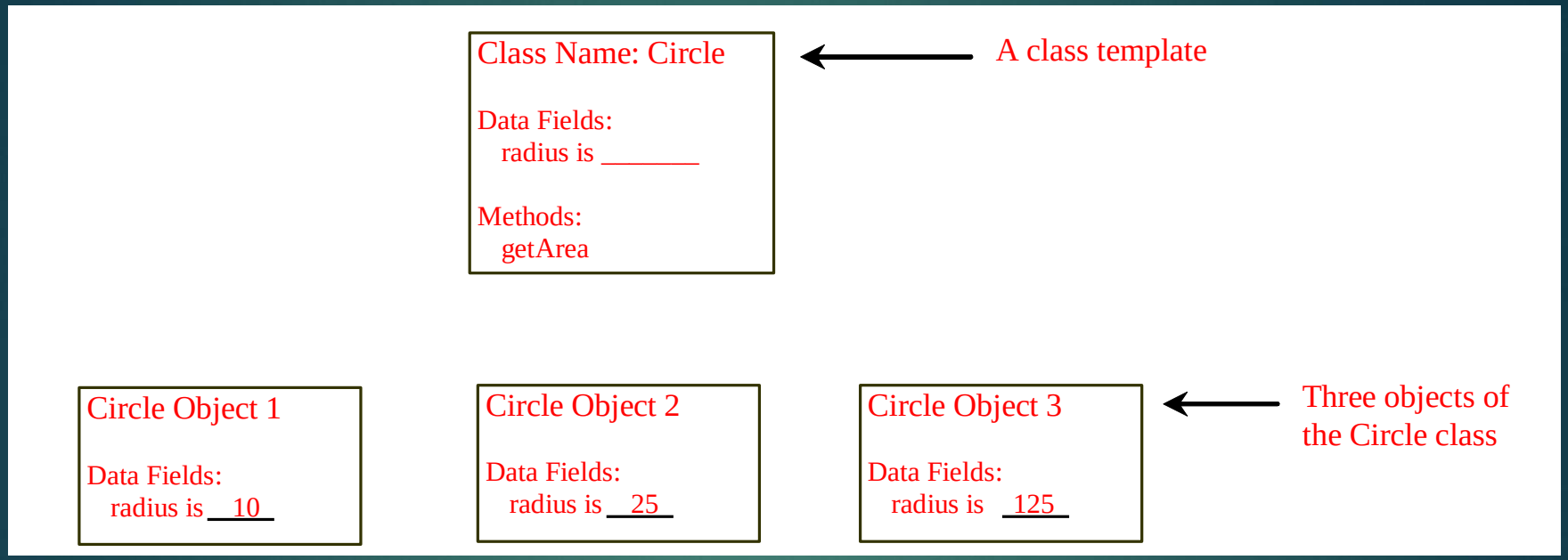
Chapter 8 Objects and Classes

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



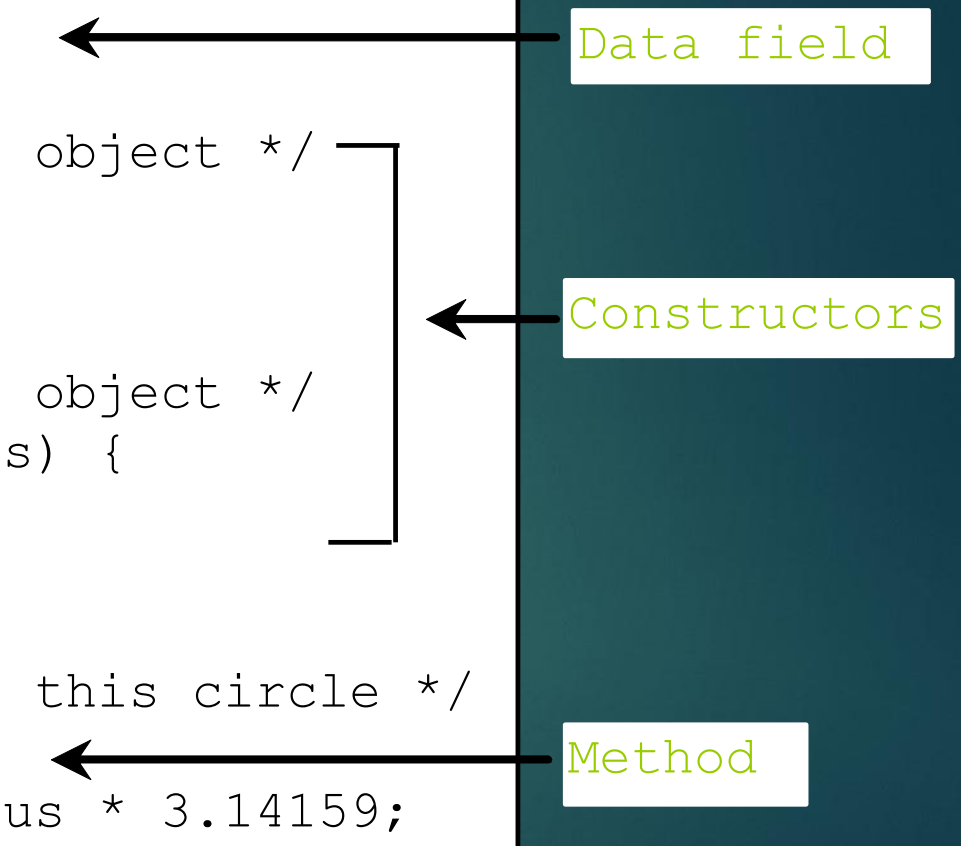
An object has both a state and behavior. The state defines the object, and the behavior defines what the object does.

Classes

