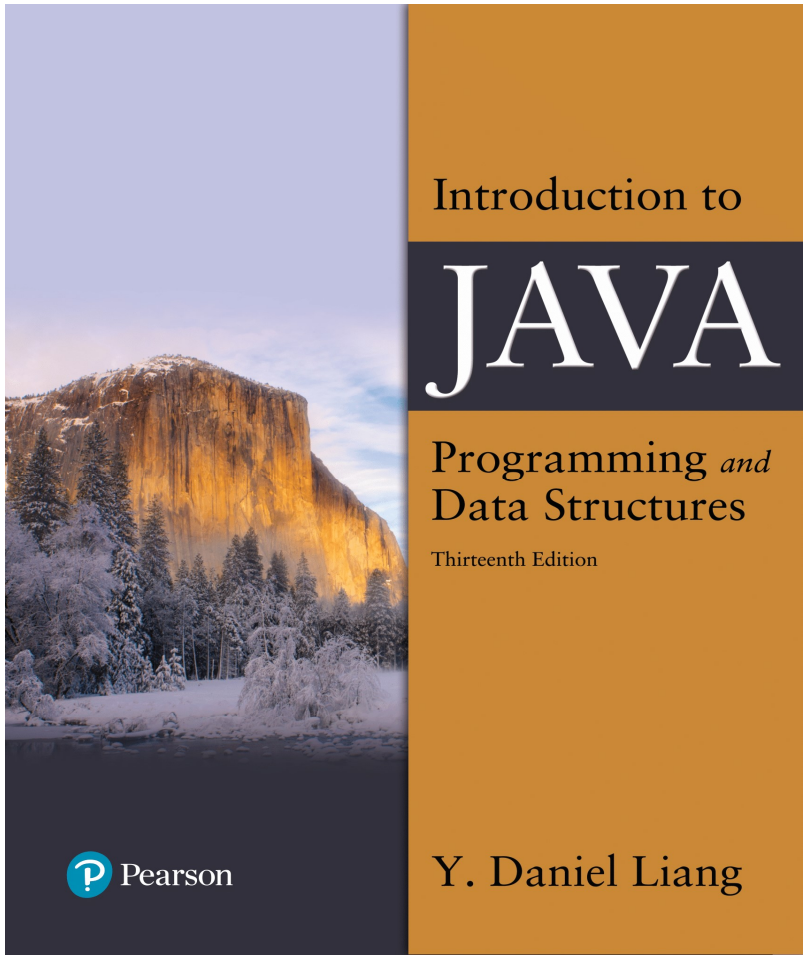
Introduction to Java Programming and Data Structures

Thirteenth Edition



Chapter 9

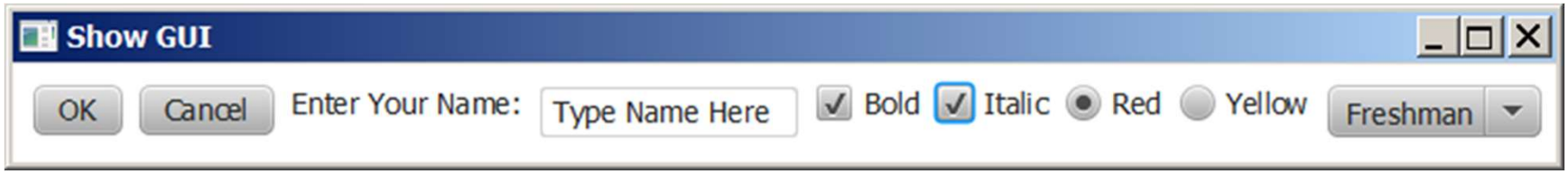
Objects and Classes

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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?



Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Objectives (1 of 2)

9.1 To describe objects and classes, and use classes to model objects (§9.2).

9.2 To use UML graphical notation to describe classes and objects (§9.2).

9.3 To demonstrate how to define classes and create objects (§9.3).

9.4 To create objects using constructors (§9.4).

9.5 To define a reference variable using a reference type and access objects via object reference variables (§9.5).

9.6 To access an object's data and methods using the object member access operator (.) (§9.5.1).

9.7 To define data fields of reference types and assign default values for an object's data fields (§9.5.2).

9.8 To distinguish between object reference variables and primitive data type variables (§9.5.3).

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Objectives (2 of 2)

9.9 To use the Java library classes **Date**, **Random**, and **Point2D** (§9.6).

9.10 To distinguish between instance and static variables and methods (§9.7).

9.11 To define private data fields with appropriate **get** and **set** methods (§9.8).

9.12 To encapsulate data fields to make classes easy to maintain (§9.9).

9.13 To develop methods with object arguments and differentiate between primitive-type arguments and object-type arguments (§9.10).

9.14 To store and process objects in arrays (§9.11).

9.15 To create immutable objects from immutable classes to protect the contents of objects (§9.12).

9.16 To determine the scope of variables in the context of a class (§9.13).

9.17 To use the keyword **this** to refer to the calling object itself (§9.14).

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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.

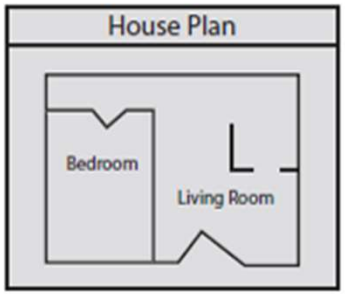
Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Classes and Instances

- Many objects can be created from a class.
- Each object is independent of the others.

