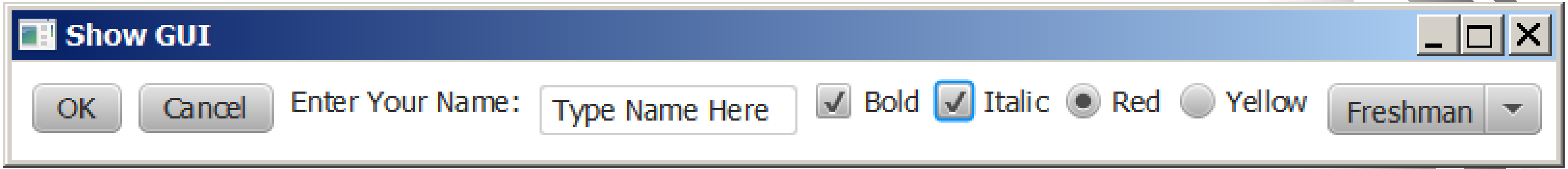
Objects and Classes

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?



Enter Your Name: Type Name Here

☑ Bold ☑ Italic ● Red ○ Yellow

Freshman

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.

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.

Defining Classes

- A class is declared by using the keyword class. The general syntax for a class declaration is

<modifier> class <className> { }

- Where <className> specifies the name of the class
- class is the keyword, and
- The <modifier> specification is optional, and specifies some characteristics of the class.

The class hierarchy

- Classes are arranged in a hierarchy
- The root, or topmost, class is `Object`
- A class may have subclasses
- Each class *inherits* all the fields and methods of its (possibly numerous) superclasses

The basic form of class declaration is as below:

```
class classname [extends superclass]
{
[variables declarations;]  [method declarations;]
}
```

Adding variables

```
class Rectangle
{
int length ;  int
width;
}
```

Methods

- A method is a named sequence of code that can be invoked by other Java code.

- A method takes some parameters, performs some computations and then optionally returns a value (or object).

- Methods can be used as part of an expression statement.

Adding method

The general form of method declaration is :

type mehodname (parameter list)

{

method body;

}

Method declarations has four basic parts

□ Method name Returns

□ type Parameter list

□ The body of the method

□

```
class Rectangle
{
int length ;  int width   ;
void getdata(int x,int y)
{
length =x;  width = y;
}
}
```

```
class Rectangle
{
int length ;  int width       ;
void getdata(int x,int y)
{
length=x;  width = y;
}
int rectArea()
{
int area = length*width;
return(area);
}
}
```

Creating Objects

- When you write a class, you must keep in mind the objects that will be created from it, which correspond to the objects in the problem that is being solved by the program. You may also look upon the classes as data types.
- for example, of a primitive data type and assign it a value, as follows:

```
int i=0;
```

Creating objects

Objects in java are created using the **new** operator .

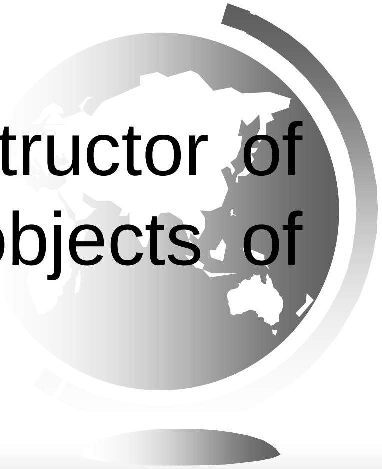
The new operator creates an objects of the specified class and returns a reference to that objects.

Ex.

Rectangle rect1;   //declare rect1 = new

rectangle();//instantiate

The method rectangle is the default constructor of the class .we can create any no. of objects of rectangle



Syntax of object

- Similarly, you can declare a variable (a reference variable) of a class and assign it a value with the following syntax:

  <className> <variableName> = new <classConstructor>

- Eg. Room1 obj = new Room1();
- Where <variableName> is the name of the object and you choose this name. <className> is the name of an existing class.

