CLASS & OBJECT

ITX 2001, CSX 3002, IT 2371

CLASS & OBJECT

- To understand objects and classes and use classes to model objects
- To learn how to declare a class and how to create an object of a class
- To understand the roles of constructors and use constructors to create objects
- To use UML graphical notations to describe classes and objects

OBJECT-ORIENTED PROGRAMMING CONCEPT

- Object-oriented programming (OOP) involves programming using objects.
- An object represents an entity in the real world that can be distinctly identified.

OBJECT-ORIENTED PROGRAMMING CONCEPT

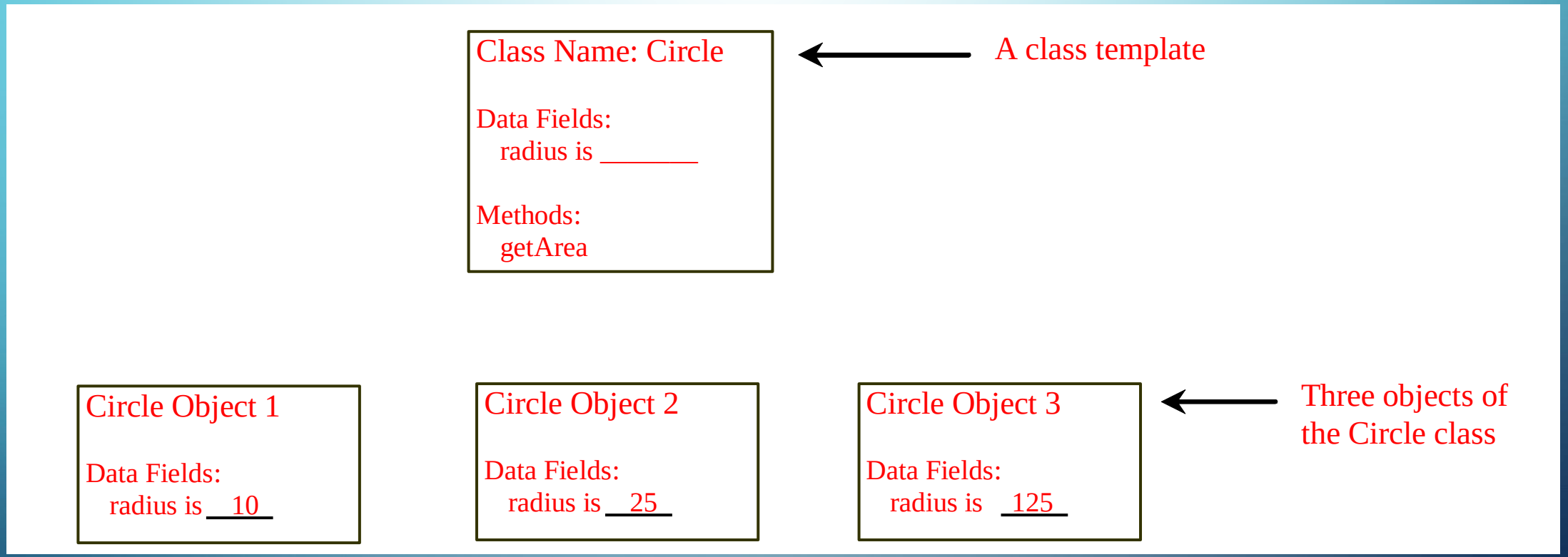
Ex: a student, a desk, a circle, a button, and even a loan can all be viewed as objects.

OBJECT-ORIENTED PROGRAMMING CONCEPT

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 state defines the object
- The behavior of an object defined by a set of methods.
  - The behavior defines what the object does.