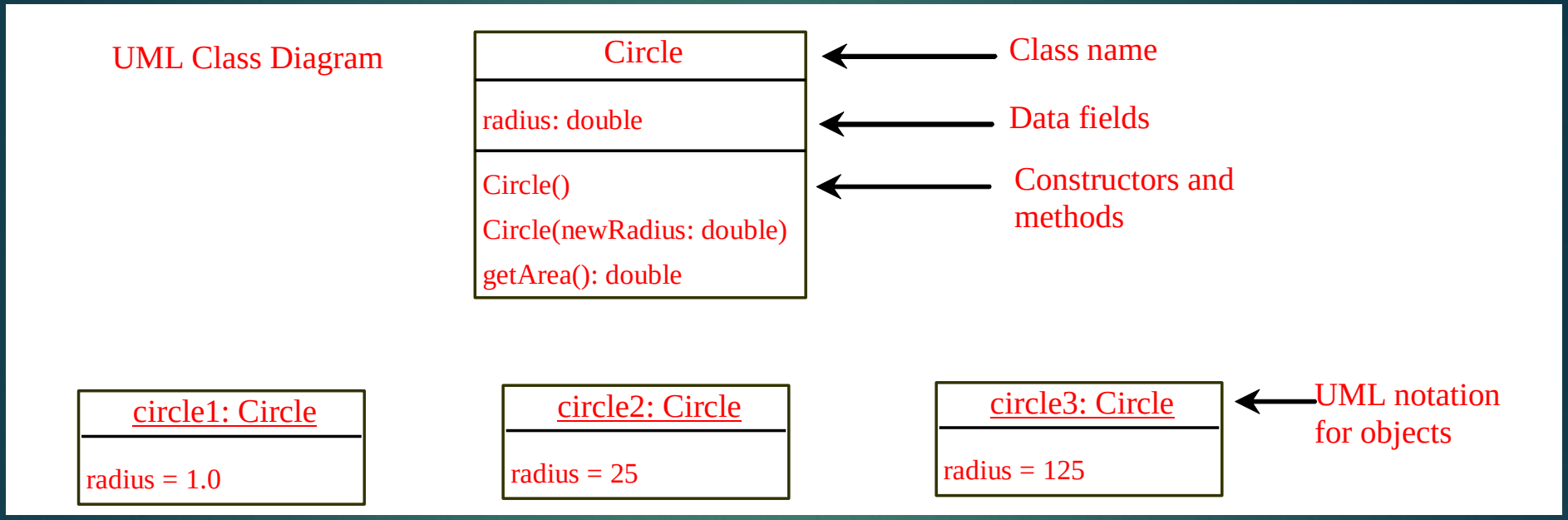
Classes are constructs that define objects of the same type. A Java class uses variables to define data fields and methods to define behaviors. Additionally, a class provides a special type of methods, known as constructors, which are invoked to construct objects from the class.

