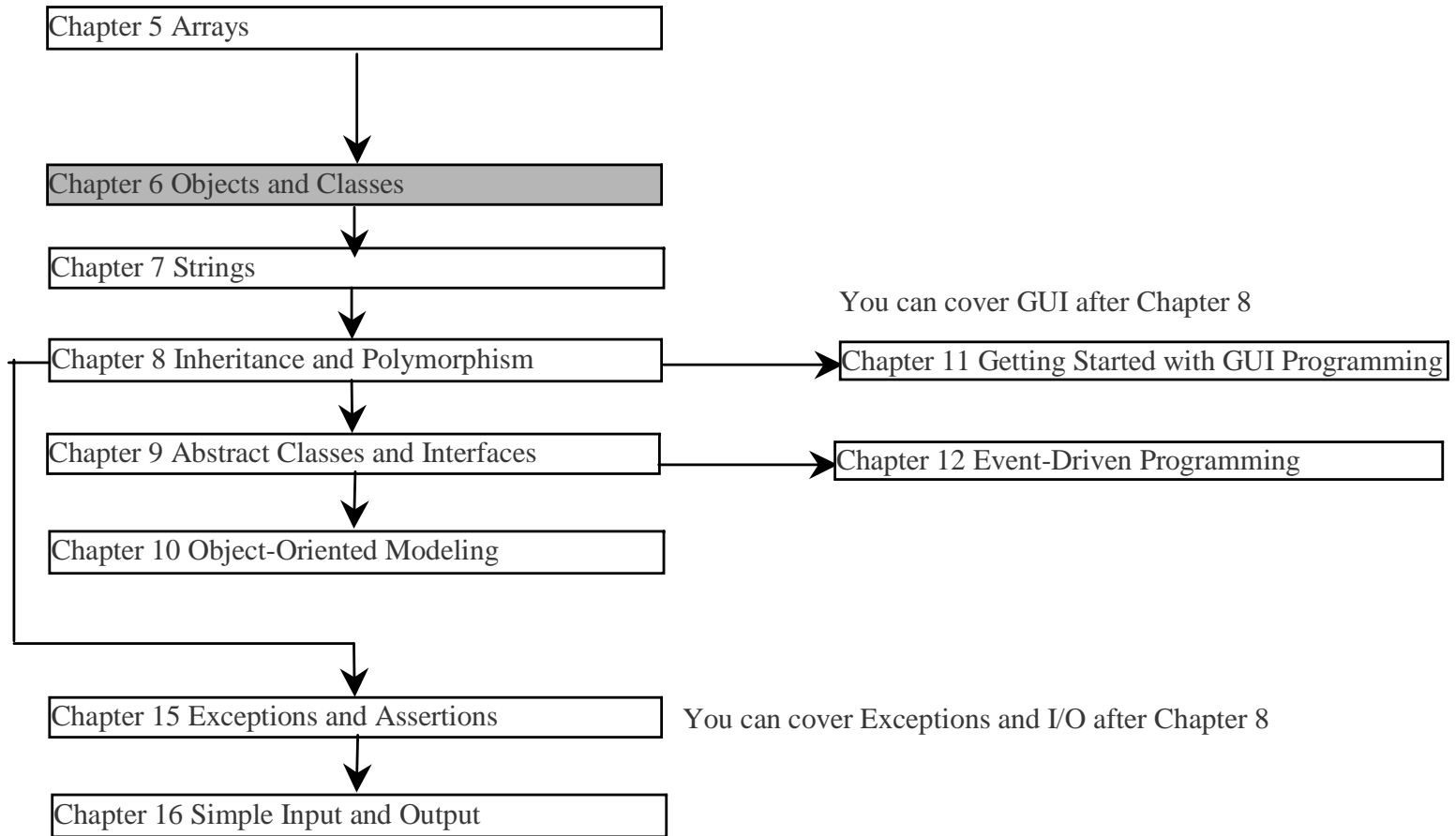


Chapter 6 Objects and Classes

Prerequisites for Part II



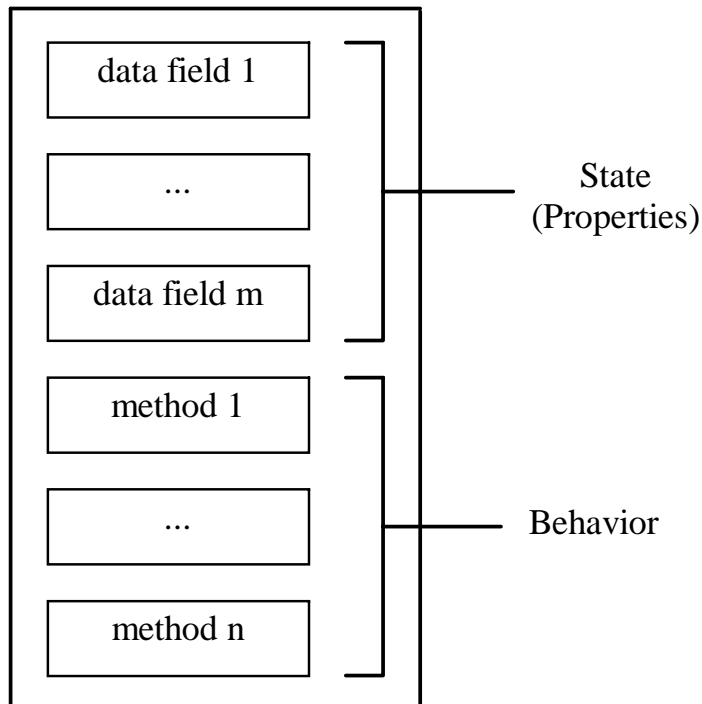
Objectives

- ☞ To understand objects and classes and use classes to model objects (§6.2).
- ☞ To learn how to declare a class and how to create an object of a class (§6.3).
- ☞ To understand the roles of constructors and use constructors to create objects (§6.3).
- ☞ To use UML graphical notations to describe classes and objects (§6.3).
- ☞ To distinguish between object reference variables and primitive data type variables (§6.4).
- ☞ To use classes in the Java library (§6.5).
- ☞ To declare private data fields with appropriate get and set methods to make class easy to maintain (§6.6-6.8).