Blueprint that describes a house

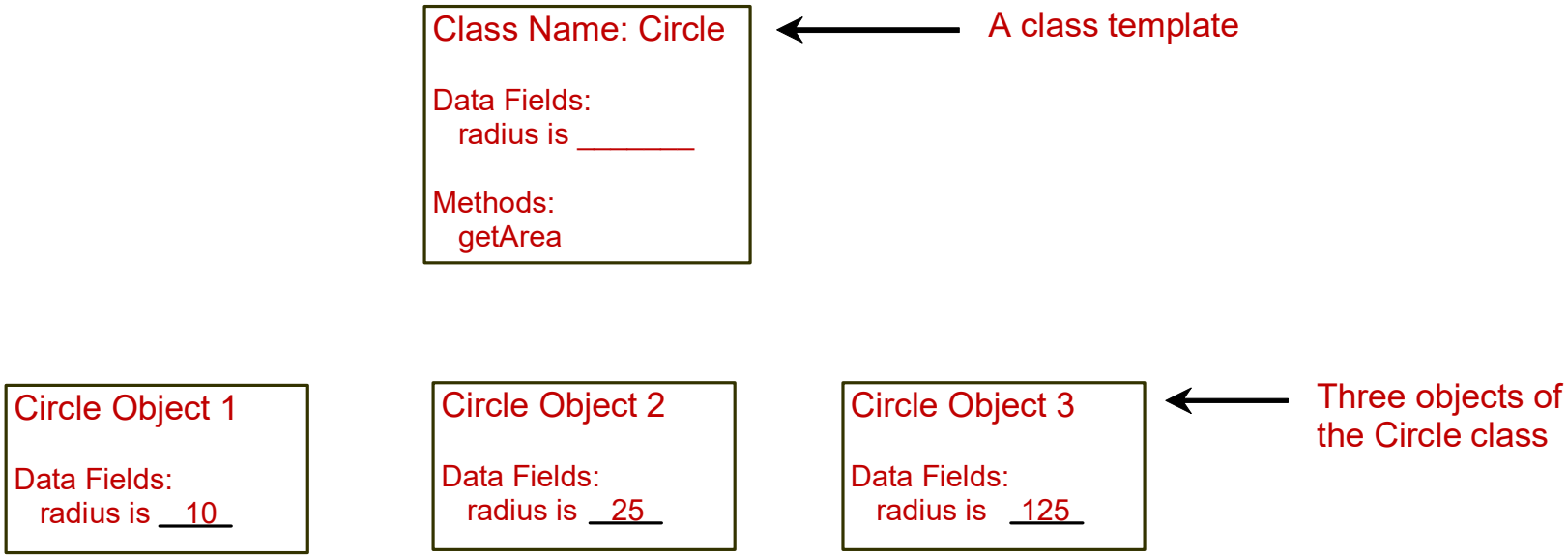


Instances of the house described by the blueprint

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Objects



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

Pearson

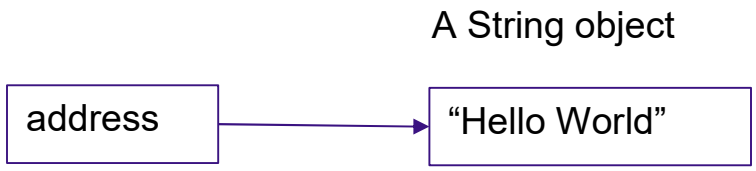
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Objects and Programs

– For example, the `String` class provides methods:
  - Examples: `length()` and `charAt()` methods

```
String greeting = new String
   (“Hello World”);
int len = greeting.length();
char c1 = greeting.charAt(0);
```

The *greeting* variable Holds the address of a String object.



Pearson
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

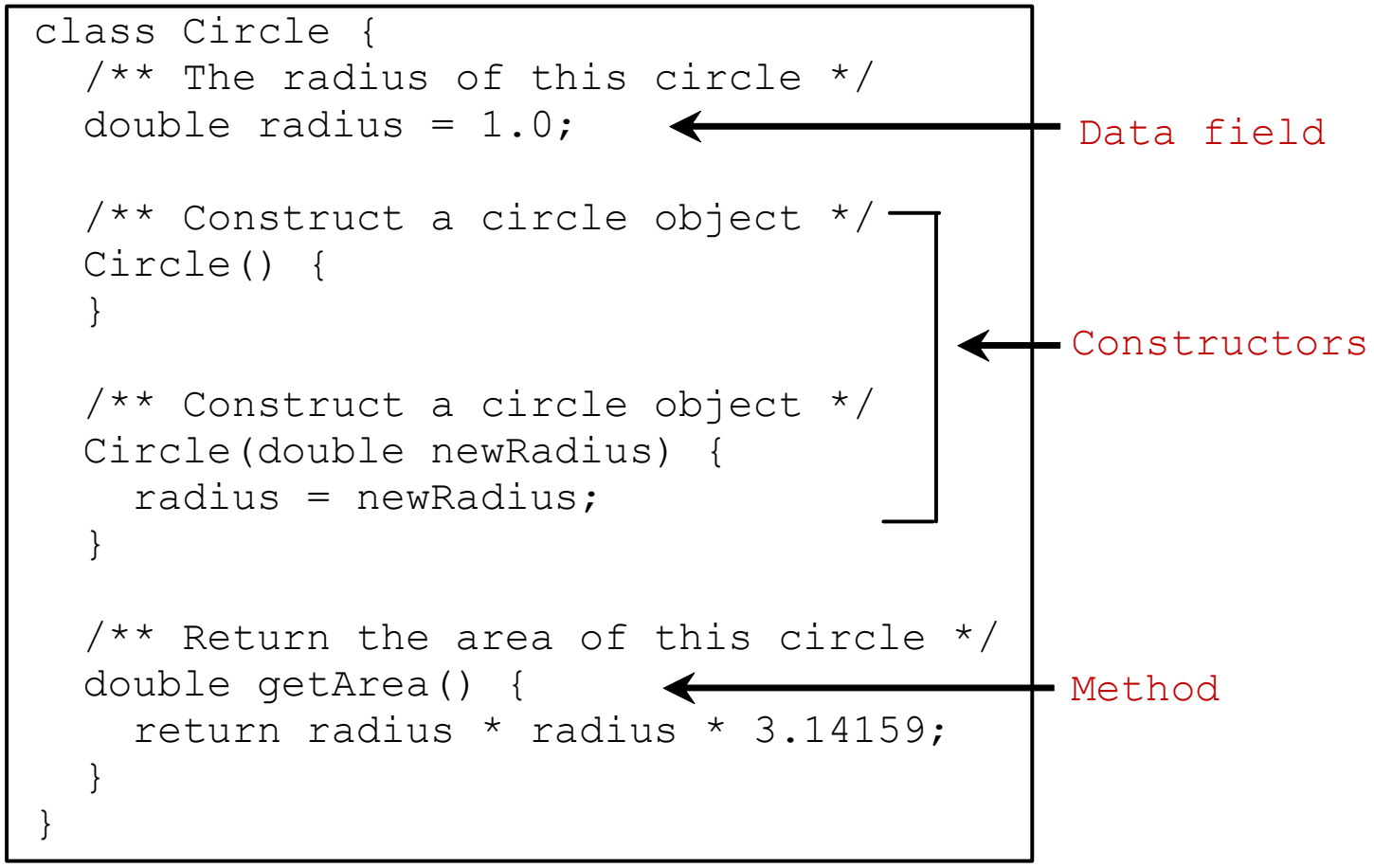
Classes (1 of 2)

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.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Classes (2 of 2)