OBJECTS

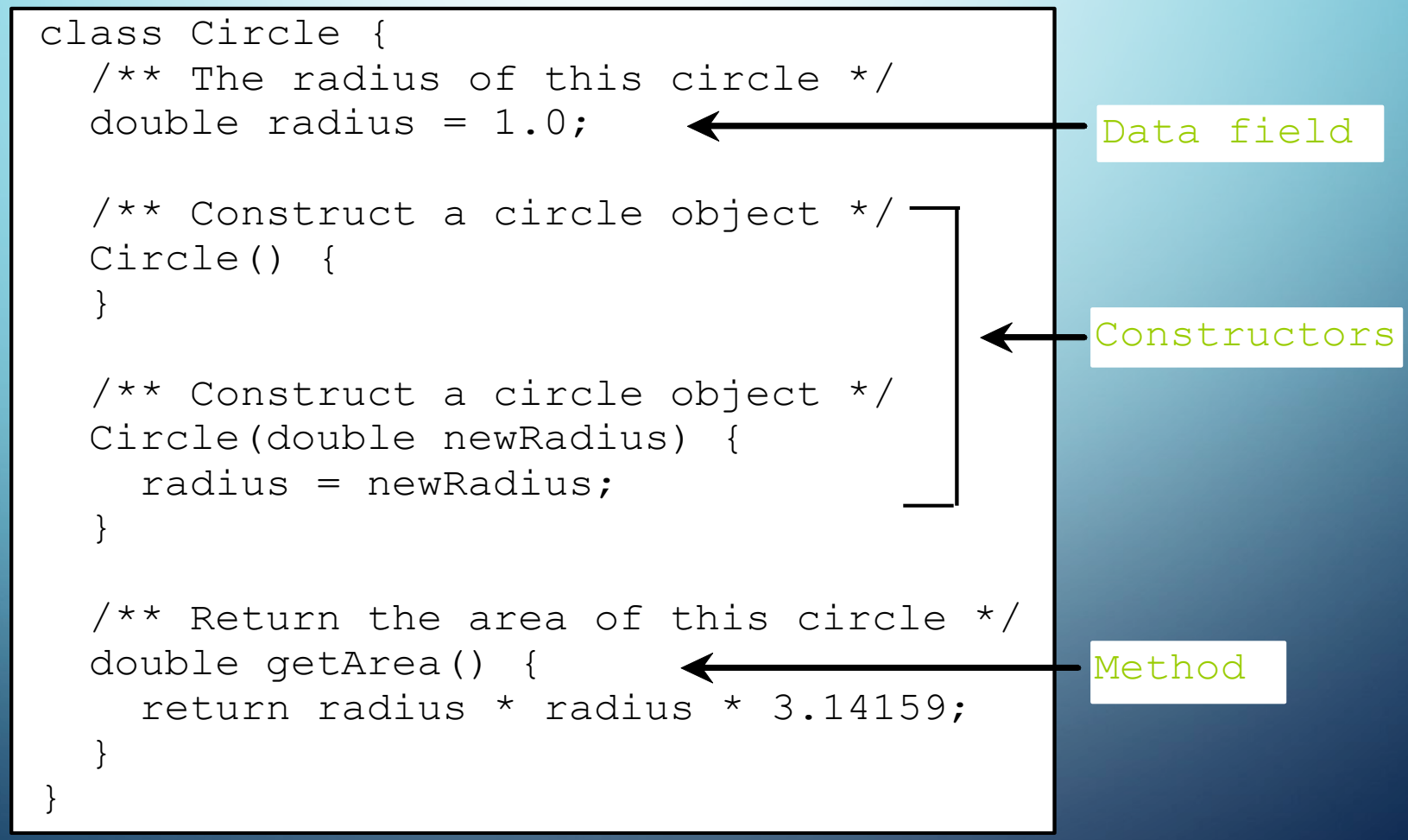

Class Name: Circle
Data Fields:
radius is _____
Methods:
getArea
A class template
Circle Object 1
Data Fields:
radius is 10
Circle Object 2
Data Fields:
radius is 25
Circle Object 3
Data Fields:
radius is 125
Three objects of the Circle class

CLASS

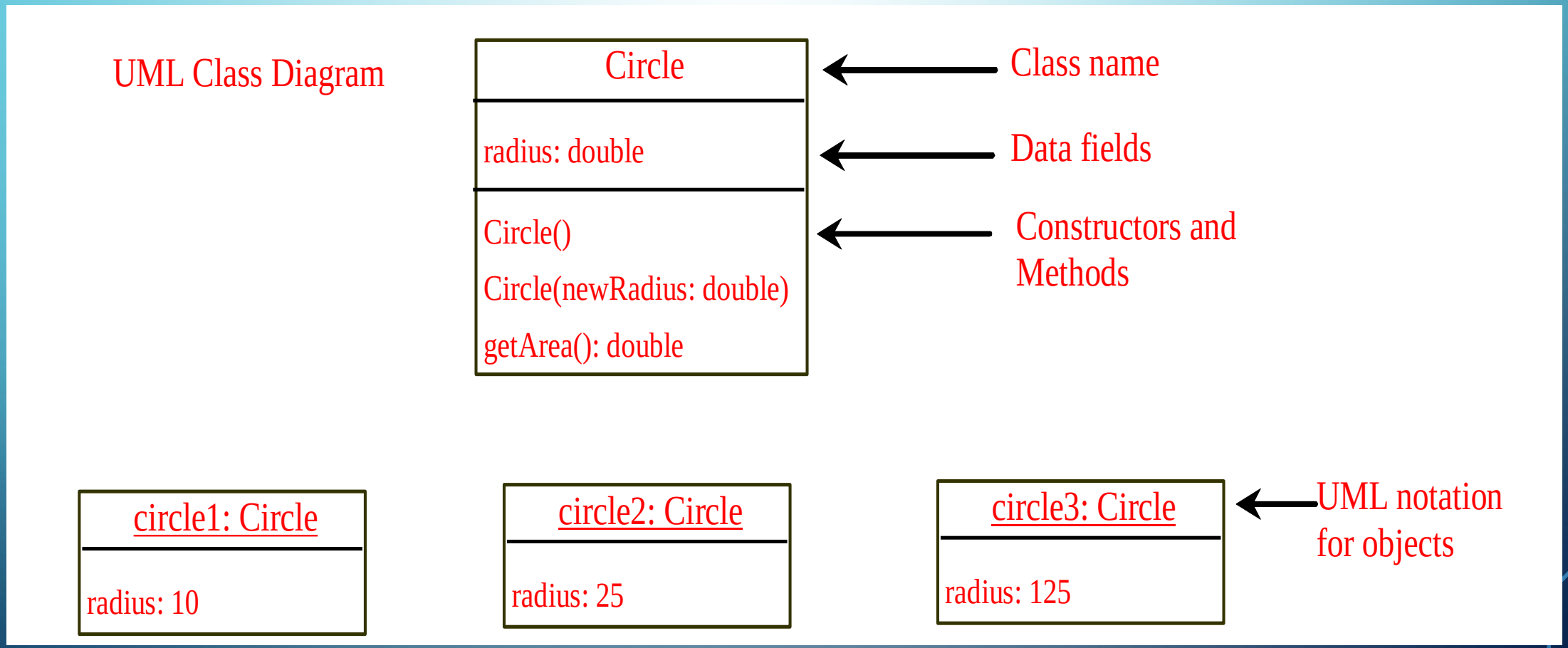
- *Classes* are constructs that define objects of the same type.
- Class member are
  - Attributes / variables to define data fields and
  - Methods to define behaviors.
  - Constructor: A class provides a special type of methods, known as constructors, which are invoked to construct objects from the class.

CLASS SPECIFICATION

```
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;
  }
}
```



UML CLASS DIAGRAM



VARIABLES

- Class Variables
  - Variables used in the class context
  - Attributes that are shared by all created objects
- Instance Variables
  - Attributes that are distinct to individual objects

CLASS'S FIELDS / ATTRIBUTES

- **Variables declared int the class body**

```
Syntax: type fieldName;


class CheckingAccount
{
  String owner; // name of person who owns
                // this checking account
  int balance;  // amount of money that can be withdrawn
}
```

CLASS'S FIELDS / ATTRIBUTES

- Field could be of array types.

```
class WeatherData
{
  String country;
  String[] cities;
  double[][] temperatures;
}
```

CLASS'S FIELD / ATTRIBUTE

```
class WeatherData
{
  String country = "United States";
  String[] cities = {"Chicago",
                     "New York",
                     "Los Angeles"};
  double[][] temperatures = {{0.0, 0.0},
                             {0.0, 0.0},
                             {0.0, 0.0}};
```

It is not necessary to declare fields at the beginning of the class definition, although this is the most common practice.

It is only necessary that they be declared and initialized before they are used.

STATIC FIELD

- Sometimes you need a single copy of a field no matter how many objects are created.
- Just create a static field.
- It is a field that associates with a class instead of with that class's objects.

STATIC FIELD USAGE

- You introduce a counter field into this class.
- What's the proper initial value?
- You also place code in the class's constructor to increase counter's value by 1 when an object is created.
- However, because each object has its own copy of the counter field, this field's value never advances past 1.