Accessing class member

Objectname.variable name

Objectname.methodname(parameter-list) ;

rect1.length =15

Using Object

- New operator dynamically allocates memory for an object and returns reference to it.

```
Fruit plum=new Fruit();  int cals;
cals = plum.total_calories();
```

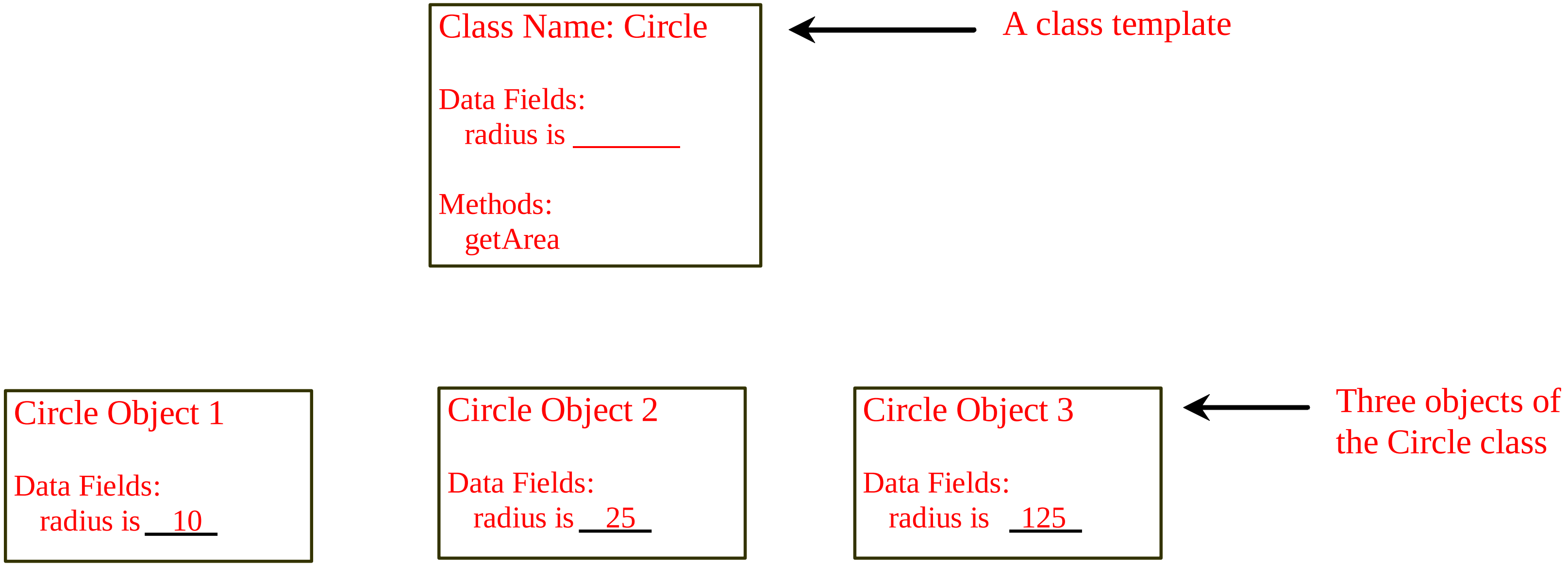
- ***Dot operator*** allows you to access (public) data/methods inside Fruit class

illustration of class and object

```
class Rect
{
int length;  int width;

void getdata(int x,int y)
{
length=x;  width=y;
}
int rectArea()
{
int area=length*width;
return(area);
}
}
```

```
class RectangleArea
{
public static void main(String[]args)
{
int area1,area2;
Rect  rect1=new  Rect();    Rect  rect2=new
Rect();  rect1.length = 1;
rect1.width = 2;
area1=rect1.length*rect1.width;
rect2.getdata(5,10);  area2=rect2.rectArea();
System.out.println("area1="+area1);
System.out.println("area2="+area2);
}
}
```