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;
  }
}
```



UML Class Diagram



Example: Defining Classes and Creating Objects

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

TestCircle1

Run

Constructors


```
Circle() {
}

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

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

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*.

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:

Accessing Objects

- Referencing the object's data:

```
objectRefVar.data
```

e.g., `myCircle.radius`

- Invoking the object's method:

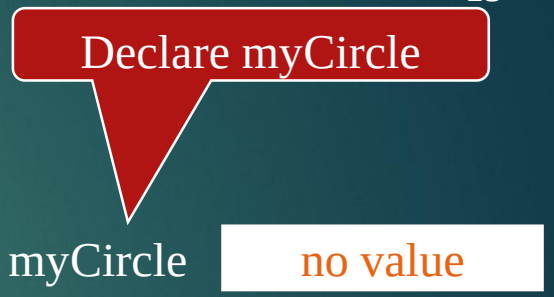
```
objectRefVar.methodName(arguments)
```

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

animation

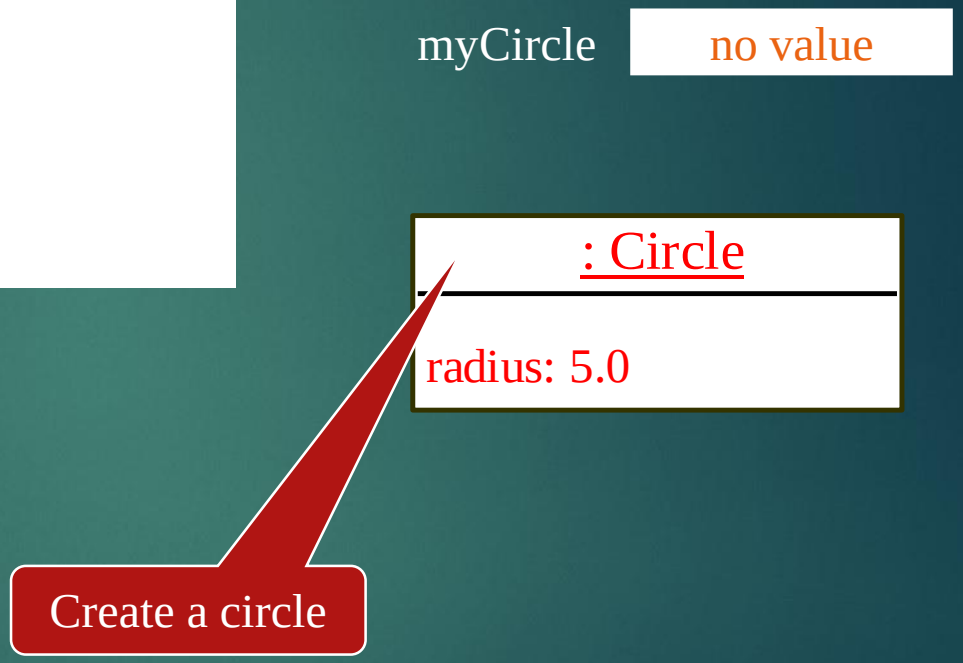
Trace Code


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

SCircle yourCircle = new Circle();