- ☞ To develop methods with object arguments (§6.9).
- ☞ To understand the difference between instance and static variables and methods (§6.10).
- ☞ To determine the scope of variables in the context of a class (§6.11).
- ☞ To use the keyword this as the reference to the current object that invokes the instance method (§6.12).
- ☞ To store and process objects in arrays (§6.13).
- ☞ To apply class abstraction to develop software (§6.14).
- ☞ To declare inner classes (§6.17).

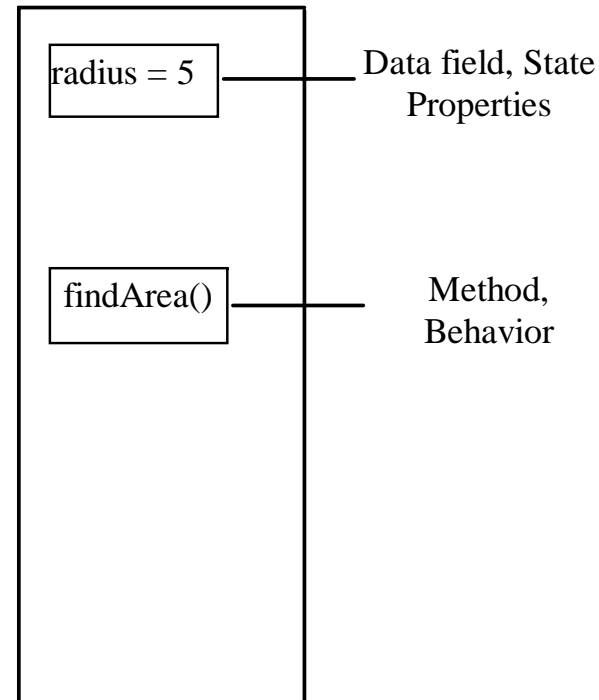
OO Programming Concepts

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



(A) A generic object



(B) An example of circle object

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 findArea() {  
        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:

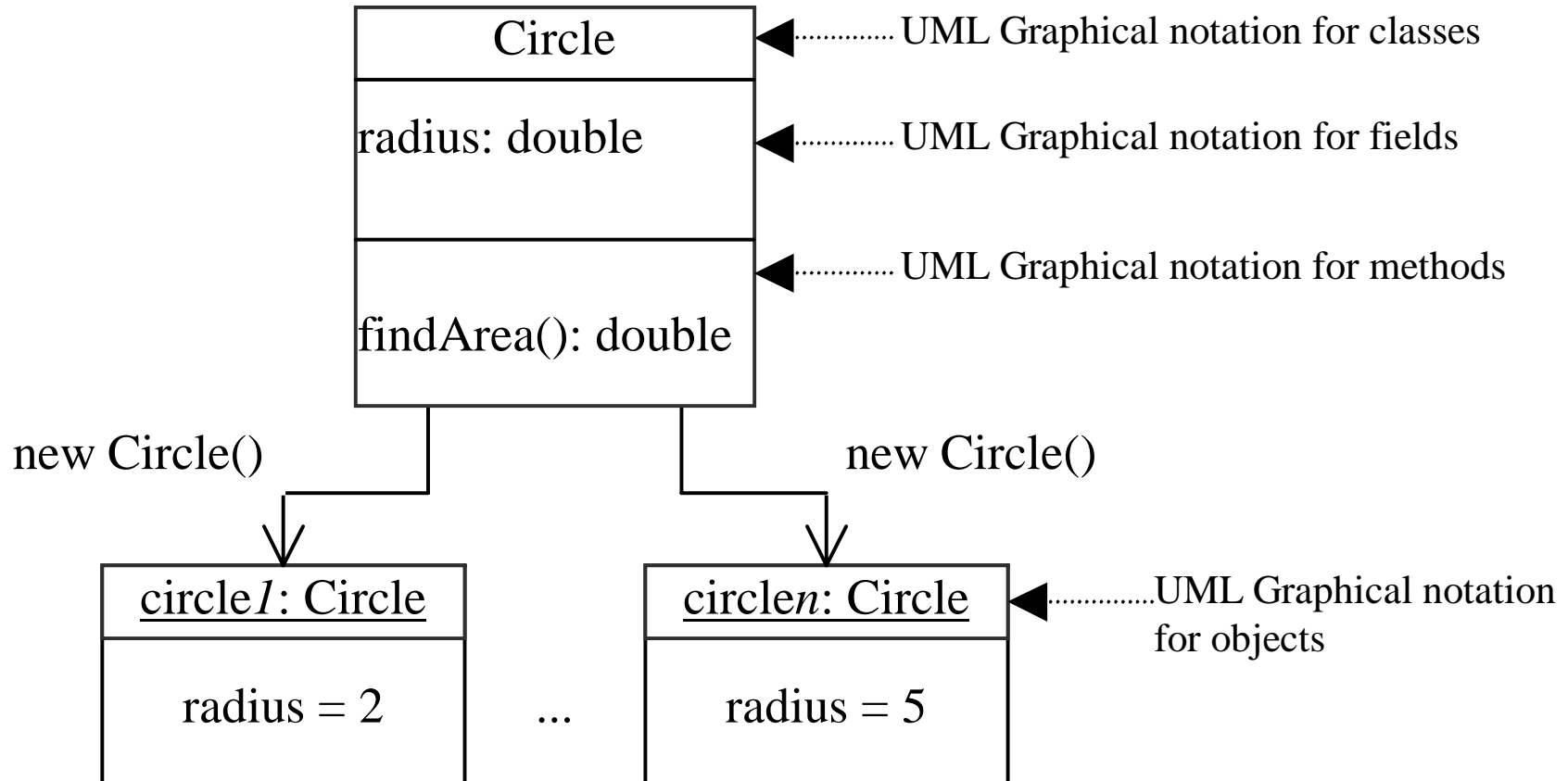
```
new Circle ( ) ;
```

```
new Circle ( 5.0 ) ;
```

Default Constructor

A class may be 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*.

Constructing Objects, cont.



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 Objects

- ☞ Referencing the object's data:

`objectRefVar.data`

e.g., `myCircle.radius`

- ☞ Invoking the object's method:

`objectRefVar.methodName(arguments)`

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

Example 6.1 Using Objects

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

TestSimpleCircle

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 findArea() using SimpleCircle.findArea()? The answer is no. All the methods used before this chapter are static methods, which are defined using the static keyword. However, findArea() is non-static. It must be invoked from an object using

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

More explanations will be given in Section 6.7, “Static Variables, Constants, and Methods.”