READ-ONLY FIELD

- Specified by a reserved word “final”

```
class Employee {
   final static int RETIREMENT_AGE =
65;
}
```

CONSTRUCTOR

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

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

CONSTRUCTORS

- Constructors must have the same name as the class itself.
- Constructors play the role of initializing objects.
- A constructor with no parameters is referred to as a *no-argument constructor*.
- Constructors do not have a return type—not even void.
- Constructors are invoked using the new operator when an object is created.

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 non-argument 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.

DECLARING OBJECT REFERENCE VARIABLES

- To reference an object, assign the object to a reference variable by using the syntax:

  ClassName objectRefVar;

- Example:

  Circle myCircle;

DECLARING/CREATING OBJECTS IN A SINGLE STEP

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

Example:

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

ACCESSING OBJECTS

- Referencing the object's data:

  objectRefVar.data

  e.g., myCircle.radius

- Invoking the object's method:

  objectRefVar.methodName(arguments)

  e.g., myCircle.getArea()

;

A Simple Circle Class

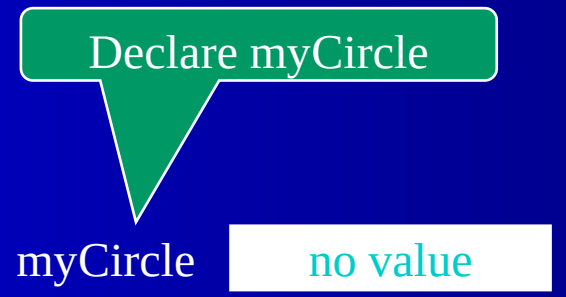
Objective: Demonstrate creating objects, accessing data, and using methods.

TestCircle1

animation

Trace Code

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

SCircle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

myCircle | no value

: Circle

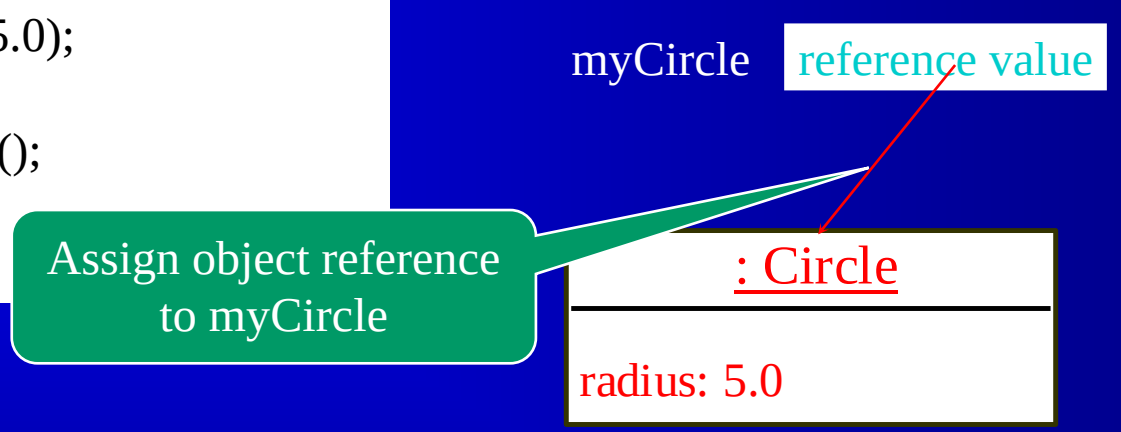
radius: 5.0

Create a circle

animation

Trace Code, cont.

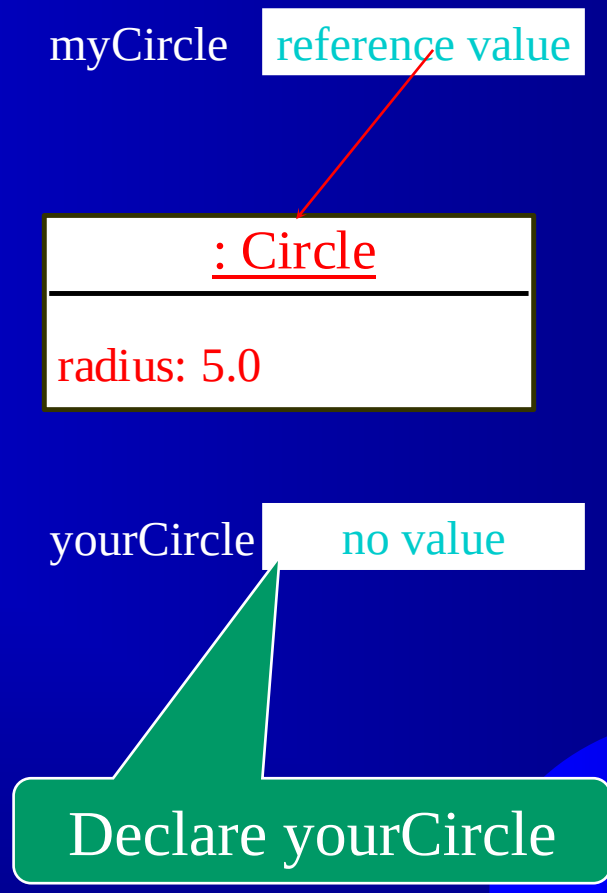
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