yourCircle.radius = 100;
```



Trace Code, cont.


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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

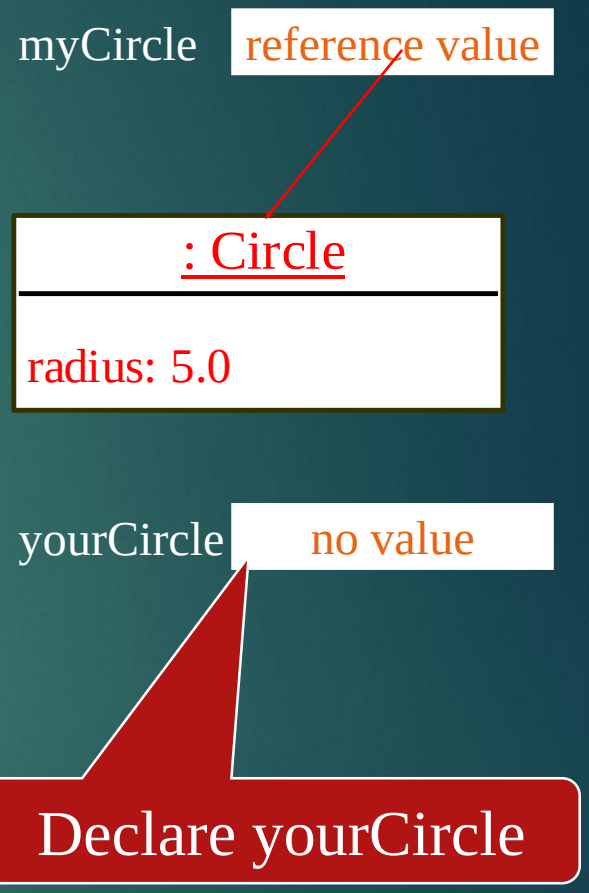
Trace Code, cont.


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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

animation

Trace Code, cont.

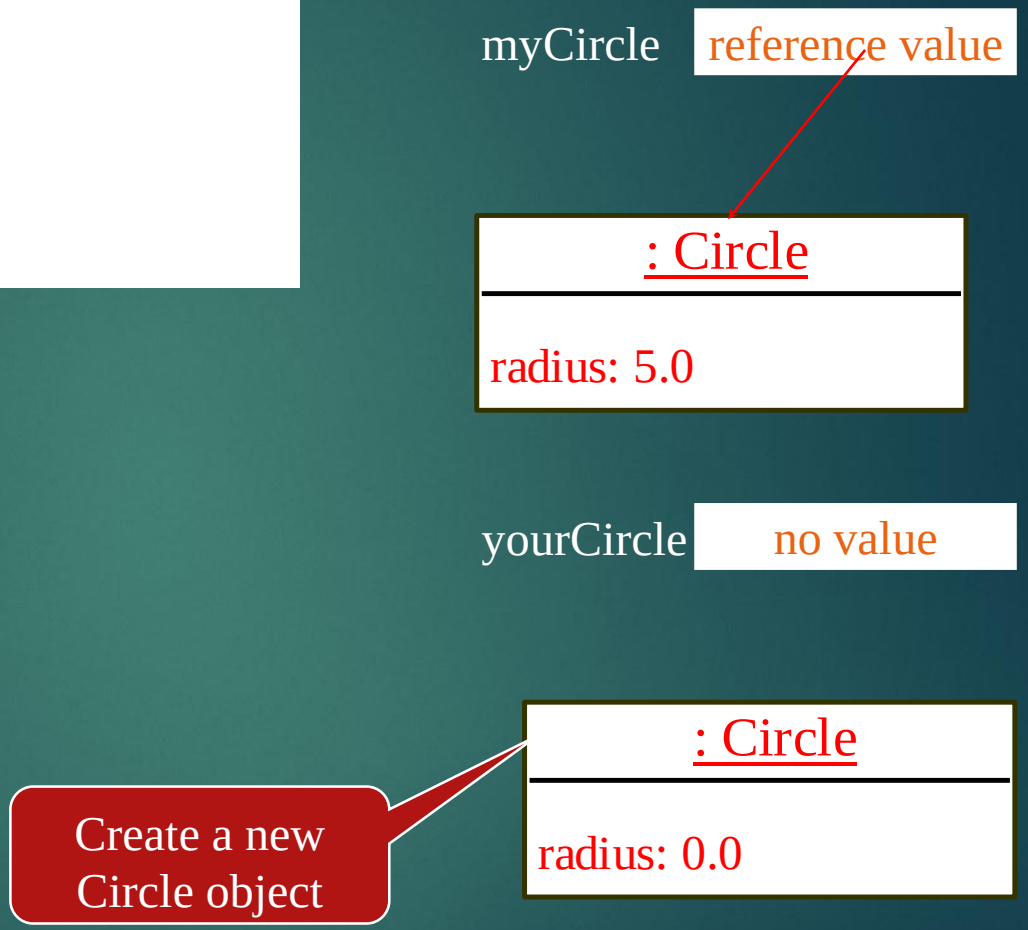

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

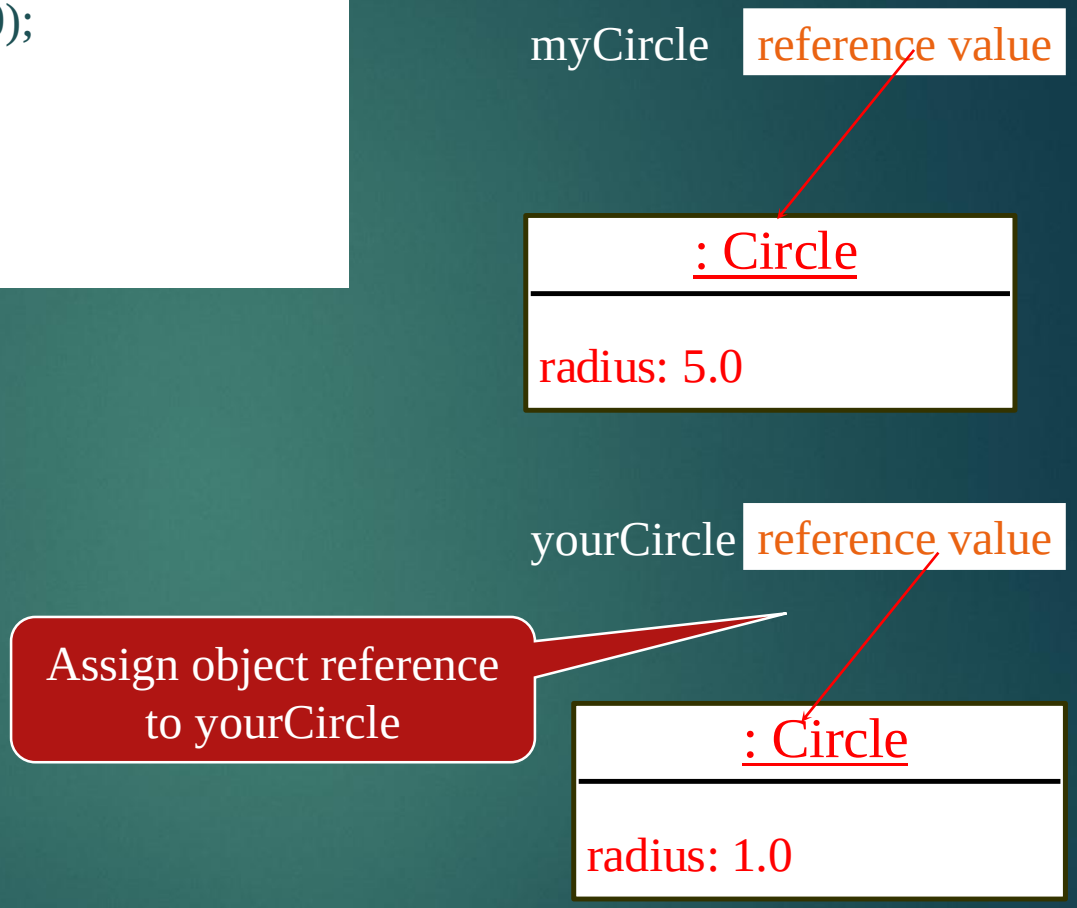