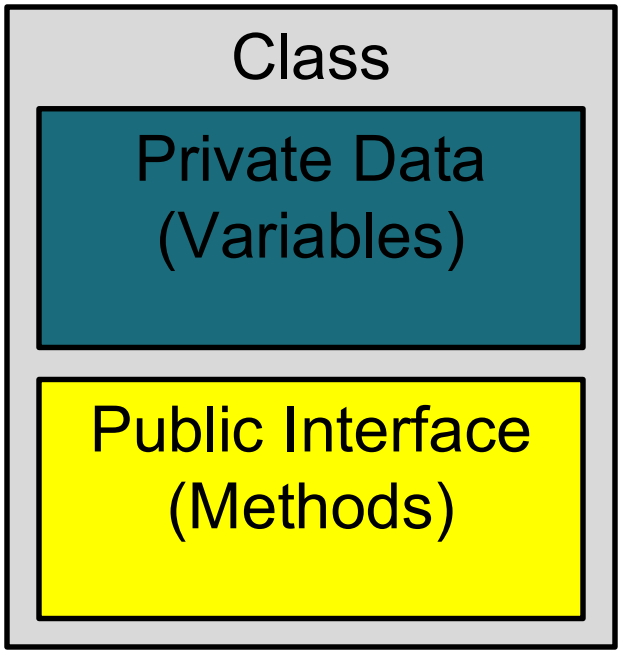


Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

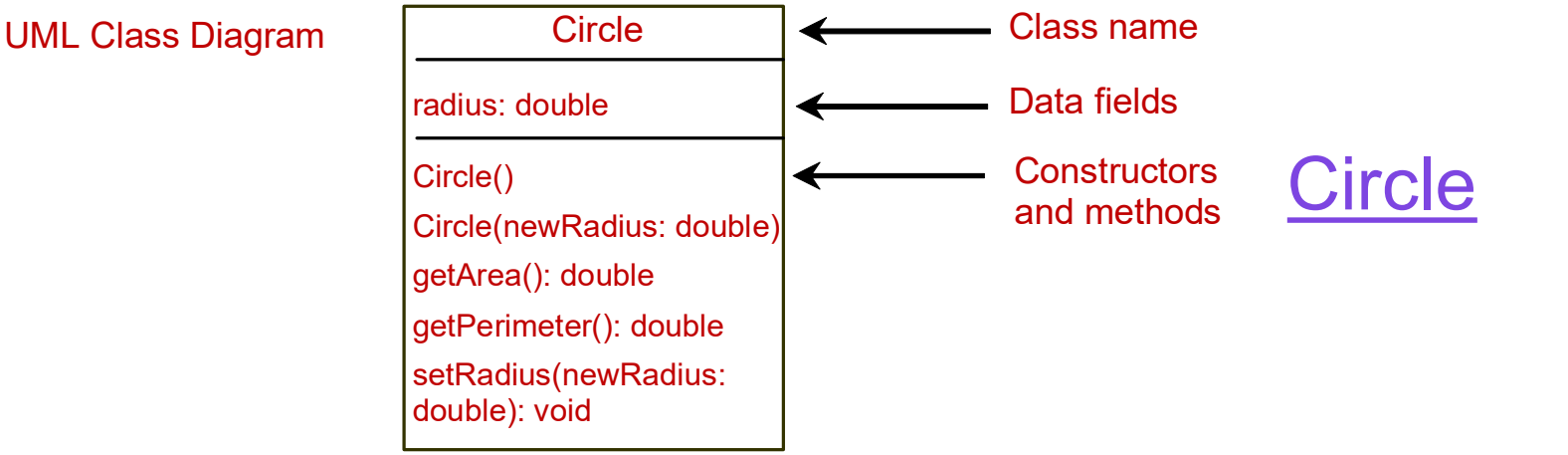
Diagram of a Class

- Private Data
  - Each object has its own private data that other objects cannot directly access
  - Methods of the public interface provide access to private data, while hiding implementation details:
  - This is called Encapsulation
- Public Interface
  - Each object has a set of methods available for other objects to use



Copyright © 2024 Pearson Education, Inc. All Rights Reserved

UML Class Diagram



TestCircle

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Constructors

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

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.
- **`Example:  new ClassName();`**

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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**.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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:

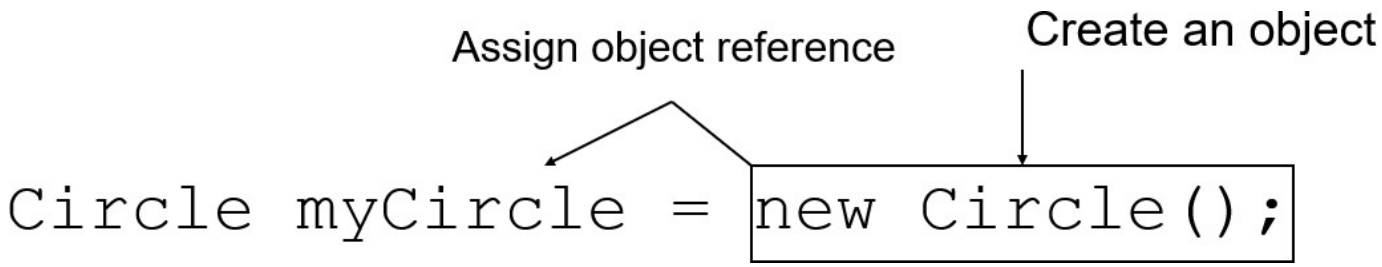
```
Circle myCircle;
```


Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Declaring/Creating Objects in a Single Step

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

Example:



Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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()`

Pearson

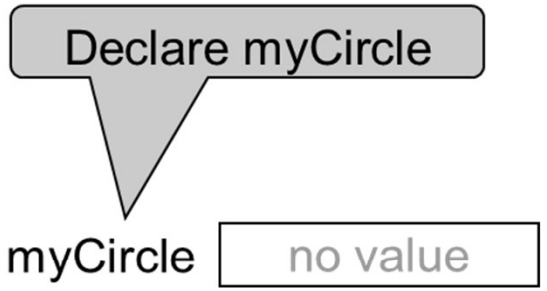
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Trace Code (1 of 7)

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Pearson

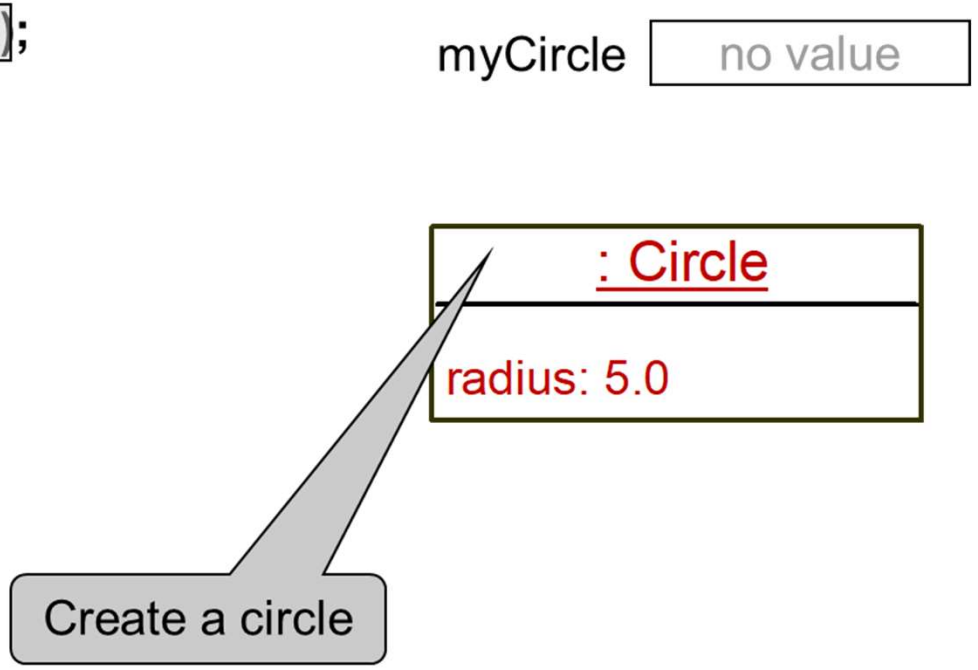
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Trace Code (2 of 7)