The null Value

If a variable of a reference type does not reference any object, the variable 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.

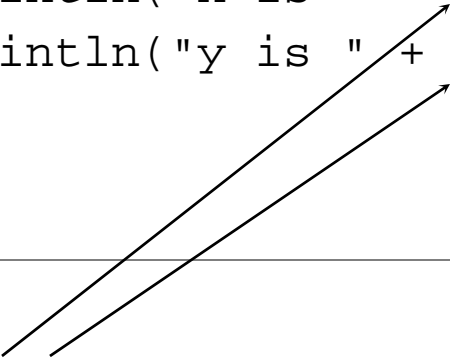
Example

```
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'

    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);
    }
}
```

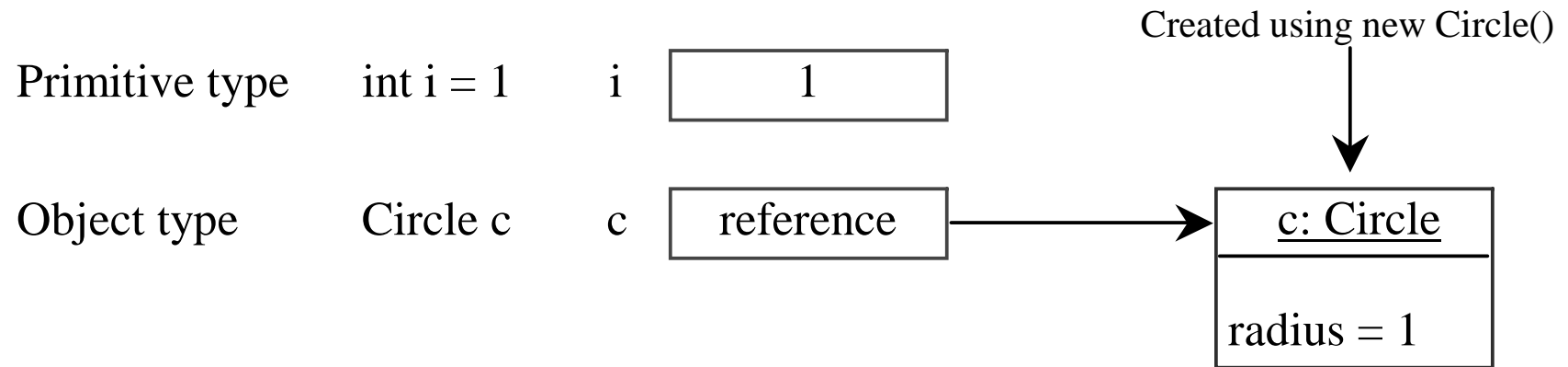
Example

```
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

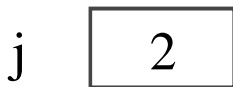
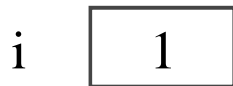
Differences between Variables of Primitive Data Types and Object Types



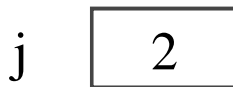
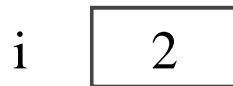
Copying Variables of Primitive Data Types and Object Types

Primitive type assignment
 $i = j$

Before:

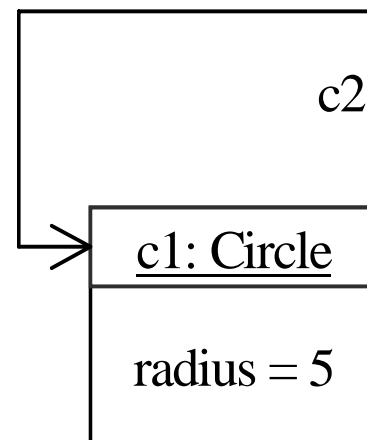


After:

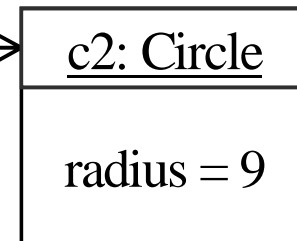


Object type assignment
 $c1 = c2$

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 useful. 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 Java VM will automatically collect the space if the object is not referenced by any variable .

Using Classes from the Java Library

Example 6.1 declared the SimpleCircle class and created objects from the class. Often you will use the classes in the Java library to develop programs. You learned to obtain the current time using System.currentTimeMillis() in Example 2.5, “Displaying Current Time.” You used the division and remainder operators to extract current second, minute, and hour.

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. 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.

Example of Using Classes from the Java Library

- ➡ Objective: Demonstrate using classes from the Java library. Use the JFrame class in the javax.swing package to create two frames; use the methods in the JFrame class to set the title, size and location of the frames and to display the frames.

TestFrame

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 {
    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 {
    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();
}
```