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

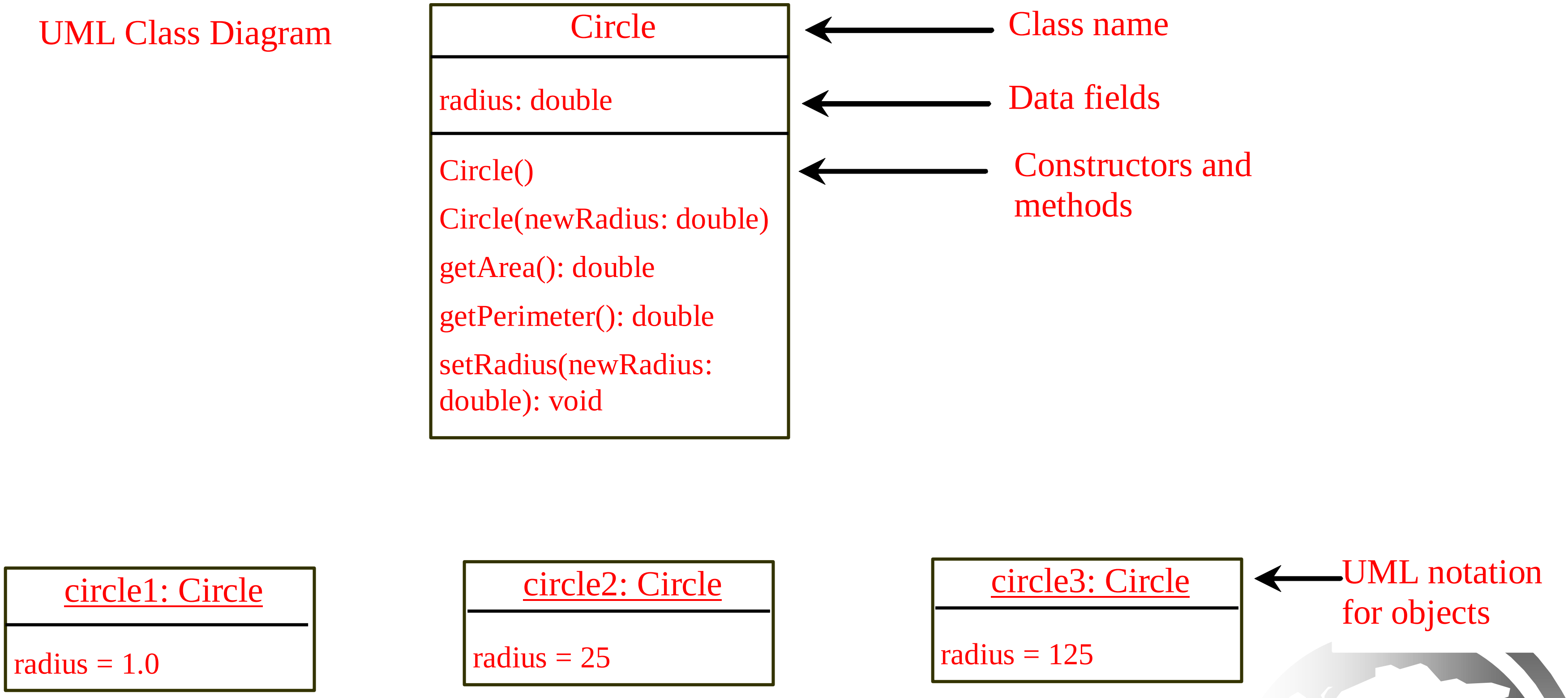
Data Fields:
radius is _125_

Three objects of
the Circle class

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

UML Class Diagram

UML Class Diagram



Circle

radius: double

Circle()

Circle(newRadius: double)

getArea(): double

getPerimeter(): double

setRadius(newRadius: double): void

Class name

Data fields

Constructors and methods

circle1: Circle

radius = 1.0

circle2: Circle

radius = 25

circle3: Circle

radius = 125

UML notation for objects

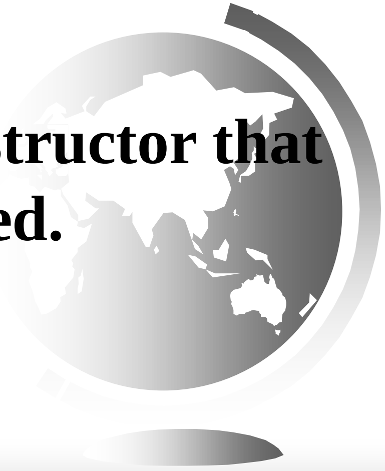
Constructors

We know that all object that are created need initial values.