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Pearson

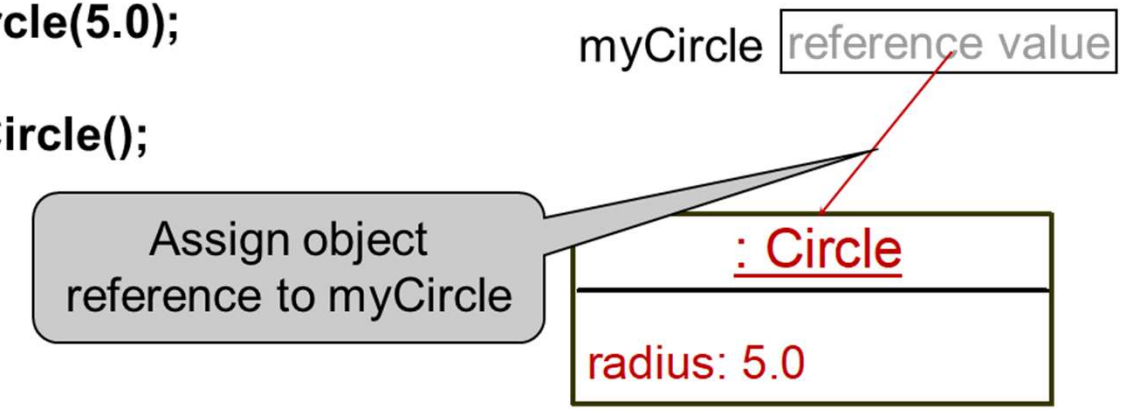
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Trace Code (3 of 7)

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



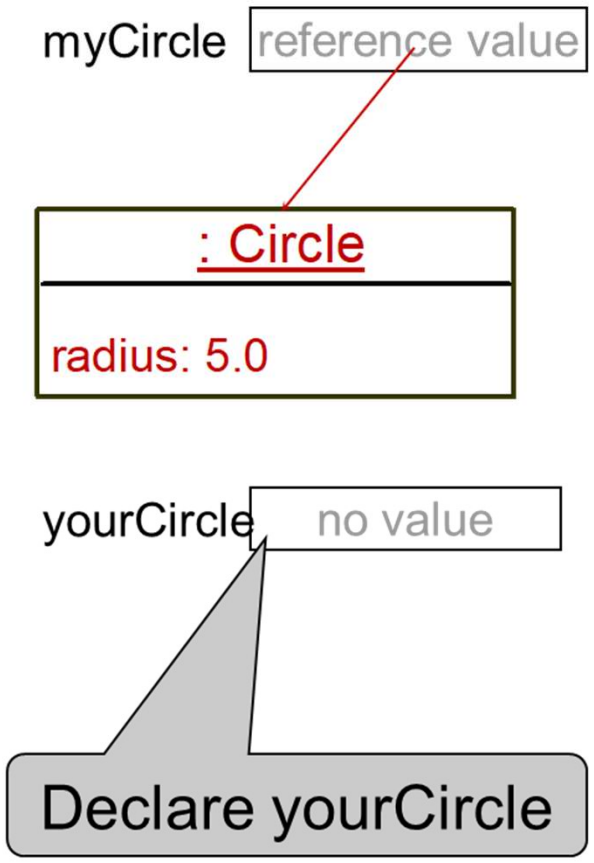
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Trace Code (4 of 7)

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Pearson

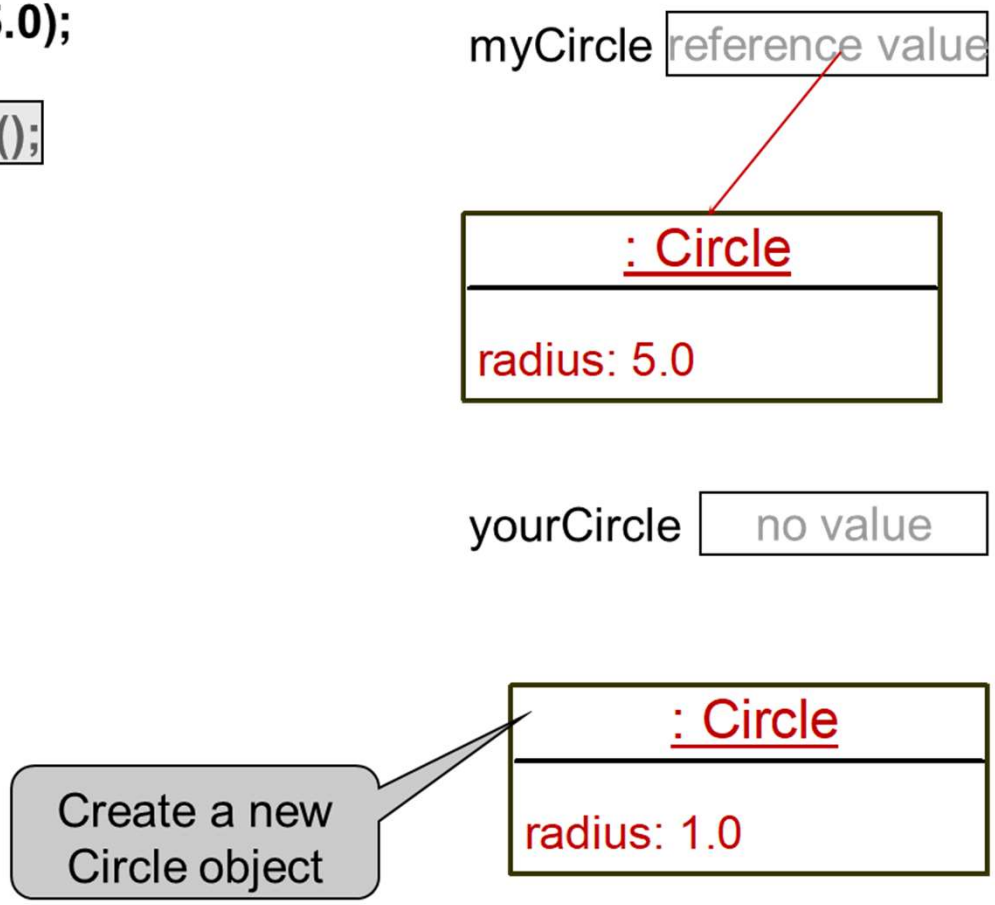
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Trace Code (5 of 7)