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

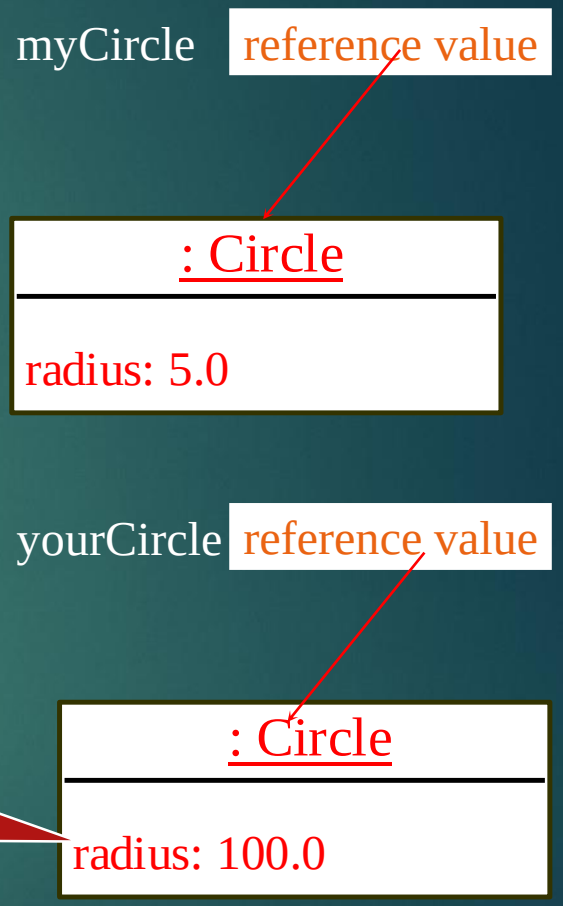

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



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()? 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

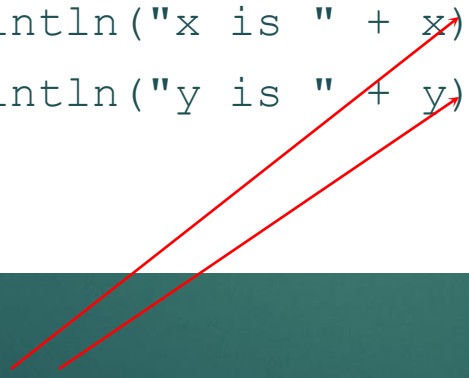
The default value of a data field is null for a reference type, 0 for a numeric type, false for a boolean type, and '\u0000' for a char type. However, Java assigns no default value to a local variable inside a method

