CS112

Objects and Classes (Part 3)

Lecture 04

Spring 2022 - 1443

College of Computer Science and Engineering

جامعة طيبة

Visibility Modifiers and Accessor/Mutator Methods (1)

- 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.

Visibility Modifiers and Accessor/Mutator Methods (2)

- The private modifier restricts access to within a class
- The default modifier restricts access to within a package
- The public modifier enables unrestricted access

```java
package p1;

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

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

```java
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();
  }
}
```

```java
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();
  }
}
```

```java
package p1;

class C1 {
  ...
}
```

```java
package p1;

public class C2 {
  can access C1
}
```

```java
package p2;

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

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).

```java
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**.

```java
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 1 (1)

The - sign indicates private modifier

Circle	
-radius: double	The radius of this circle (default: 1.0).
-numberOfObjects: int	The number of circle objects created.
+Circle()	Constructs a default circle object.
+Circle(radius: double)	Constructs a circle object with the specified radius.
+getRadius(): double	Returns the radius of this circle.
+setRadius(radius: double): void	Sets a new radius for this circle.
+getNumberOfObjects(): int	Returns the number of circle objects created.
+getArea(): double	Returns the area of this circle.

Example 1 (2)

```
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;
  }
}
```

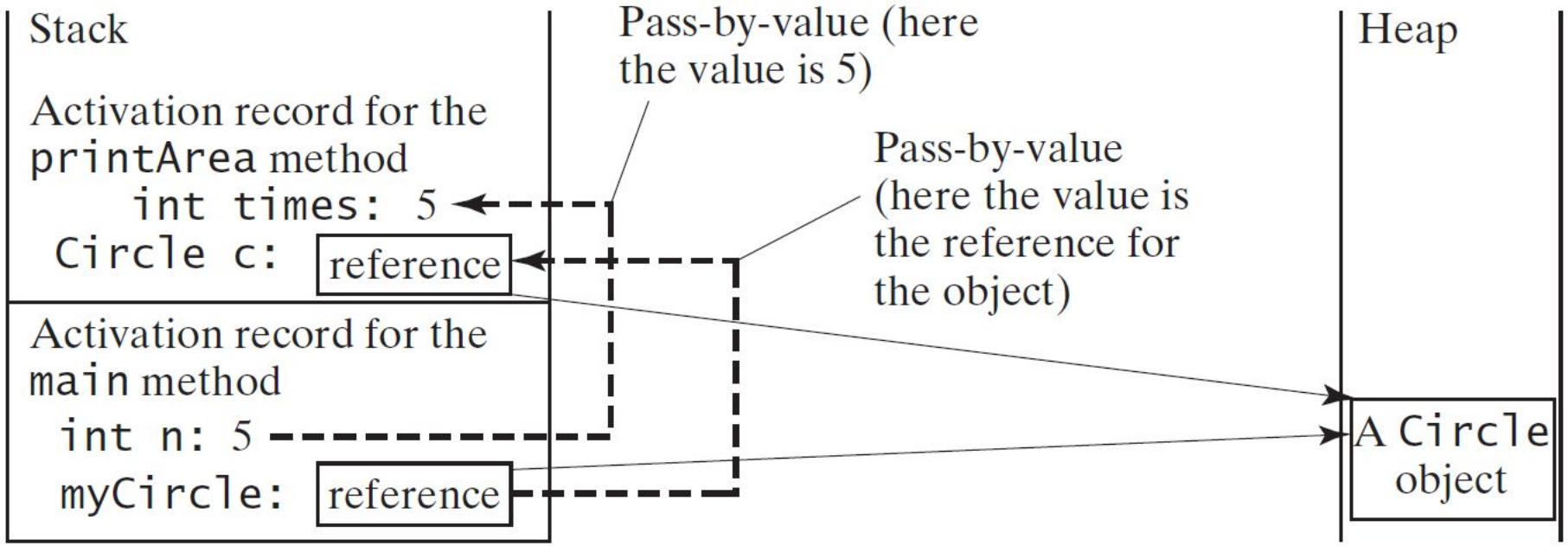
Example 1 (3)

```
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

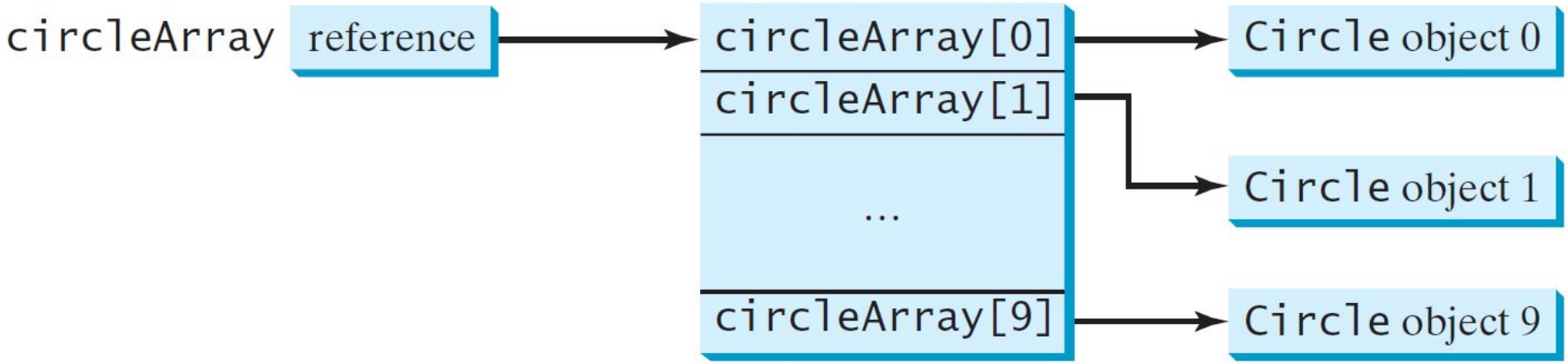
- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)



Array of Objects

- An array of objects is actually an *array of reference variables*.
- So, invoking circleArray[1].getArea() involves two levels of referencing:

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



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

Example 2

```java
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;
  }

  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