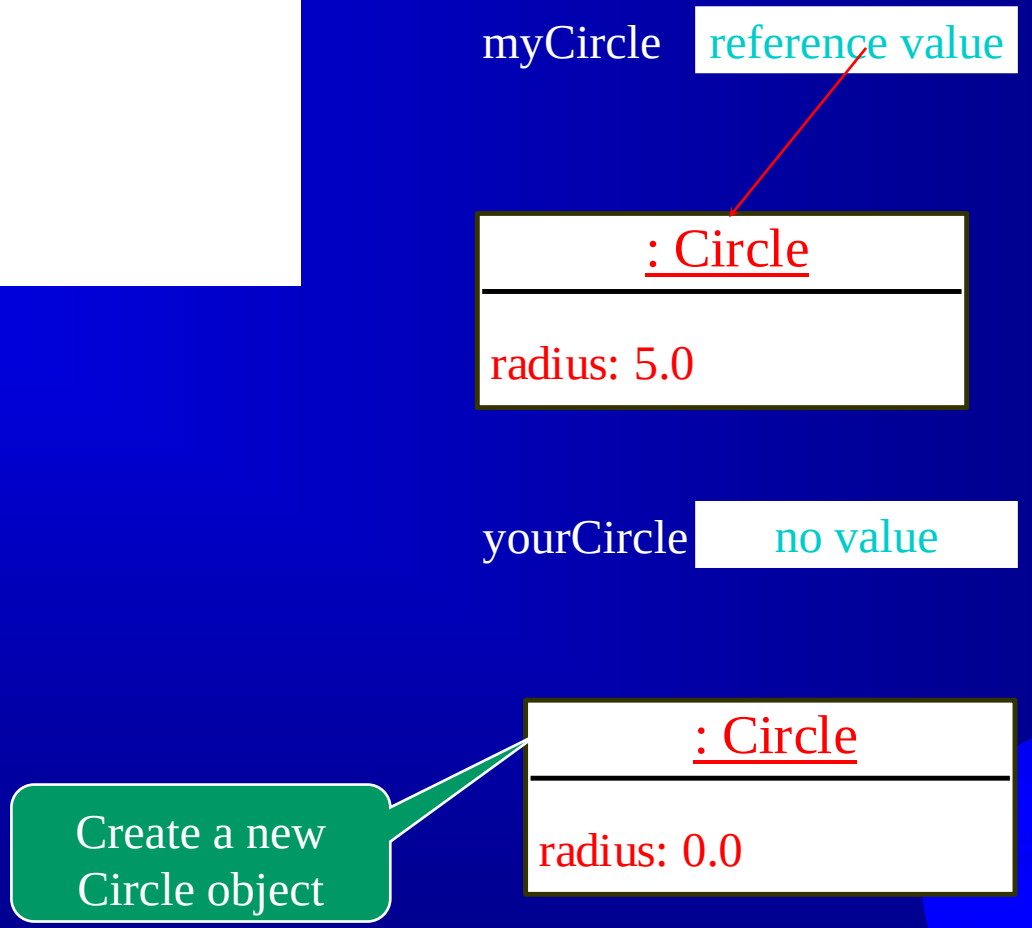
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

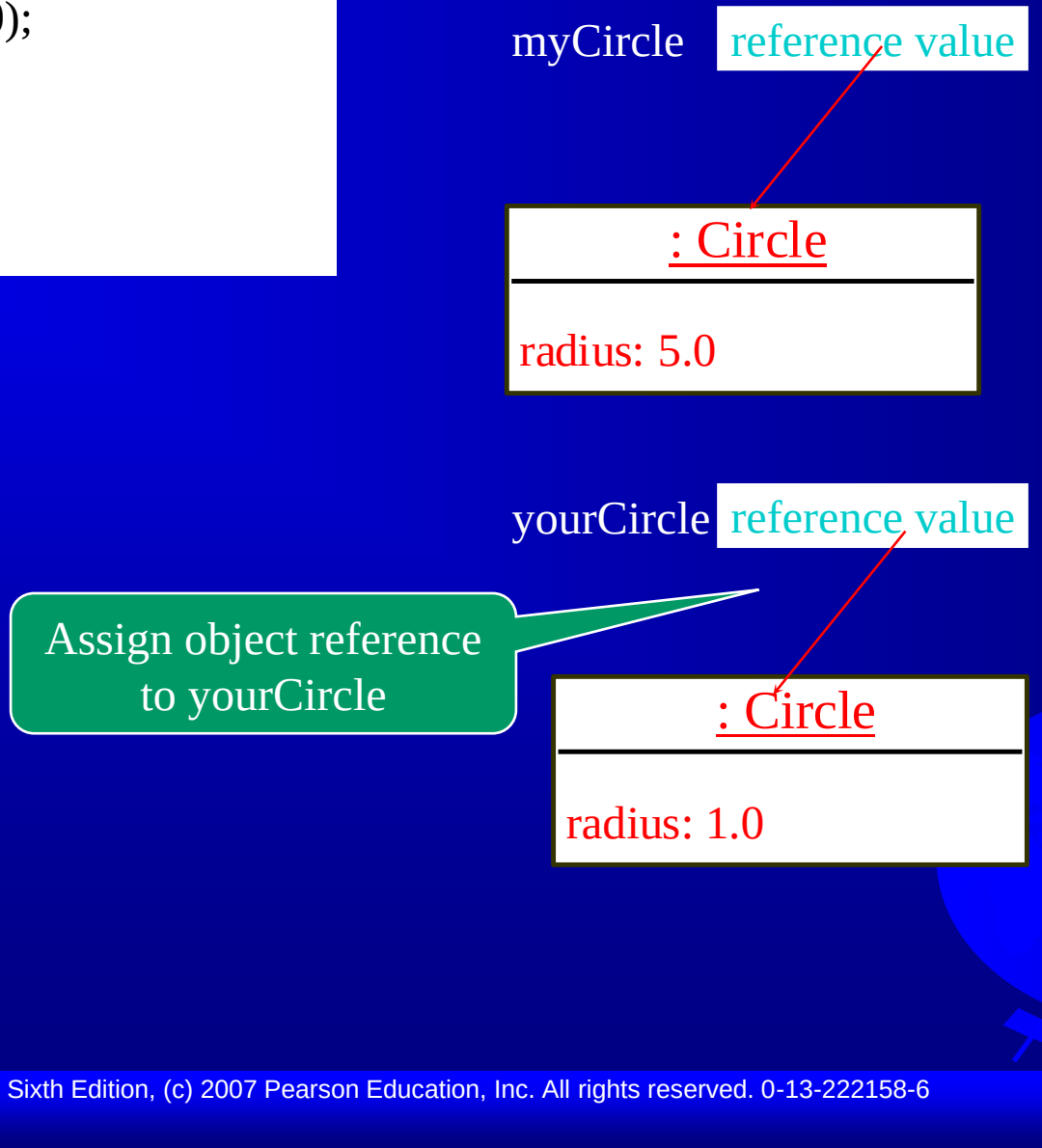
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

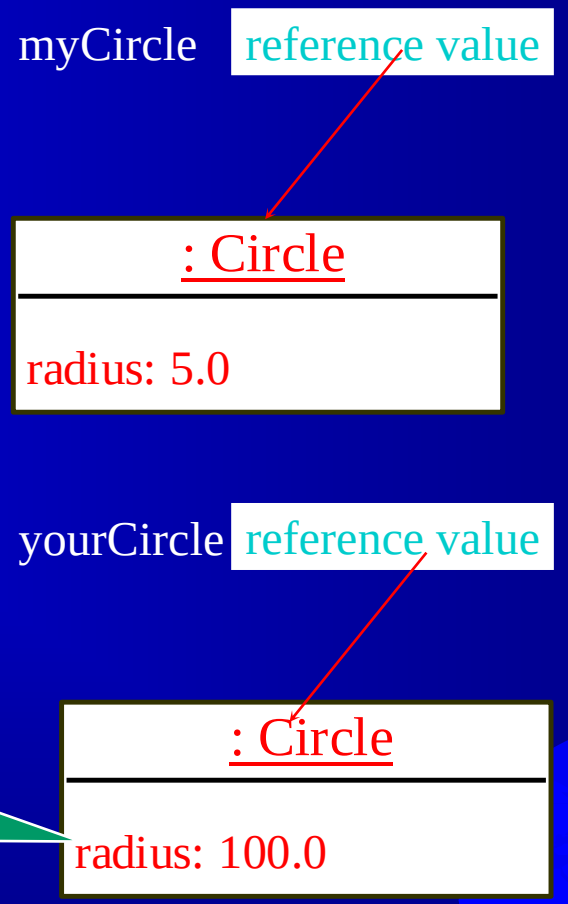
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



CAUTION

- Recall that you use

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

to invoke a method in the Math class.