```
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);
  }
}
```

Example

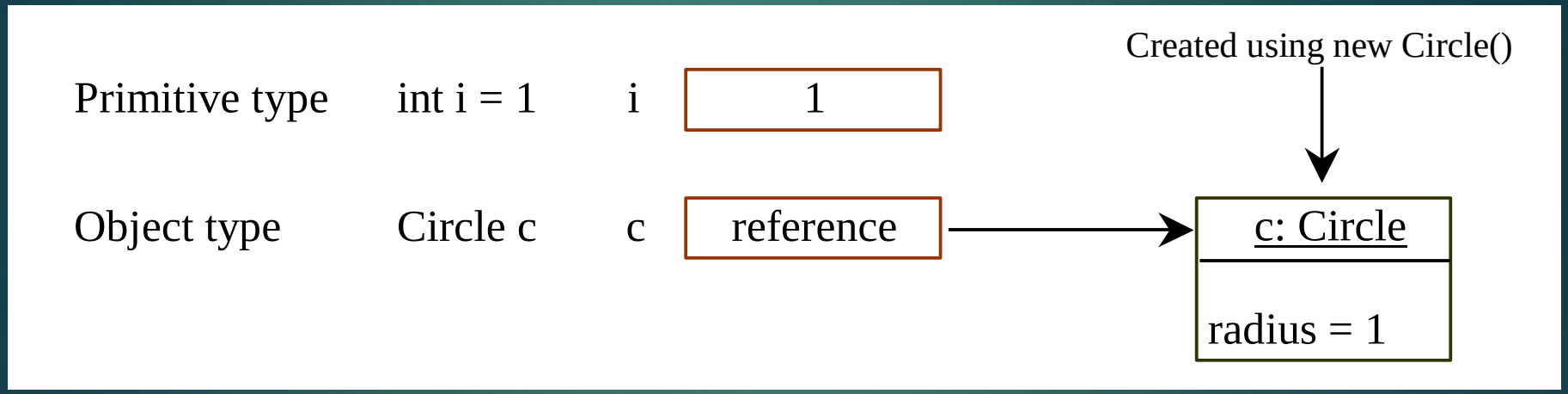
Java assigns no default value to a local variable inside a method.