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Pearson

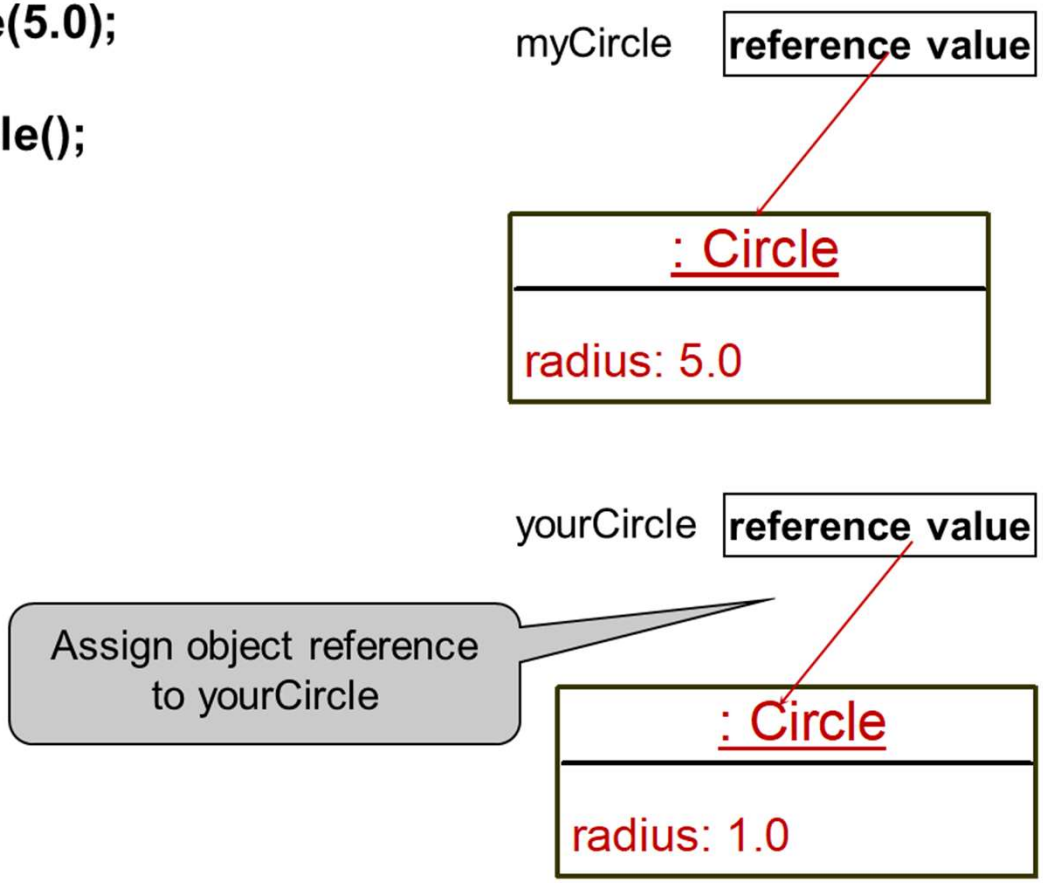
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Trace Code (6 of 7)

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Pearson

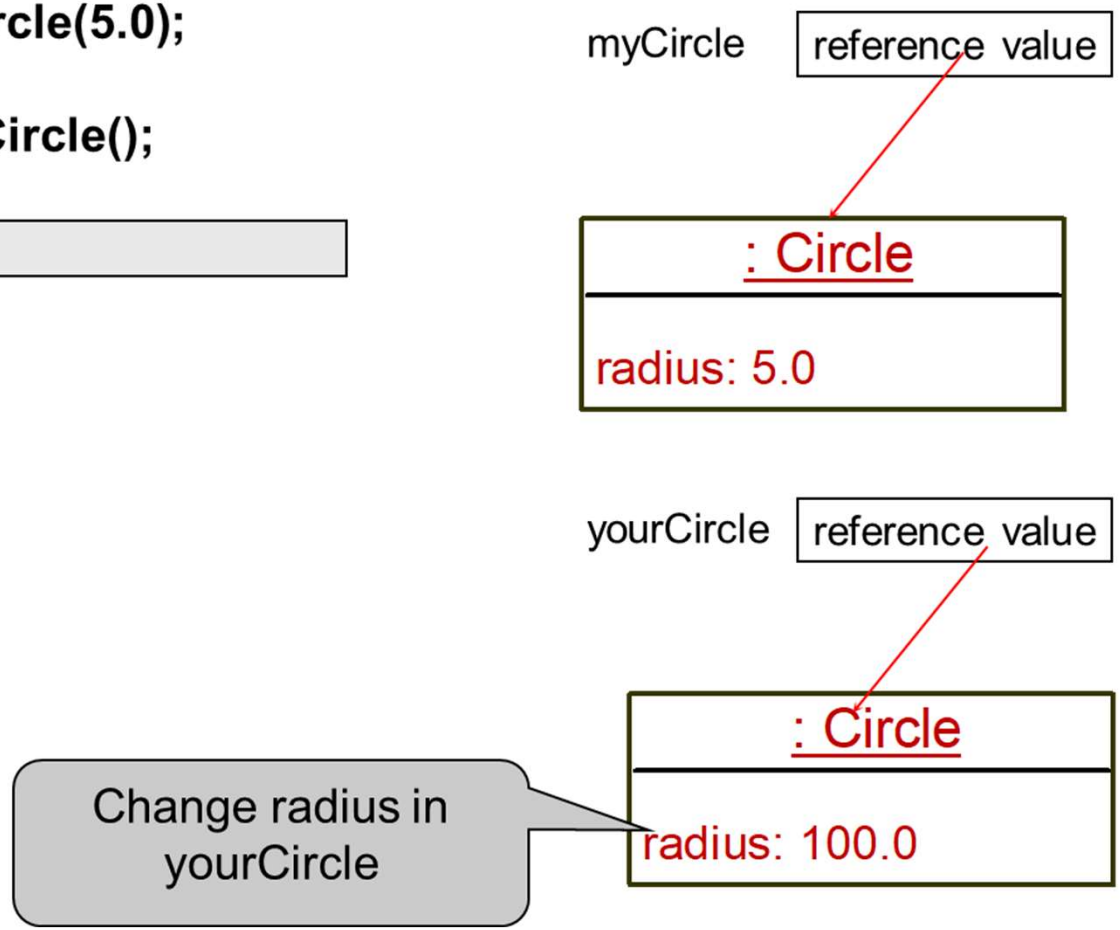
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Trace Code (7 of 7)

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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.”