- Can you invoke getArea() using Circle1.getArea() in a program?
  - 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()).

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.

DEFAULT VALUE FOR A DATA FIELD

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false
String (or any object)	null

DEFAULT VALUE FOR A DATA FIELD

```
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);
  }
}
```

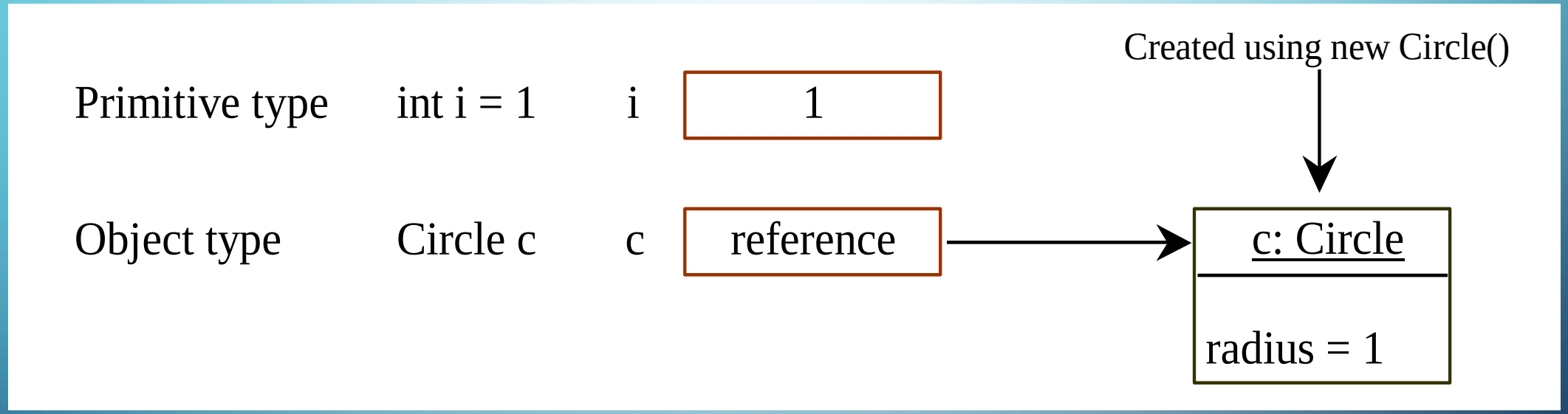
NO DEFAULT VALUE FOR A VARIABLE

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

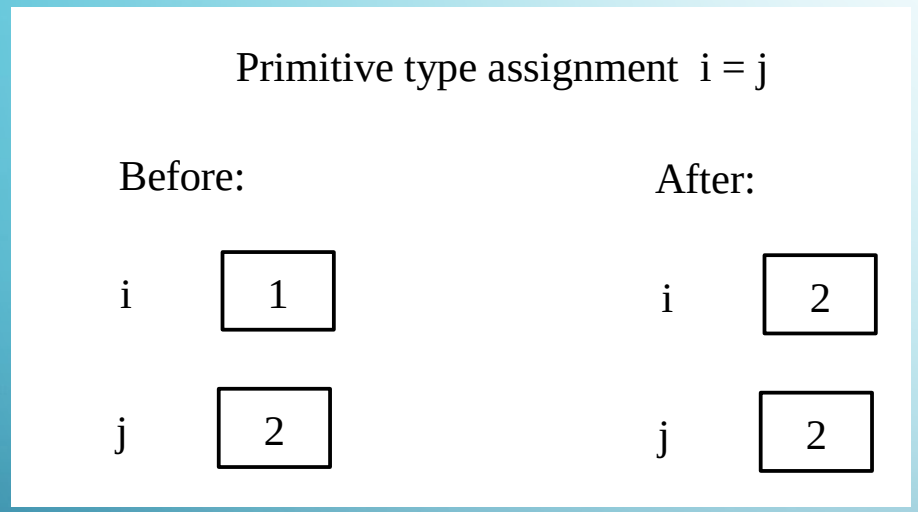
```
public class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```

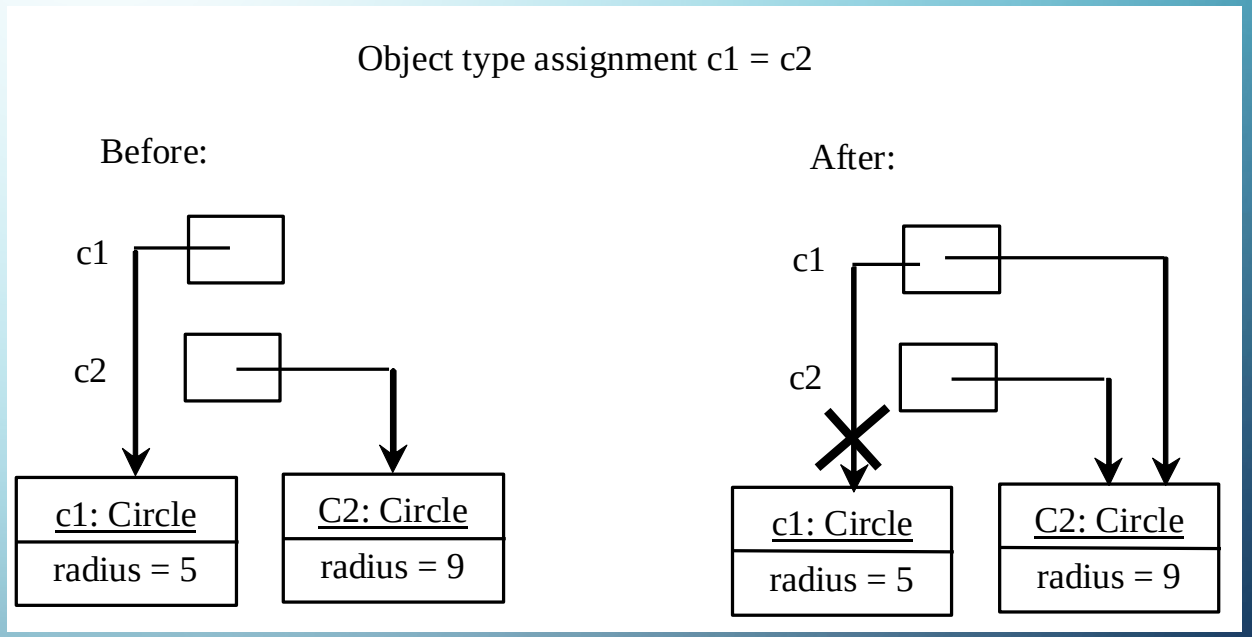
Compilation error: variables not initialized

DIFFERENCES BETWEEN VARIABLES OF PRIMITIVE DATA TYPES AND OBJECT TYPES



COPYING VARIABLES OF PRIMITIVE DATA TYPES AND OBJECT TYPES





INSTANCE VARIABLES, AND METHODS

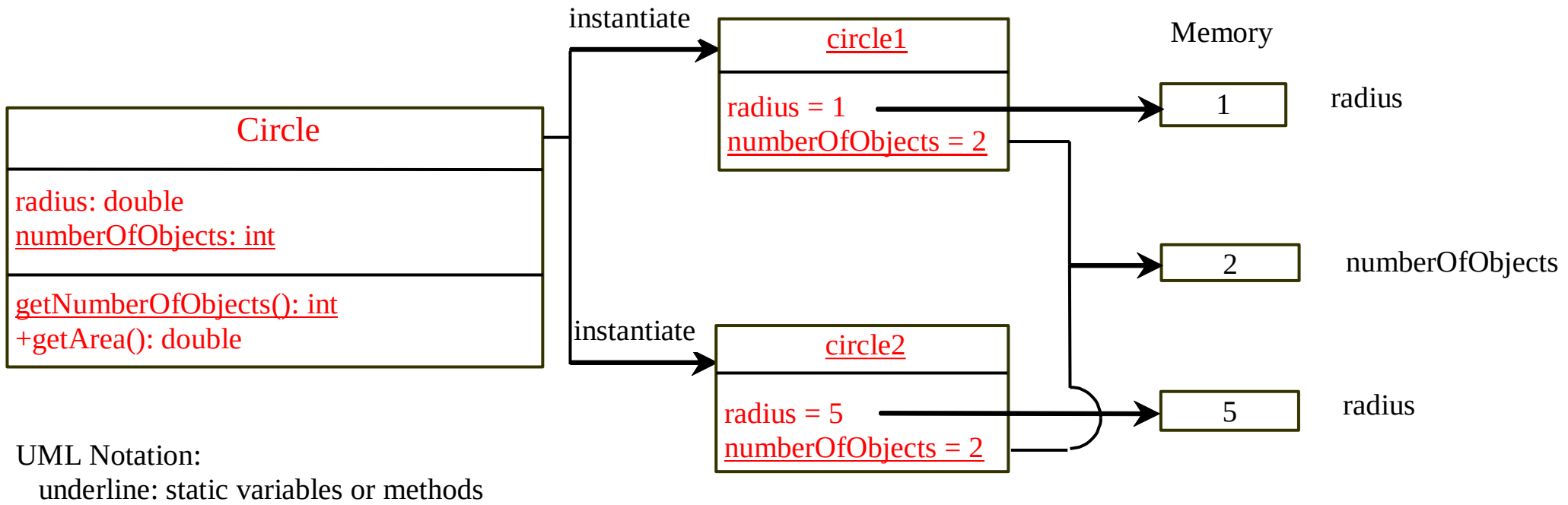
- Instance variables belong to a specific instance.
- Instance methods are invoked by an instance of the class.