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.

Why Data Fields Should Be private?

To protect data.

To make class easy to maintain.

Example of Data Field Encapsulation

In this example, private data are used for the radius and the accessor methods `getRadius` and `setRadius` are provided for the clients to retrieve and modify the radius.

Circle

TestCircle

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 the preceding example, 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 necessary to be 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?

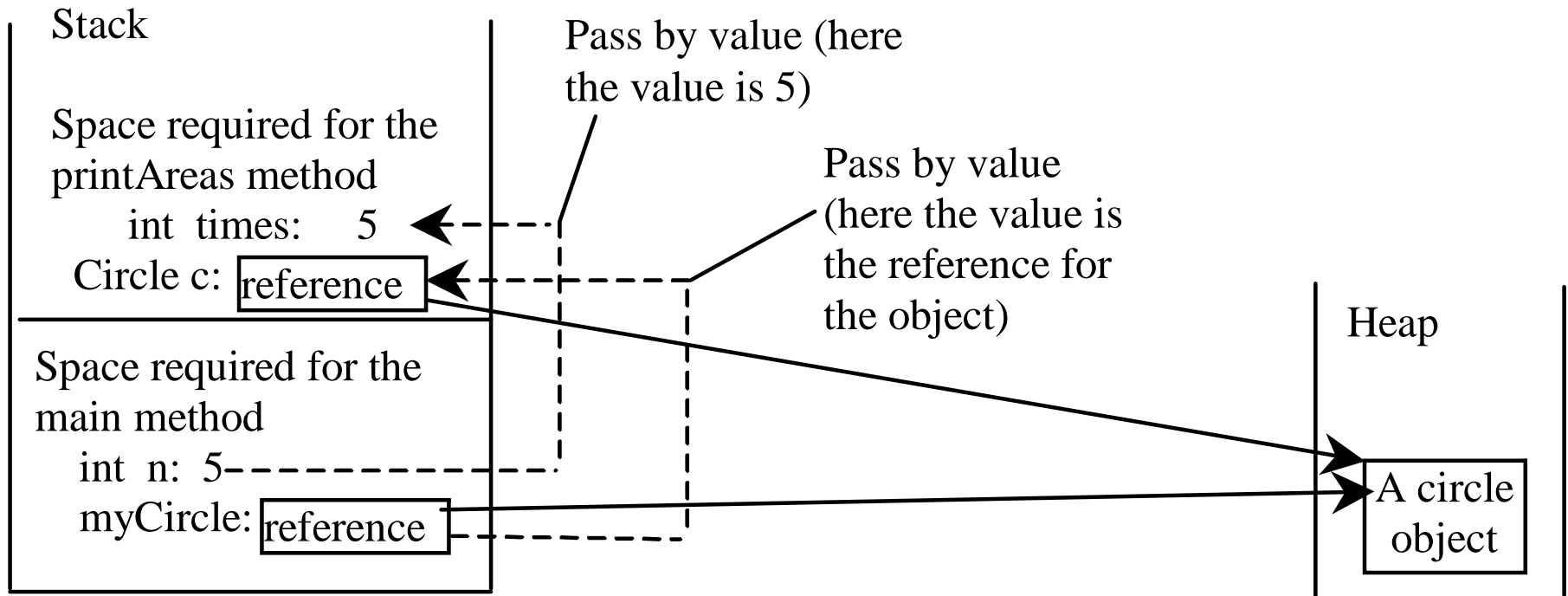
For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.

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.



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.