One approach for this is the use of Dot operator through which we access the instance variables and then assign values to them individually

In second approach we use method like getdata to initialize each object individually.

Java supports a special type of method called constructor that enable an object to initialize itself when it is created.



Constructor

- Constructor initializes an object immediately on creation.
- Constructor is automatically called immediately after the object is created.
- Constructor name is same as class name and have no return type not even **void.**

Example

```
Class Box
{
int w,h,d;
Box() { //contructor  w=10;
h=10;
d=10;     }

public static void main(String str[])
{
Box b= new Box() }
}
```

Replacement of getdata by a constructor method

```
class Rectangle
{
int length ;  int width         ;
void getdata(int x,int y)
{
length=x;  width = y;
}
int rectArea()
{
int area = length*width;
return(area)
}
}
```

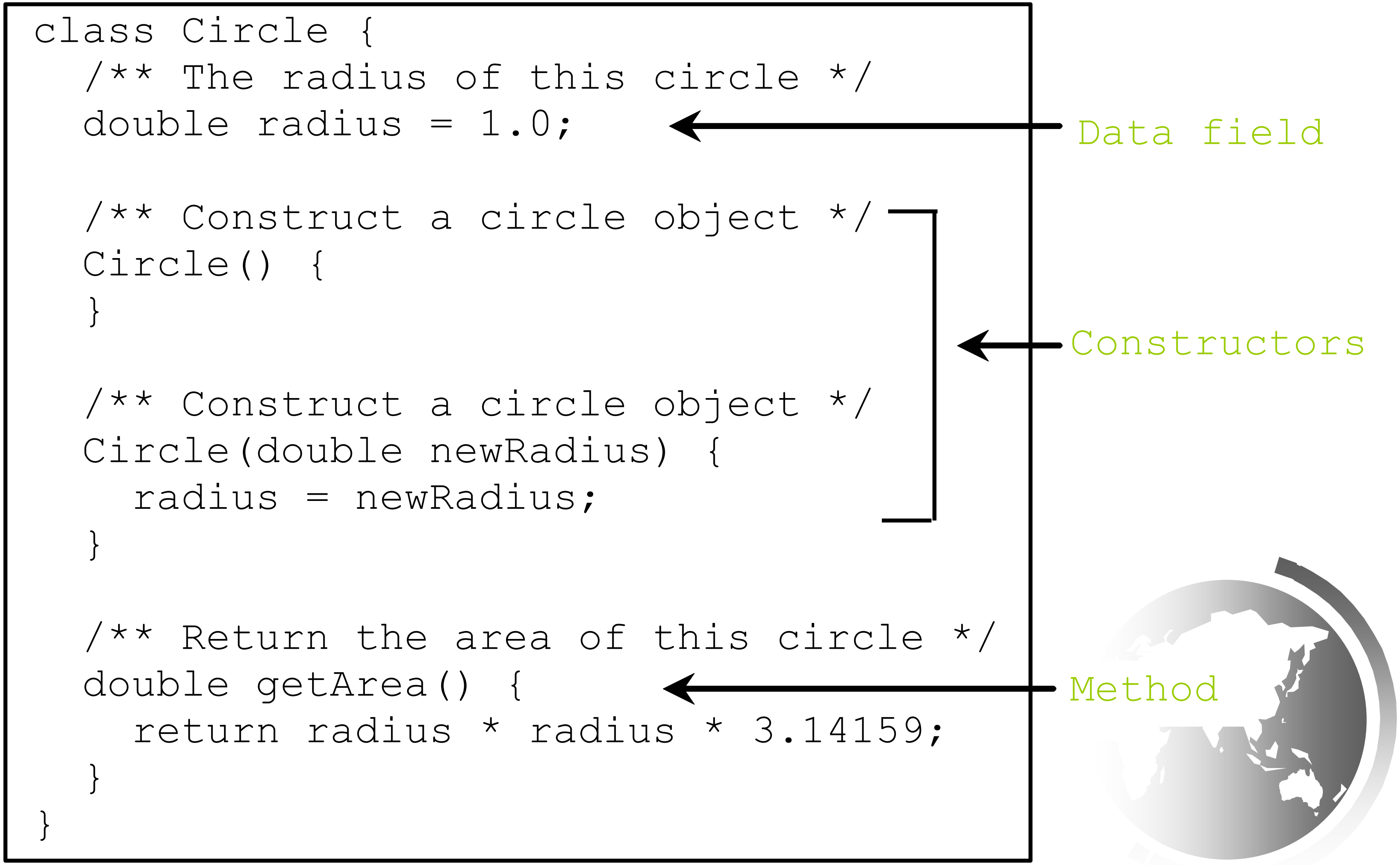
```
class Rectangle
{
int length ;  int width         ;
Rectangle (int x , int y)     // constructor  method
{
length=x;  width = y;
}
int rectArea()
{
int area = length*width;
}
}
```