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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'
}
```

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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);
  }
}
```

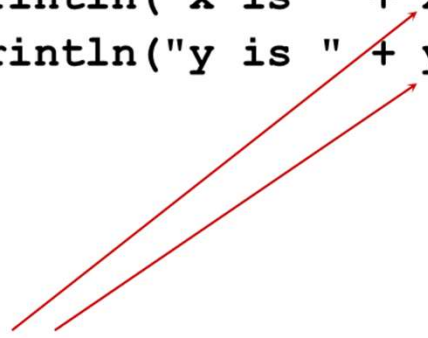
Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Example (1 of 3)

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);
  }
}
```

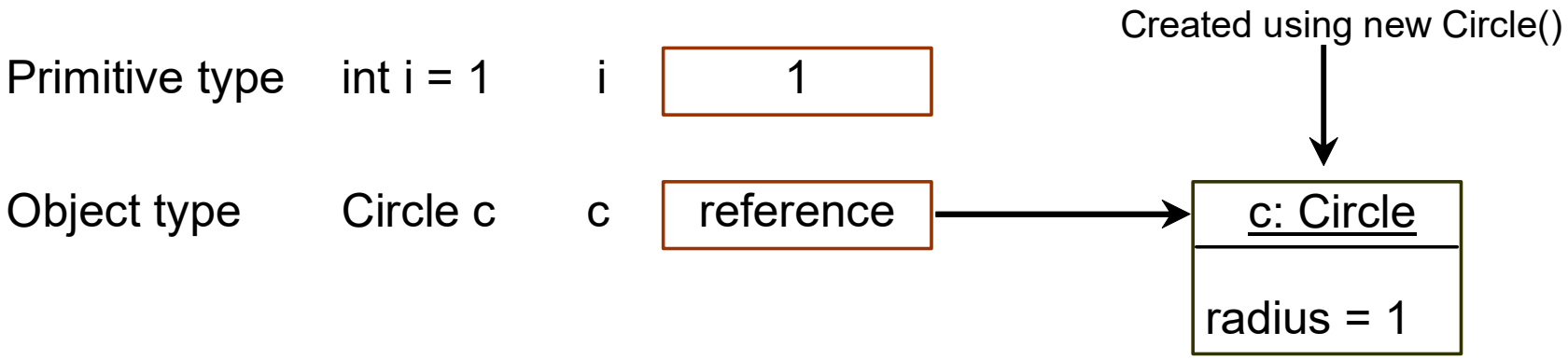


Compile error: variable not initialized

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Differences between Variables of Primitive Data Types and Object Types



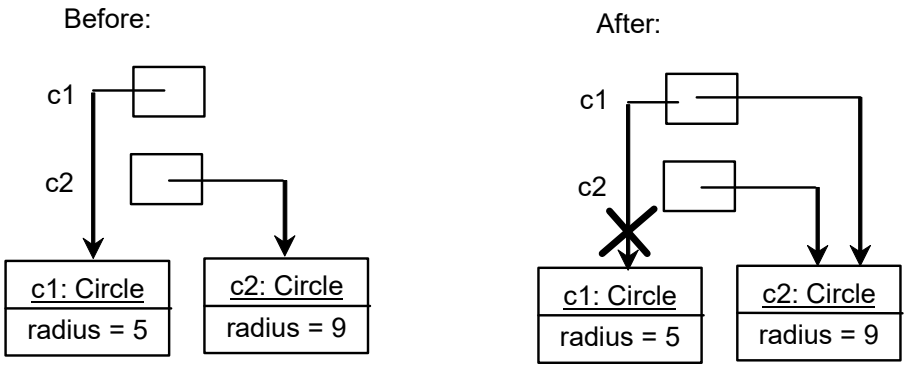
Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Copying Variables of Primitive Data Types and Object Types

Primitive type assignment i = j

Object type assignment c1 = c2



Pearson

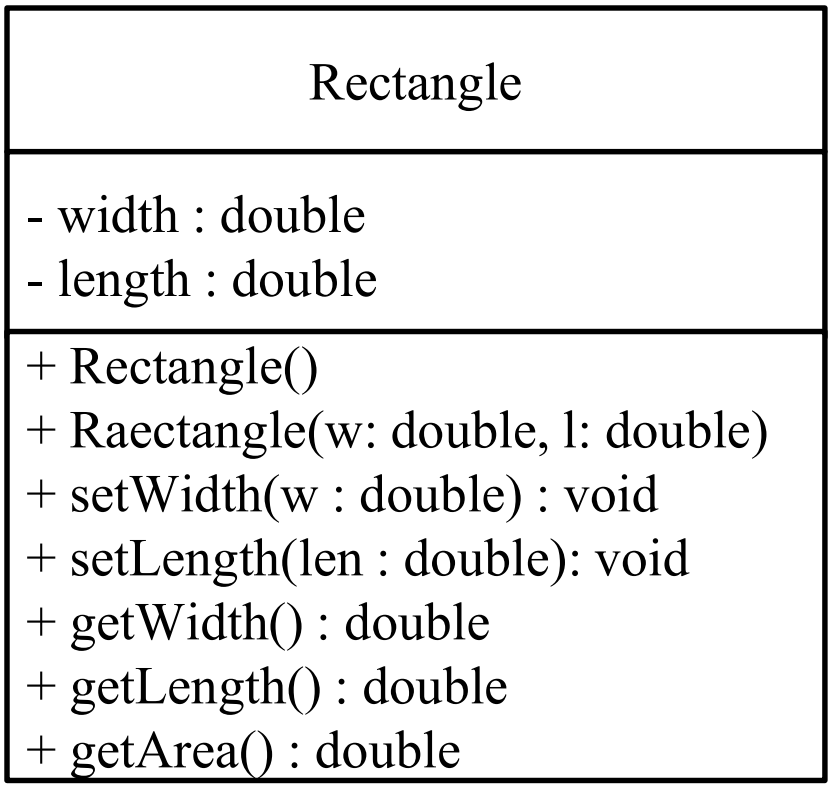
Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Rectangle Class

- When designing a class, decisions about the following must be made.
  - what data must be accounted for
  - what actions need to be performed
  - what data can be modified
  - what data needs to be accessible
  - any rules as to how data should be modified
- Class design typically is done with the aid of a Unified Modeling Language (UML) diagram.

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

The UML Diagram for Rectangle class



Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

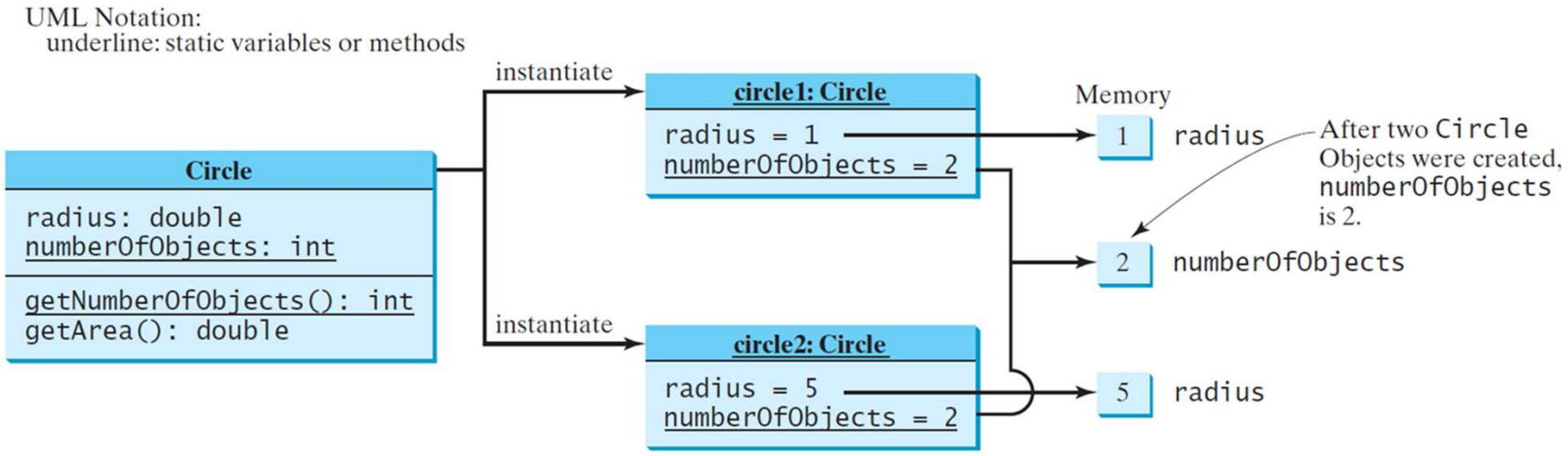
Instance Variables, and Methods

- Instance variables belong to a specific instance.
- Instance methods are invoked by an instance of the class.
- 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.
- To declare static variables, constants, and methods, use the static modifier.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Static Variables, Constants, and Methods



Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Visibility Modifiers and Accessor/Mutator Methods (1 of 3)

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.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Visibility Modifiers and Accessor/Mutator Methods (2 of 3)