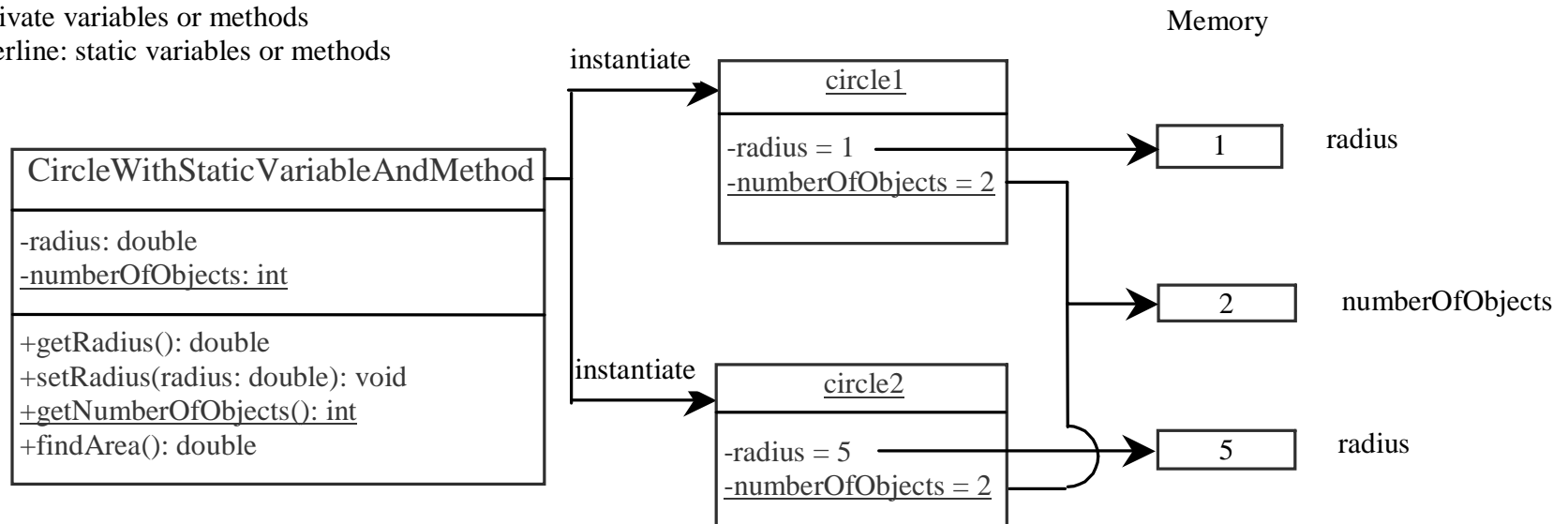
Static Variables, Constants, and Methods, cont.

UML Notation:

+: public variables or methods

-: private 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 numObjects to track the number of Circle objects created.

CircleWithStaticVariableAndMethod

TestCircleWithStaticVariableAndMethod

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

- ☞ Use `this` to refer to the object that invokes the instance method.
- ☞ Use `this` to refer to an instance data field.
- ☞ Use `this` to invoke an overloaded constructor of the same class.