illustration of constructors

```
class Recta
  {
        static int length;
        static int width;


       void getdata(int x,int y)
         {
          length=x;
           width=y;
         }
         Recta(int x,int y)

  {

   length=x;

    width=y;

  }
Recta ()

  {

                }
         int rectArea()
         {
          int area=length*width;
           return(area);
         }
  }
```

```
class RectArea
{
public static void main(String[]args)
{
   int area1,area2;
   Recta rect1=new Recta();
   Recta rect2=new Recta(10,20);
   rect1.length = 1;
   rect1.width = 2;
   area1=rect1.length*rect1.width;
   rect2.getdata(5,10);
   area2=rect2.rectArea();
System.out.println("area1="+area1);
System.out.println("area2="+area2);
}
}
```

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;
  }
}
```

Example: Defining Classes and Creating Objects

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

TestSimpleCircle

Run

www.cs.armstrong.edu/liang/intro11e/html/TestSimpleCircle.html

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 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*.

Difference between constructors and methods

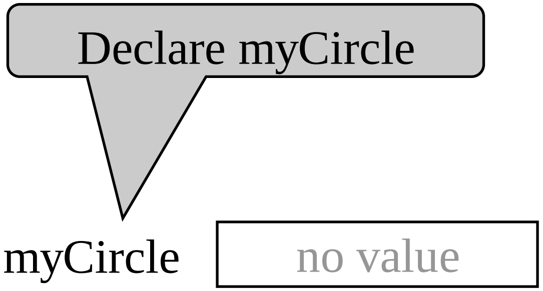
- The important difference between methods is that constructors create and initialize objects that don't exist yet, while methods perform operations on objects that already exist.
- Constructors can't be called directly; they are called implicitly when the new keyword creates an object. Methods can be called directly on an object that has already been created with new.