```
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();
  }
}
```

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.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Visibility Modifiers and Accessor/Mutator Methods (3 of 3)

```
package p1;

class C1 {
  ...
}
```

```
package p1;

public class C2 {
  can access C1
}
```

```
package p2;

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

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

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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).

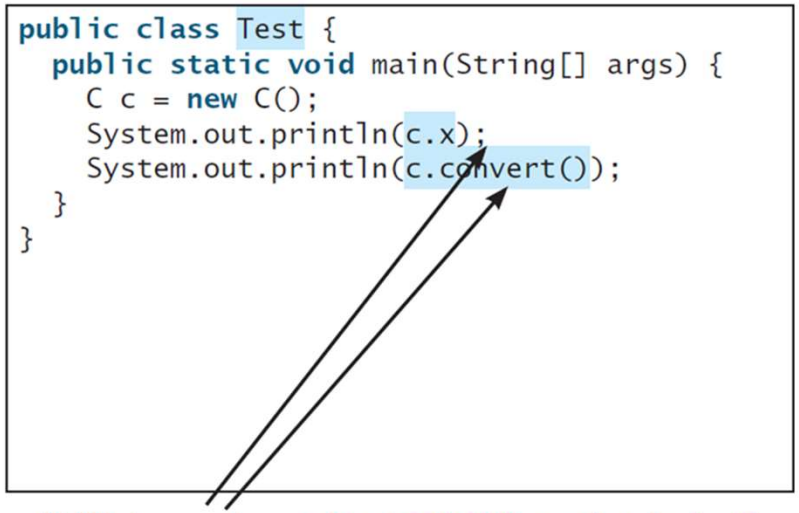
```
public class C {
  private boolean x;

  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());
  }
}
```

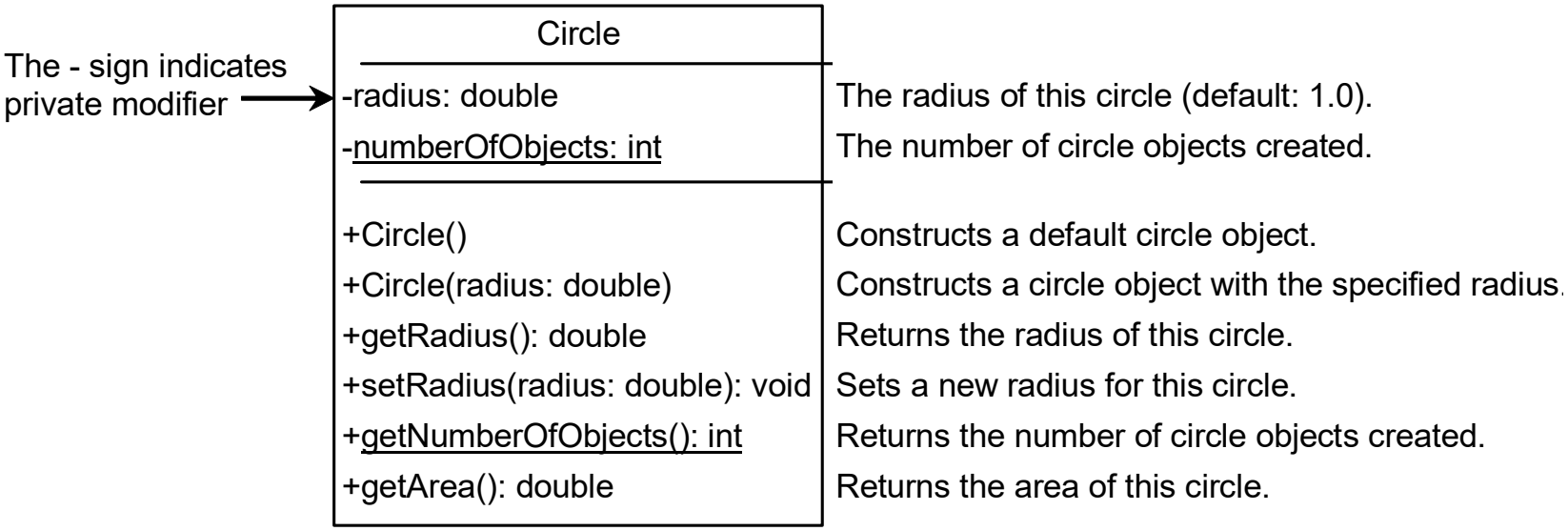


(b) This is wrong because `x` and `convert` are private in class `C`.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Example of Data Field Encapsulation



CircleWithPrivateDataFields

TestCircleWithPrivateDataFields

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Passing Objects to Methods (1 of 2)