Serving as Proxy to the Calling Object

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

Suppose that f1 and f2 are two objects of Foo

Invoking f1.setI(10) is to execute

→ f1.i = 10, where this is replaced by f1


Invoking f2.setI(45) is to execute

→ f2.i = 45, where this is replaced by 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 findArea() {  
        return this.radius * this.radius * Math.PI;  
    }  
}
```

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

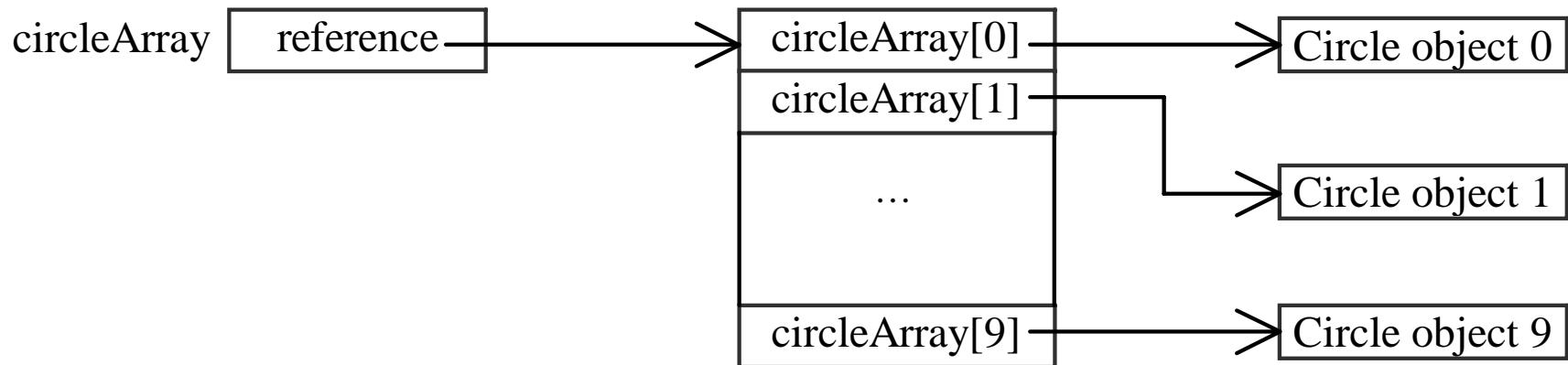
Array of Objects

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

An array of objects is actually an *array of reference variables*. So invoking `circleArray[1].findArea()` 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];
```



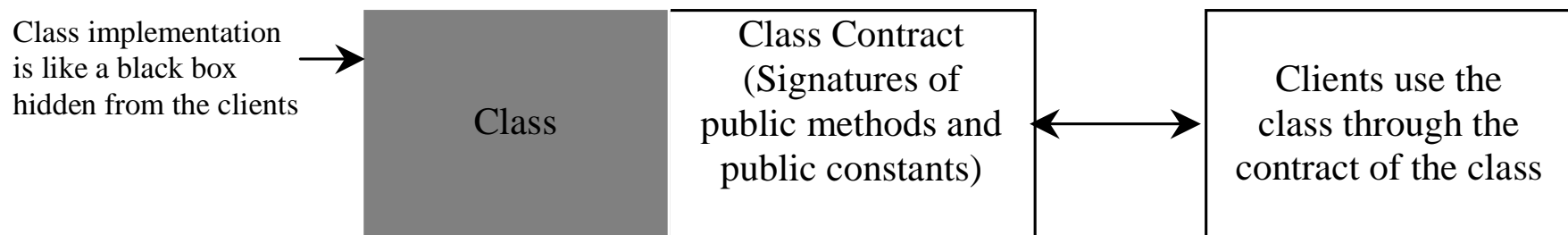
Array of Objects, cont.

Example 6.2: Summarizing the areas of the circles

TotalArea

Class Abstraction and Encapsulation

Class abstraction means to separate class implementation from the use of the class. The creator of the class provides a description of the class and let the user know how the class can be used. The user of the class does not need to know how the class is implemented. The detail of implementation is encapsulated and hidden from the user.