```
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);
  }
}
```

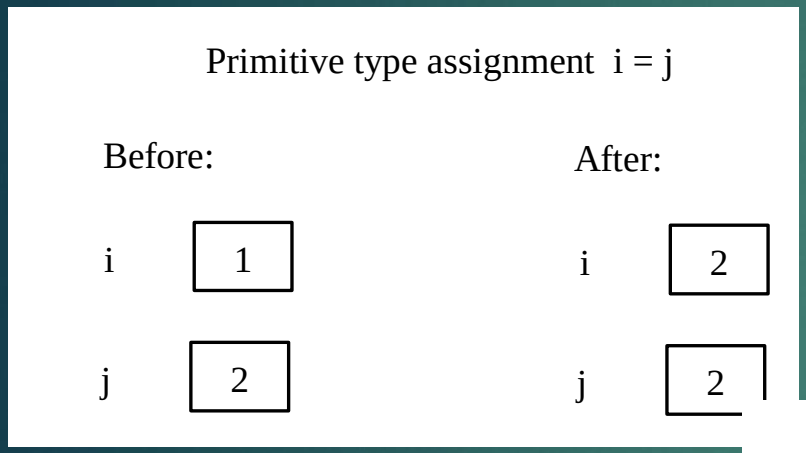


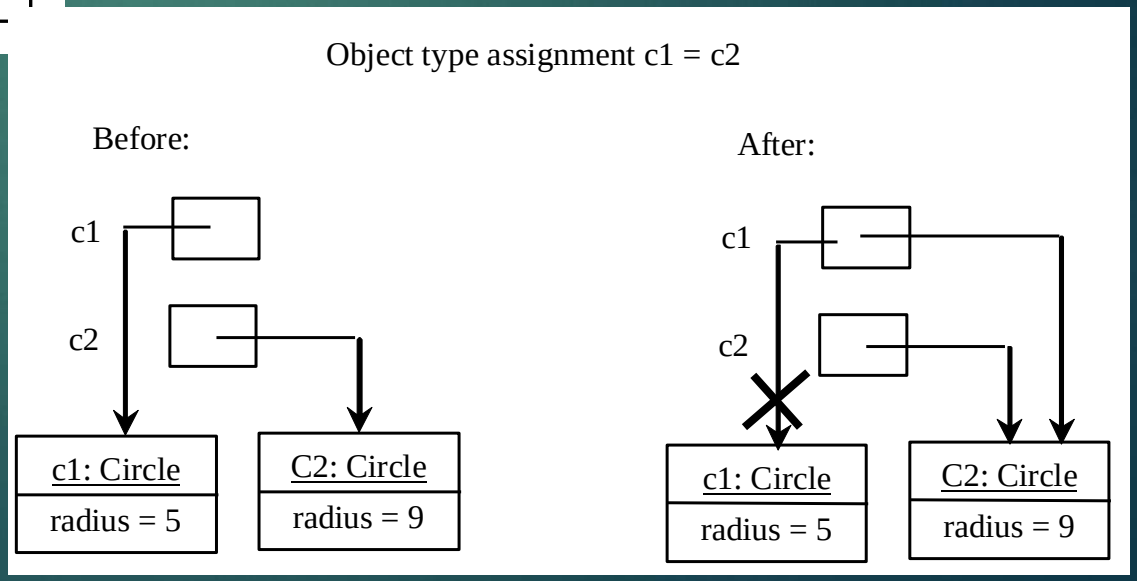
Compilation error: variables not initialized

Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types





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 modifer

java.util.Date	
+Date()	Constructs a Date object for the current time.
+Date(elapseTime: long)	Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.
+toString(): String	Returns a string representing the date and time.
+getTime(): long	Returns the number of milliseconds since January 1, 1970, GMT.
+setTime(elapseTime: long): void	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
```