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.

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

STATIC VARIABLES, CONSTANTS, AND METHODS



UML Notation:
underline: static variables or methods

VISIBILITY MODIFIERS

- 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.
- Protected: accessible from all classes in the same package as the member's class as well as subclasses of that class, regardless of package.
- Private: The data or methods can be accessed only by the declaring class.

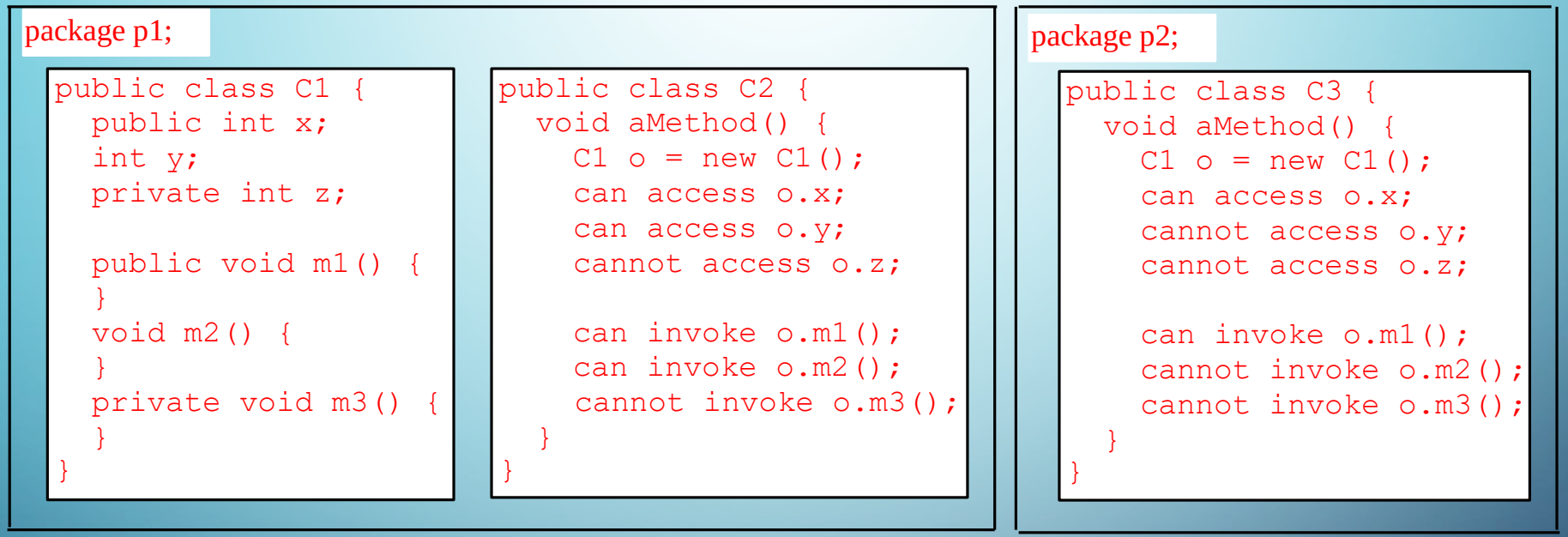
VISIBILITY MODIFIERS

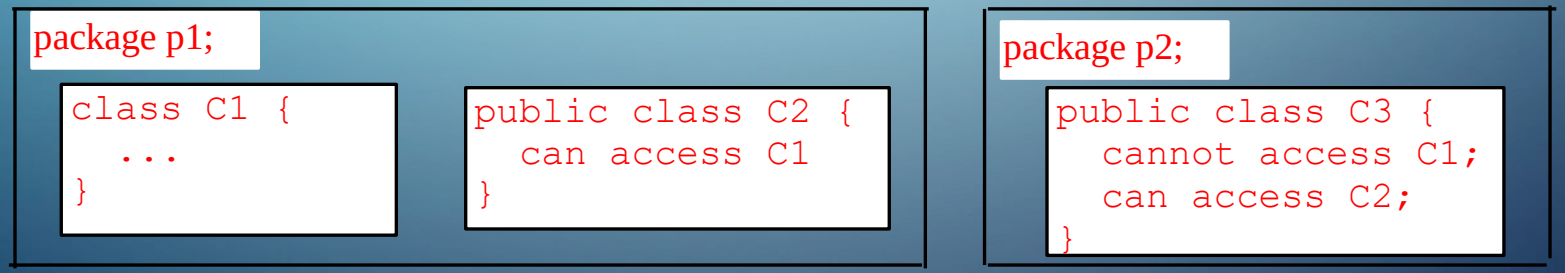
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

VISIBILITY MODIFIERS AND ACCESSOR

- Accessor is represented as get and set method.
- The get and set methods are used to read and modify private properties.

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.



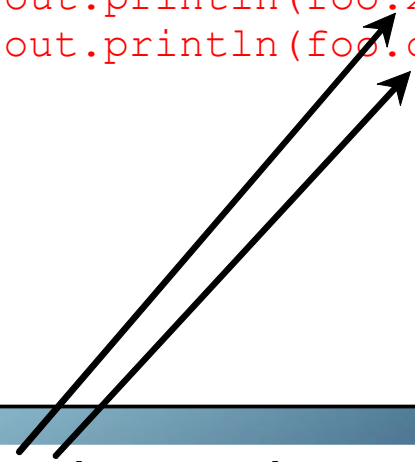


An object cannot access its private members, as shown in (b). It is OK, however, if the object is declared in its own class, as shown in (a).