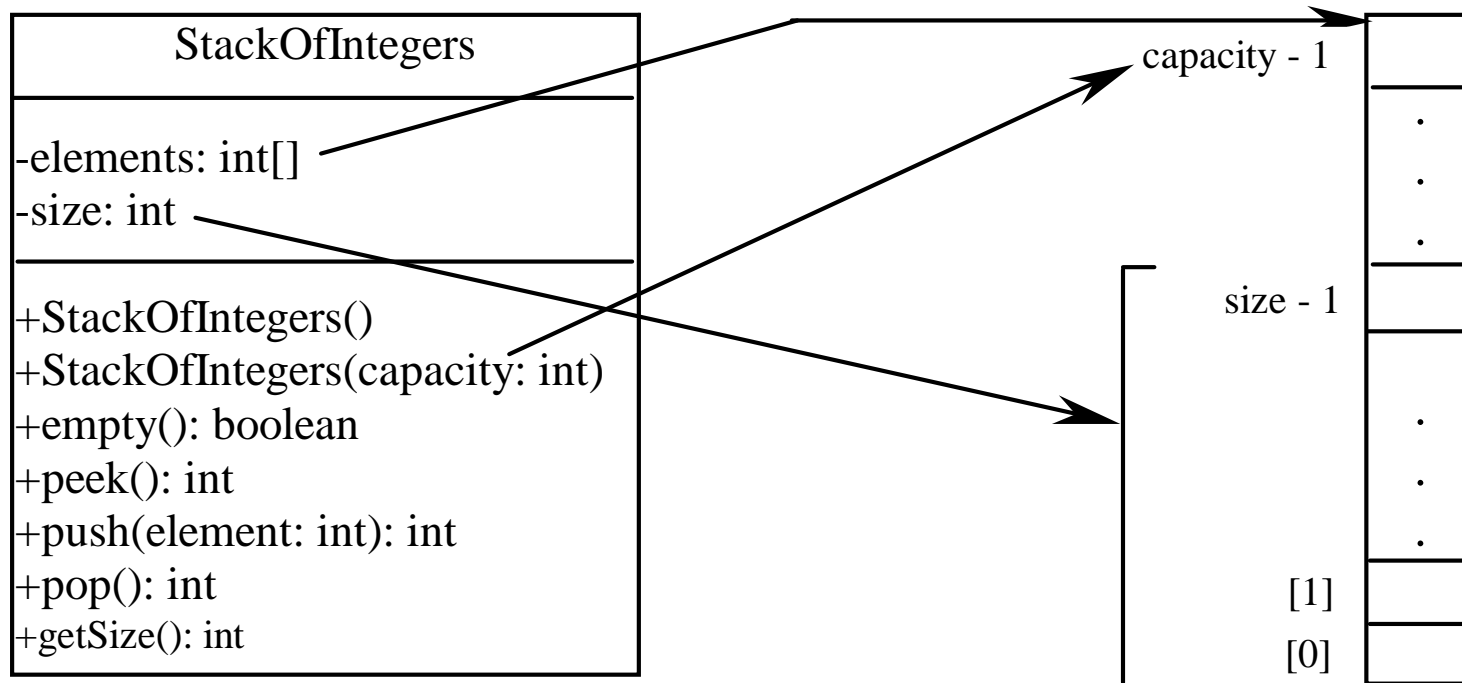
Example 6.3 The Loan Class

Loan	
-annualInterestRate: double	The annual interest rate of the loan (default: 2.5).
-numberOfYears: int	The number of years for the loan (default: 1)
-loanAmount: double	The loan amount (default: 1000)..
-loanDate: Date	The date this loan was created.
+Loan()	Constructs a default loan object.
+Loan(annualInterestRate: double, numberOfYears: int, loanAmount: double)	Constructs a loan with specified interest rate, years, and loan amount.
+getAnnualInterestRate(): double	Returns the annual interest rate of this loan.
+getNumberOfYears(): int	Returns the number of the years of this loan.
+getLoanAmount(): double	Returns the amount of this loan.
+getLoanDate(): Date	Returns the date of the creation of this loan.
+setAnnualInterestRate(annualInterestRate: double): void	Sets a new annual interest rate to this loan.
+setNumberOfYears(numberOfYears: int): void	Sets a new number of years to this loan.
+setLoanAmount(loanAmount: double): void	Sets a new amount to this loan.
+monthlyPayment(): double	Returns the monthly payment of this loan.
+totalPayment(): double	Returns the total payment of this loan.

Loan

TestLoanClass

Example 6.4 The StackOfIntegers Class



TestStackOfIntegers

Inner Classes

Inner class: A class is a member of another class.

Advantages: In some applications, you can use an inner class to make programs simple.

- ☞ An inner class can reference the data and methods defined in the outer class in which it nests, so you do not need to pass the reference of the outer class to the constructor of the inner class.

ShowInnerClass

Inner Classes (cont.)

- ☞ Inner classes can make programs simple and concise.
- ☞ An inner class supports the work of its containing outer class and is compiled into a class named *OutClassName\$InnerClassName.class*. For example, the inner class InnerClass in ShowInnerClass is compiled into *ShowInnerClass\$InnerClass.class*.

Inner Classes (cont.)

- ☞ An inner class can be declared public, protected, or private subject to the same visibility rules applied to a member of the class.
- ☞ An inner class can be declared static. A static inner class can be accessed using the outer class name. A static inner class cannot access nonstatic members of the outer class