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.

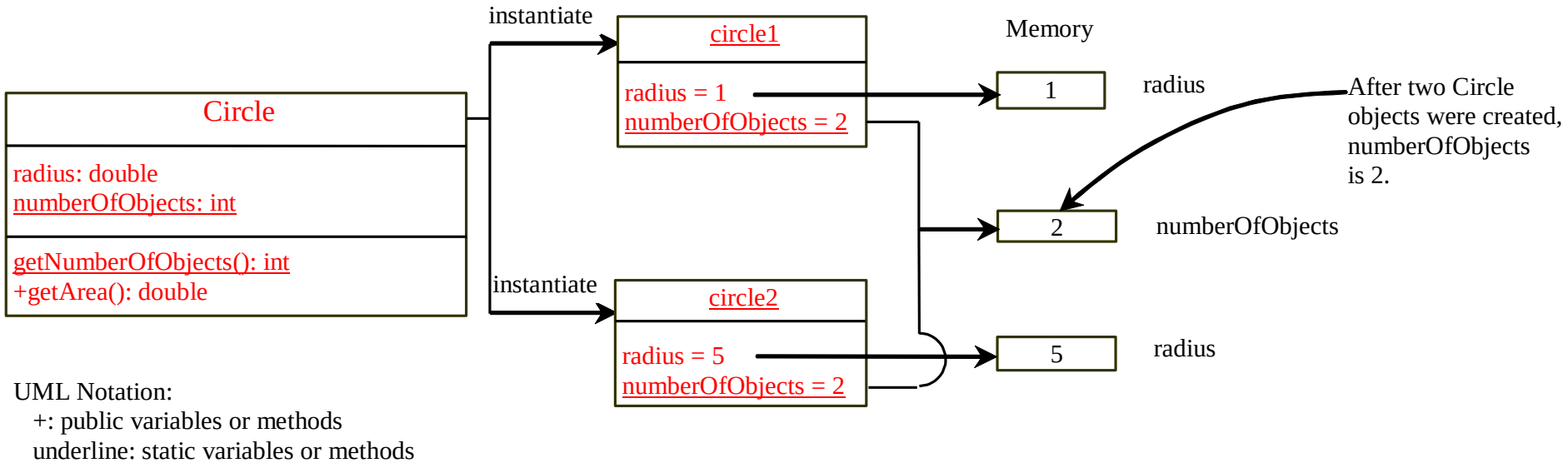
Static methods are not tied to a specific object.

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
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.

Circle2

TestCircle2

Run

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 {
  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 {
  ...
}
```

```
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.

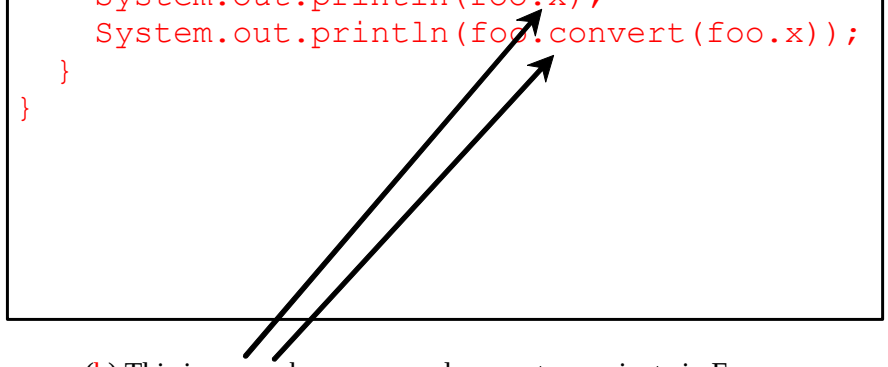
NOTE

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 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.

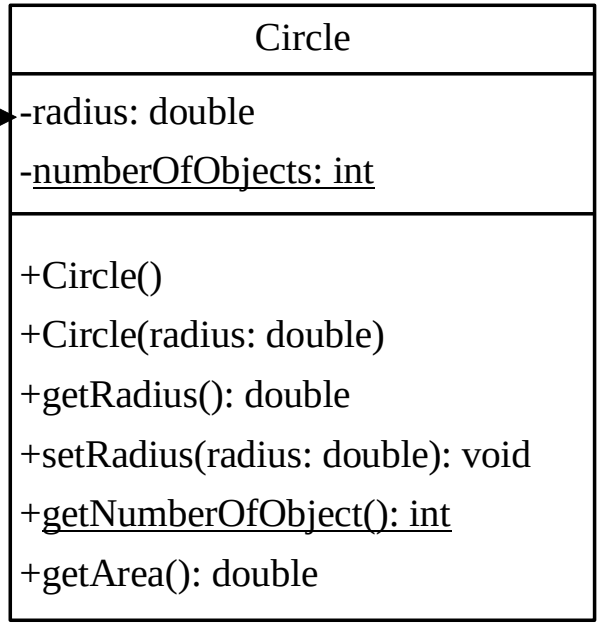
Why Data Fields Should Be private?

To protect data.

To make class easy to maintain.

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.

Circle3

TestCircle3

Run

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

Run

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];
```