```
public class Foo {
  private boolean x;

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

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

(a) This is OK because object foo is used inside the Foo class

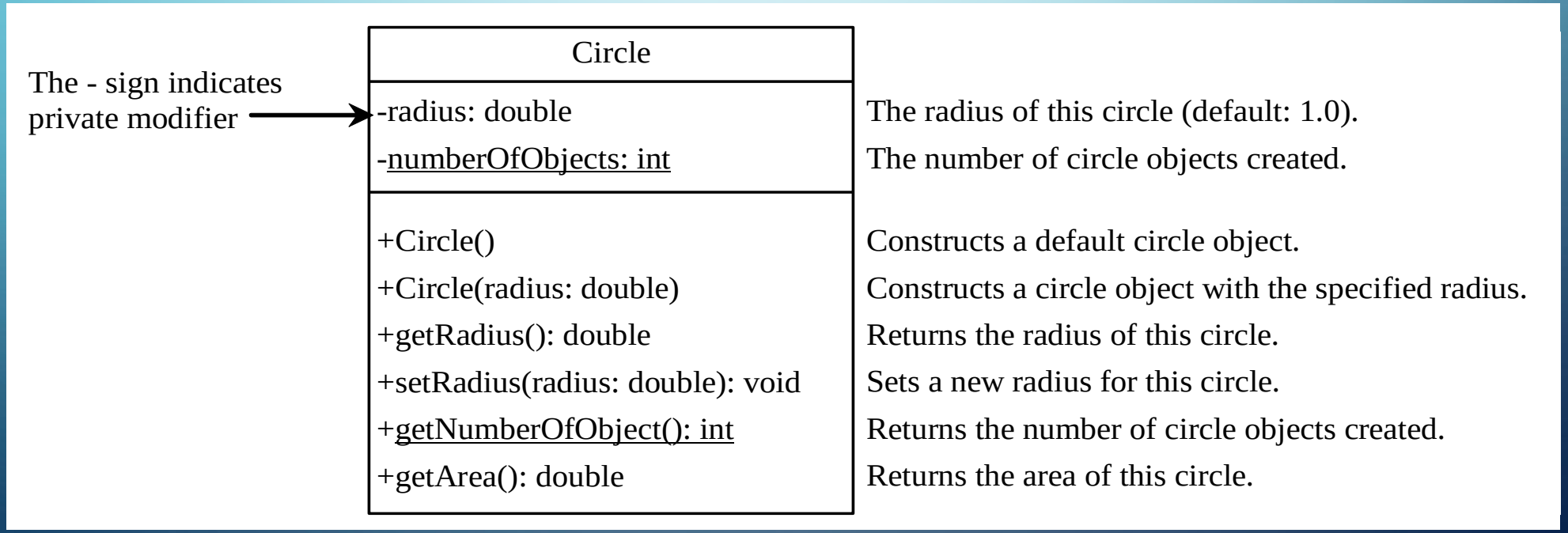
```
public class Test {
  public static void main(String[] args) {
    Foo foo = new Foo();
    System.out.println(foo.x);
    System.out.println(foo.convert(foo.x));
  }
}
```



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

WHY DATA FIELDS SHOULD BE PRIVATE?

