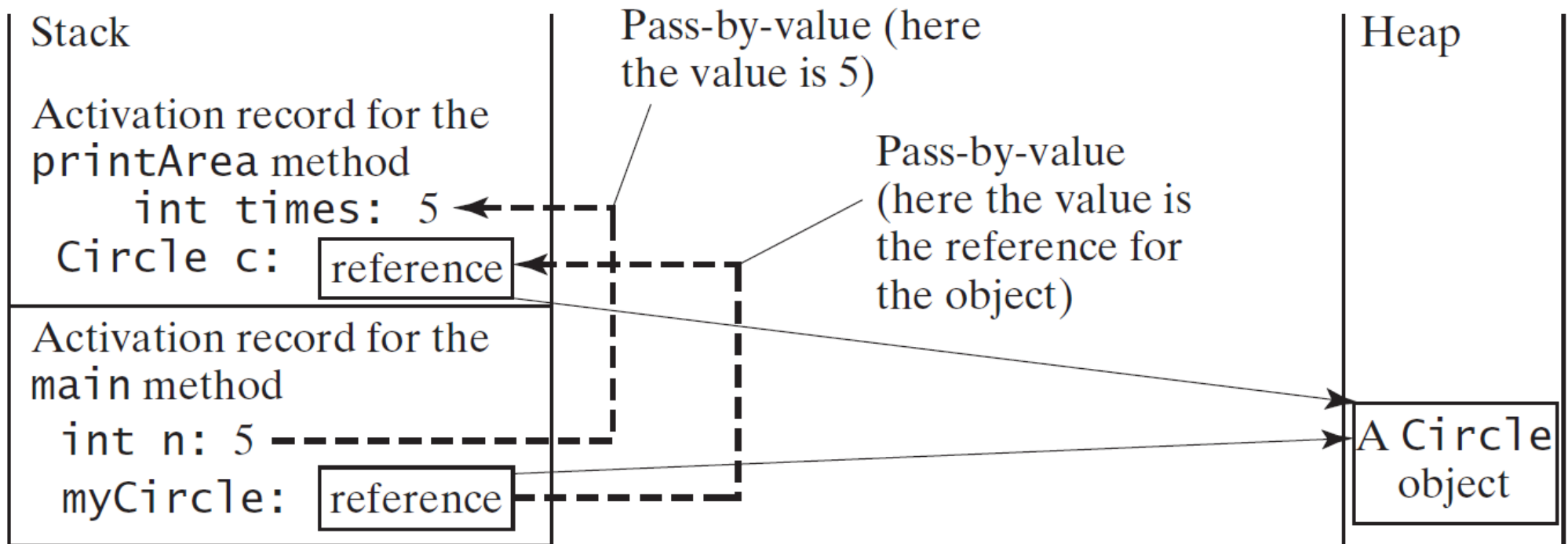
```

Radius is 6.0
n is 5

```




Passing Objects to Methods, cont.



Array of Objects

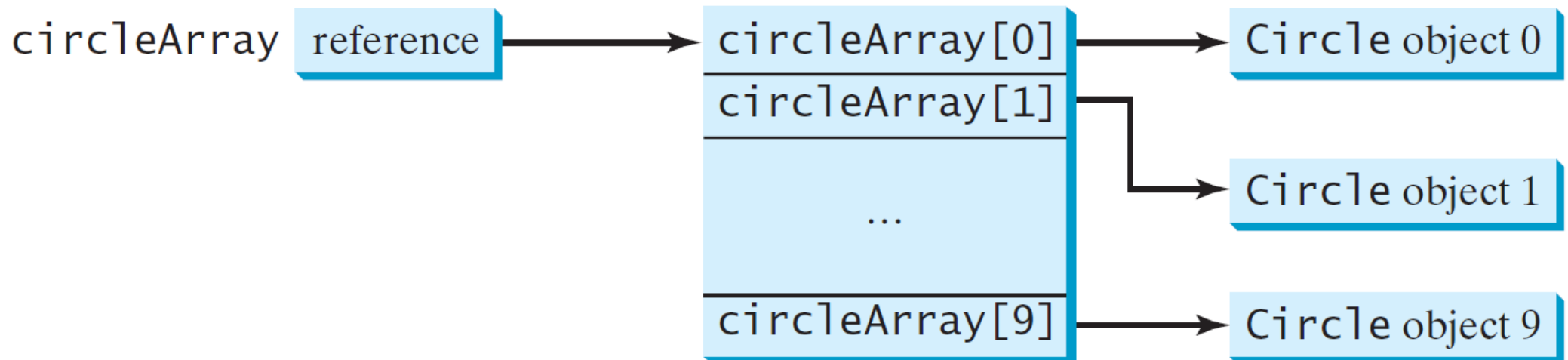
```
Circle[] circleArray = new Circle[10];
```

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.