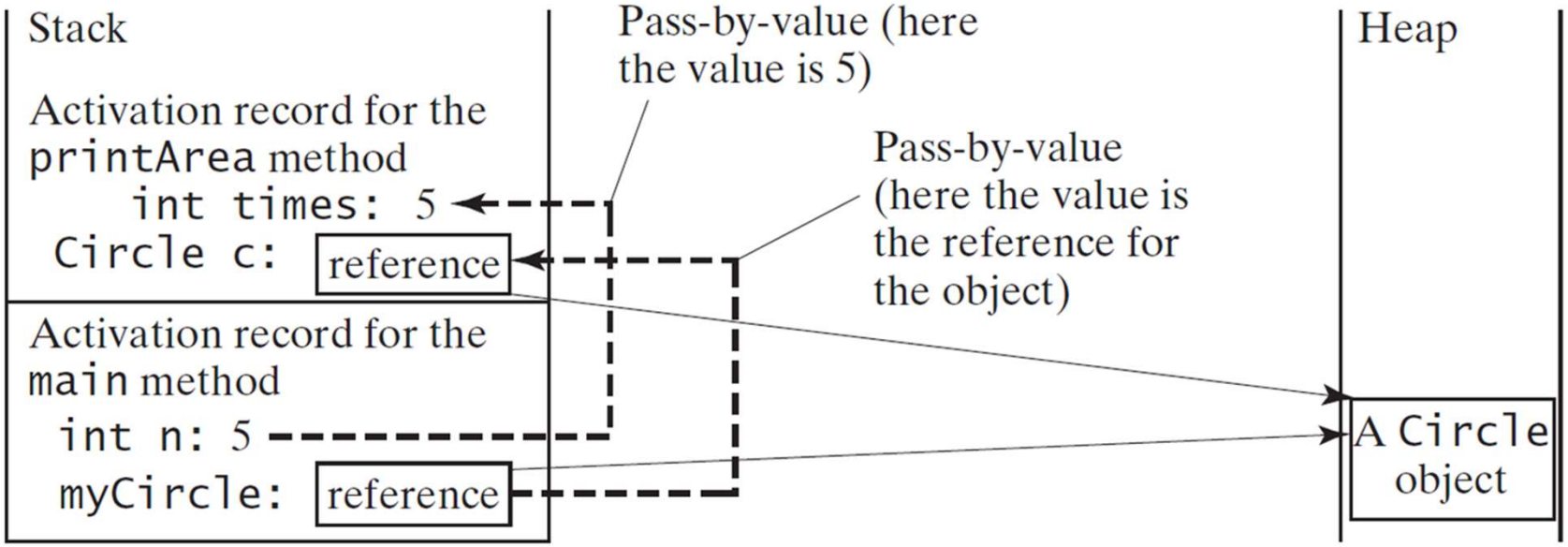
- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)

TestPassObject

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Passing Objects to Methods (2 of 2)



Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Array of Objects (1 of 3)

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

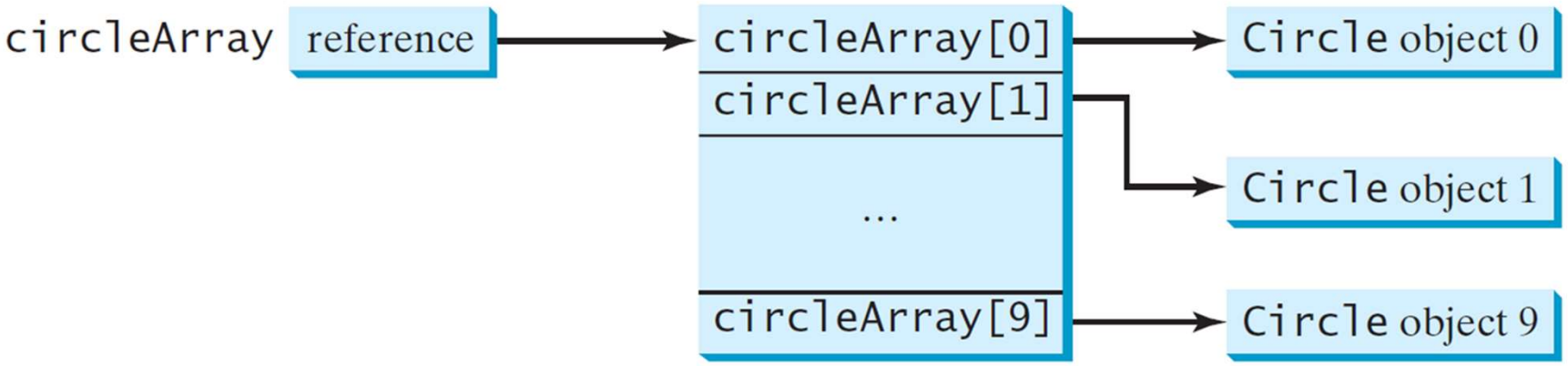
An array of objects is actually an **array of reference variables**. So invoking circleArray[1].getArea() involves two levels of referencing as shown in the next figure. circleArray references to the entire array. circleArray[1] references to a Circle object.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Array of Objects (2 of 3)

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



Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Array of Objects Examples (3 of 3)

- Summarizing the areas of the circles
- A circle with the largest area
- Selection Sort using Circle objects (sort by radius)

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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 Listing 8.10, 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.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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.

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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.

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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.

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

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
```

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Calling Overloaded Constructor

```
public class Circle {
  private double radius;

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

  public Circle() {
    this(1.0);
  }

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

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

this is used to invoke another constructor

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

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

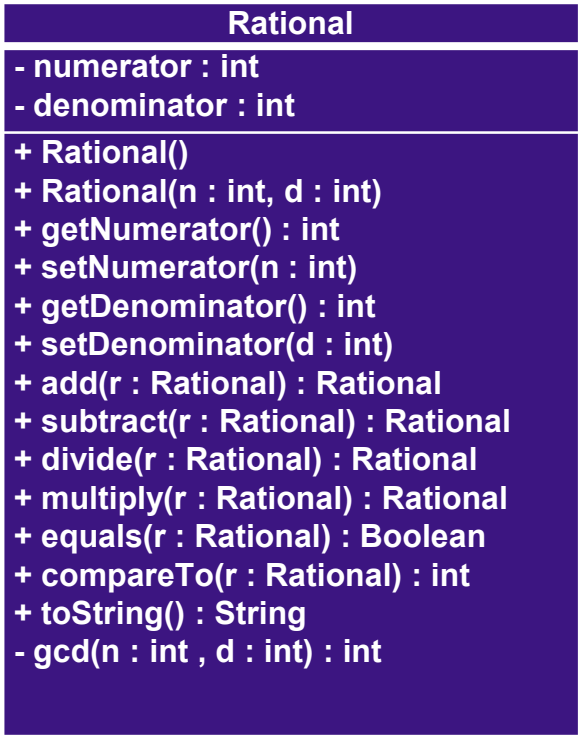
Objects & Classes

Rational Class

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

Rational Numbers UML



default value for a Rational object is 0/1
Rational objects must be in reduced form
Rational objects should not be modified by any of the methods
toString() format is "numerator/denominator"

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

- Given n, write a program that will compute the following summation series using the Rational class. output should be as follows:

$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ = result

Pearson

Copyright © 2024 Pearson Education, Inc. All Rights Reserved

- Given n Rational numbers, find the maximum

Copyright © 2024 Pearson Education, Inc. All Rights Reserved