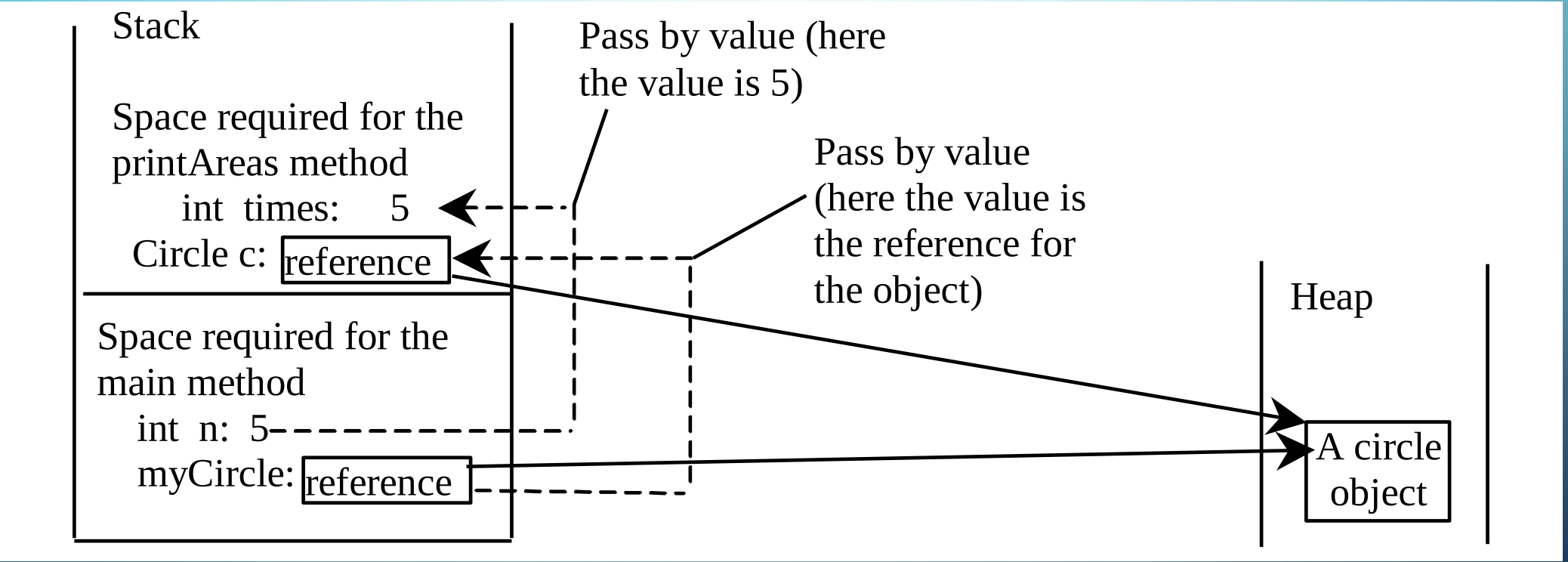
- To protect data.
- To make class easy to maintain.



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)

PASSING OBJECTS TO METHODS



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.

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.

THIS KEYWORD

```
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
```

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

OVERLOADED CONSTRUCTOR

- 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.

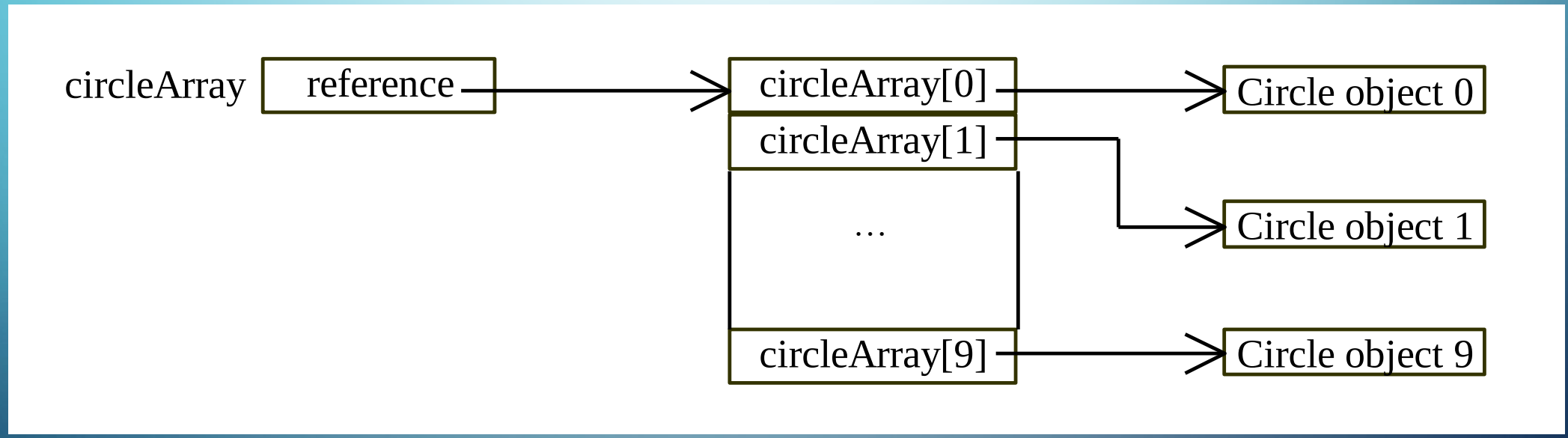
ARRAY OF OBJECTS

```
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

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



IN-CLASS EXERCISE 1

```
class CheckingAccount {
  String owner;
  int balance;
  static int counter;
  void printBalance()
  {
    // code that outputs
    // the balance field's value
  }
}
```

IN-CLASS EXERCISE (CHECKING ACCOUNT)

Add the following methods to CheckingAccount class.

- deposit
- withdraw
- printBalance

IN-CLASS EXERCISE (CHECKING ACCOUNT)

```
/**
 * Add the specified amount to balance
 * @param amount
 * @return the new balance
 */
int deposit(int amount)
```

IN-CLASS EXERCISE (CHECKING ACCOUNT)

```
/**
 * Subtract the specified amount from the balance.
 * @param amount
 * @return the new balance
 */
int withdraw(int amount)
```

IN-CLASS EXERCISE (CHECKING ACCOUNT)

```
/**
 * Print the current balance using the standard
 * currency format i.e, ###,###.## for positive
 * values and (###,###.##) for negative values
 */
void printBalance()
```

IN-CLASS EXERCISE (CHECKING ACCOUNT)

```
void printBalance()
{
  int magnitude = (balance < 0) ? -balance : balance;
  String balanceRep = (balance < 0) ? "(" : "";
  balanceRep += magnitude;
  balanceRep += (balance < 0) ? ")" : "";
  System.out.println(balanceRep);
}
```

REFERENCES

[1] Liang, “Introduction to Java Programming”, 6th Edition, Pearson Education, Inc.