Summarising the areas of the circles

TotalArea.java



```

public class TotalArea {
    /** Main method */
    public static void main(String[] args) {
        CircleWithPrivateDataFields[] circleArray;
        circleArray = createCircleArray();
        printCircleArray(circleArray);
    }
    /** Create an array of Circle objects */
    public static CircleWithPrivateDataFields[] createCircleArray() {
        CircleWithPrivateDataFields[] circleArray = new CircleWithPrivateDataFields[5];
        for (int i = 0; i < circleArray.length; i++) {
            circleArray[i] = new CircleWithPrivateDataFields(Math.random() * 100);
        }
        // Return Circle array
        return circleArray;
    }
    /** Print an array of circles and their total area */
    public static void printCircleArray( CircleWithPrivateDataFields[] circleArray) {
        System.out.printf("%-30s%-15s\n", "Radius", "Area");
        for (int i = 0; i < circleArray.length; i++) {
            System.out.printf("%-30f%-15f\n", circleArray[i].getRadius(), circleArray[i].getArea());
        }
        System.out.printf("%-30s%-15f\n", "The total areas of circles is", sum(circleArray));
    }
    /** Add circle areas */
    public static double sum(CircleWithPrivateDataFields[] circleArray) {
        double sum = 0;
        // Add areas to sum
        for (int i = 0; i < circleArray.length; i++)
            sum += circleArray[i].getArea();
        return sum;
    }
}

```

Radius	Area
91.673799	26402.213135
82.362309	21311.150837
56.689141	10096.006571
29.595337	2751.670645
82.769876	21522.587517

The total areas of circles is 82083.628705	



Immutable Objects and Classes

If the contents of an object cannot be changed once the object is created, the object is called an *immutable object* and its class is called an *immutable class*. If you delete the set method in the Circle class in TestSimpleCircle.java, the class would be immutable because radius is private and cannot be changed without a set method.

A class with all private data fields and without mutators is not necessarily immutable. For example, the following class Student has all private data fields and no mutators, but it is mutable.



Example

```
public class Student {
    private int id;
    private BirthDate birthDate;

    public Student(int ssn,
        int year, int month, int day) {
        id = ssn;
        birthDate = new BirthDate(year, month, day);
    }

    public int getId() {
        return id;
    }

    public BirthDate getBirthDate() {
        return birthDate;
    }
}
```

```
public class BirthDate {
    private int year;
    private int month;
    private int day;

    public BirthDate(int newYear,
        int newMonth, int newDay) {
        year = newYear;
        month = newMonth;
        day = newDay;
    }

    public void setYear(int newYear) {
        year = newYear;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Student student = new Student(111223333, 1970, 5, 3);
        BirthDate date = student.getBirthDate();
        date.setYear(2010); // Now the student birth year is changed!
    }
}
```



What Class is Immutable?

For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.



Scope of Variables

- ❑ The scope of instance and static variables is the entire class. They can be declared anywhere inside a class.
- ❑ The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be initialized explicitly before it can be used.



The this Keyword

- ❑ The this keyword is the name of a reference that refers to an object itself. One common use of the this keyword is reference a class's *hidden data fields*.
- ❑ Another common use of the this keyword to enable a constructor to invoke another constructor of the same class.



Reference the Hidden Data Fields

```
public class F {  
    private int i = 5;  
    private static double k = 0;  
  
    void setI(int i) {  
        this.i = i;  
    }  
  
    static void setK(double k) {  
        F.k = k;  
    }  
}
```

Suppose that f1 and f2 are two objects of F.
F f1 = new F(); F f2 = new F();

Invoking f1.setI(10) is to execute
this.i = 10, where **this** refers f1


Invoking f2.setI(45) is to execute
this.i = 45, where **this** refers f2



Calling Overloaded Constructor

```
public class Circle {  
    private double radius;
```

```
    public Circle(double radius) {  
        this.radius = radius;  
    }
```

 this must be explicitly used to reference the data field radius of the object being constructed

```
    public Circle() {  
        this(1.0);  
    }
```

 this is used to invoke another constructor

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

 Every instance variable belongs to an instance represented by this, which is normally omitted

Exercises

- **Dia** is a free (open source) drawing software. Download from: dia-installer.de/
 - Design a class named **Account** that contains:
 - A private **int** data field named **id** for the account (default **0**).
 - A private **double** data field named **balance** for the account (default **0**).
 - A private **double** data field named **annualInterestRate** that stores the current interest rate (default **0**). Assume all accounts have the same interest rate.
 - A private **Date** data field named **dateCreated** that stores the date when the account was created.
 - A no-arg constructor that creates a default account.
 - A constructor that creates an account with the specified id and initial balance.
 - The accessor and mutator methods for **id**, **balance**, and **annualInterestRate**.
 - The accessor method for **dateCreated**.
 - A method named **getMonthlyInterestRate()** that returns the monthly interest rate.
 - A method named **getMonthlyInterest()** that returns the monthly interest.
 - A method named **withdraw** that withdraws a specified amount from the account.
 - A method named **deposit** that deposits a specified amount to the account.
- Draw the UML diagram for the class and then implement the class. (*Hint* : The method **getMonthlyInterest()** is to return monthly interest, not the interest rate. Monthly interest is **balance * monthlyInterestRate**. **monthlyInterestRate** is **annualInterestRate / 12**. Note that **annualInterestRate** is a percentage, e.g., like 4.5%. You need to divide it by 100.)
- Write a test program that creates an **Account** object with an account ID of 1122, a balance of \$20,000, and an annual interest rate of 4.5%. Use the **withdraw** method to withdraw \$2,500, use the **deposit** method to deposit \$3,000, and print the balance, the monthly interest, and the date when this account was created.