- The definitions of constructors and methods look similar in code. They can take parameters, they can have modifiers (e.g. public), and they have method bodies in braces.
- Constructors must be named with the same name as the class name. They can't return anything, even void (the object itself is the implicit return).
- Methods must be declared to return something, although it can be void.

animation

Trace Code

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Declare myCircle

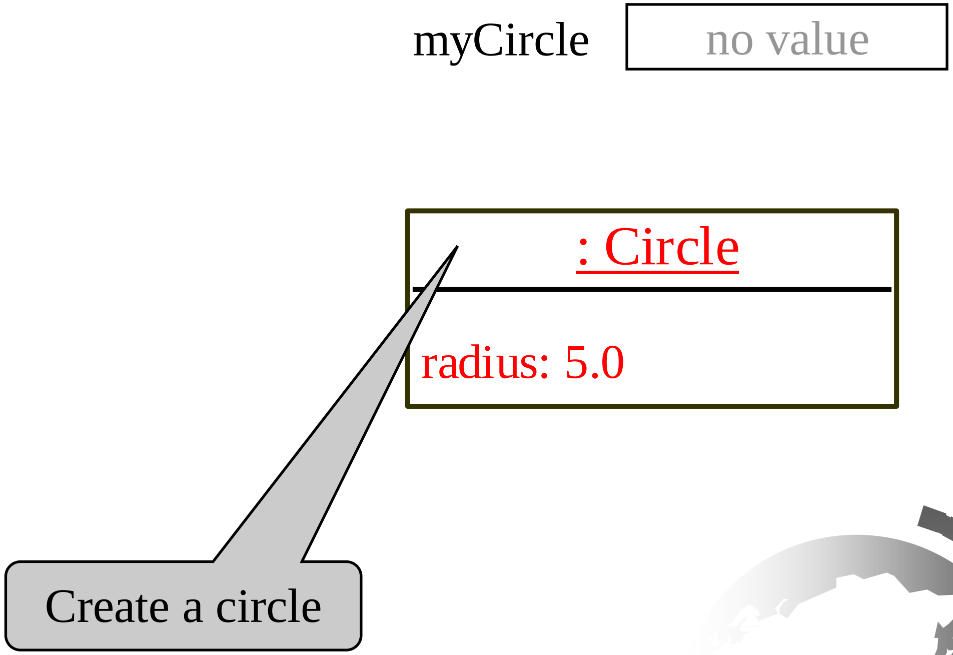
myCircle

no value

animation

Trace Code, cont.

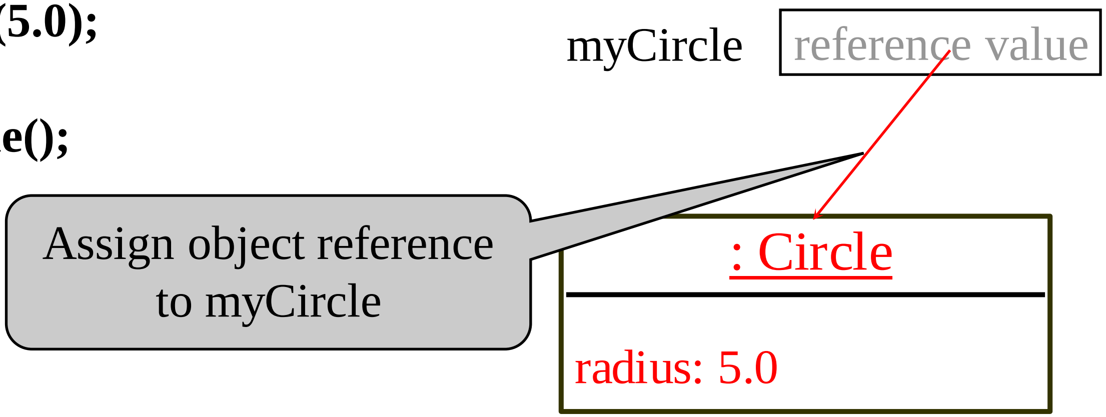
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

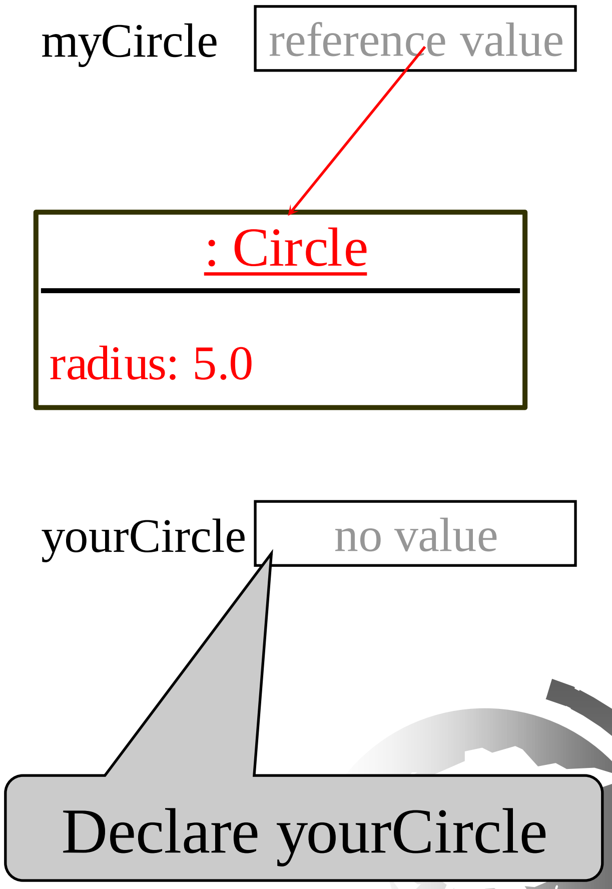
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

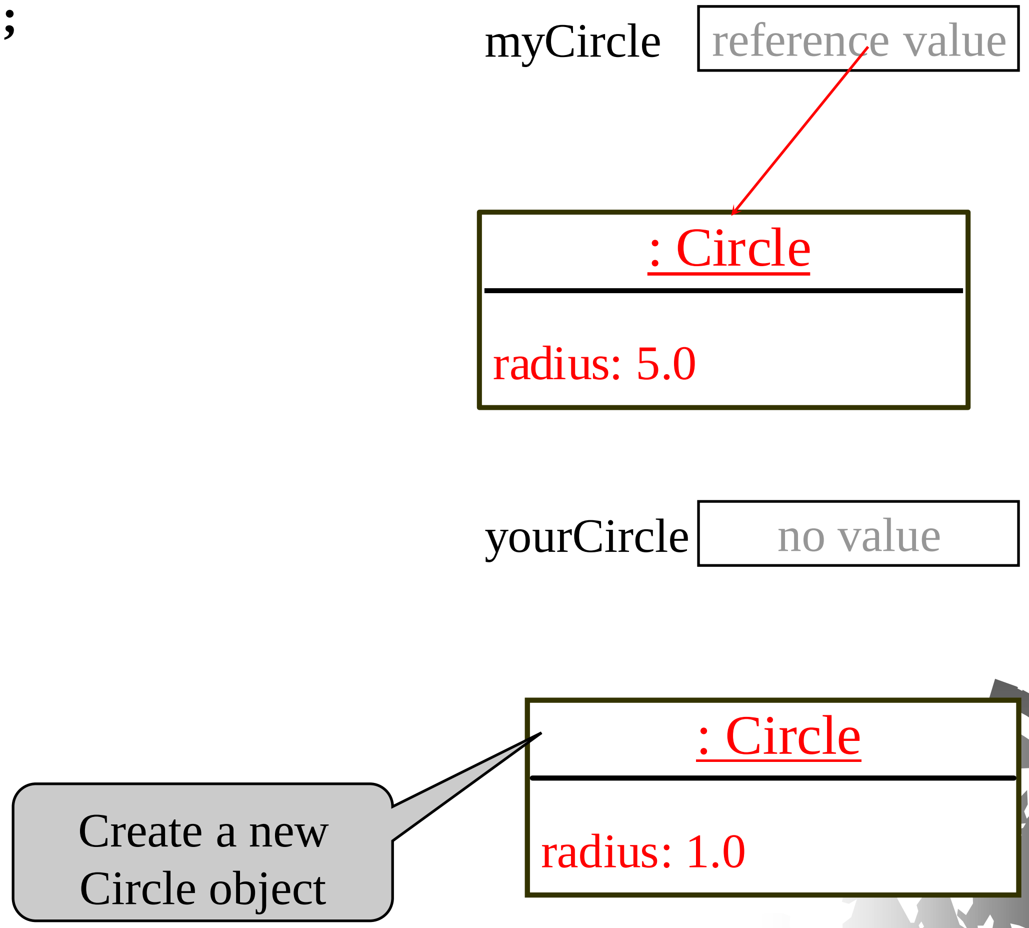


Declare yourCircle

animation

Trace Code, cont.

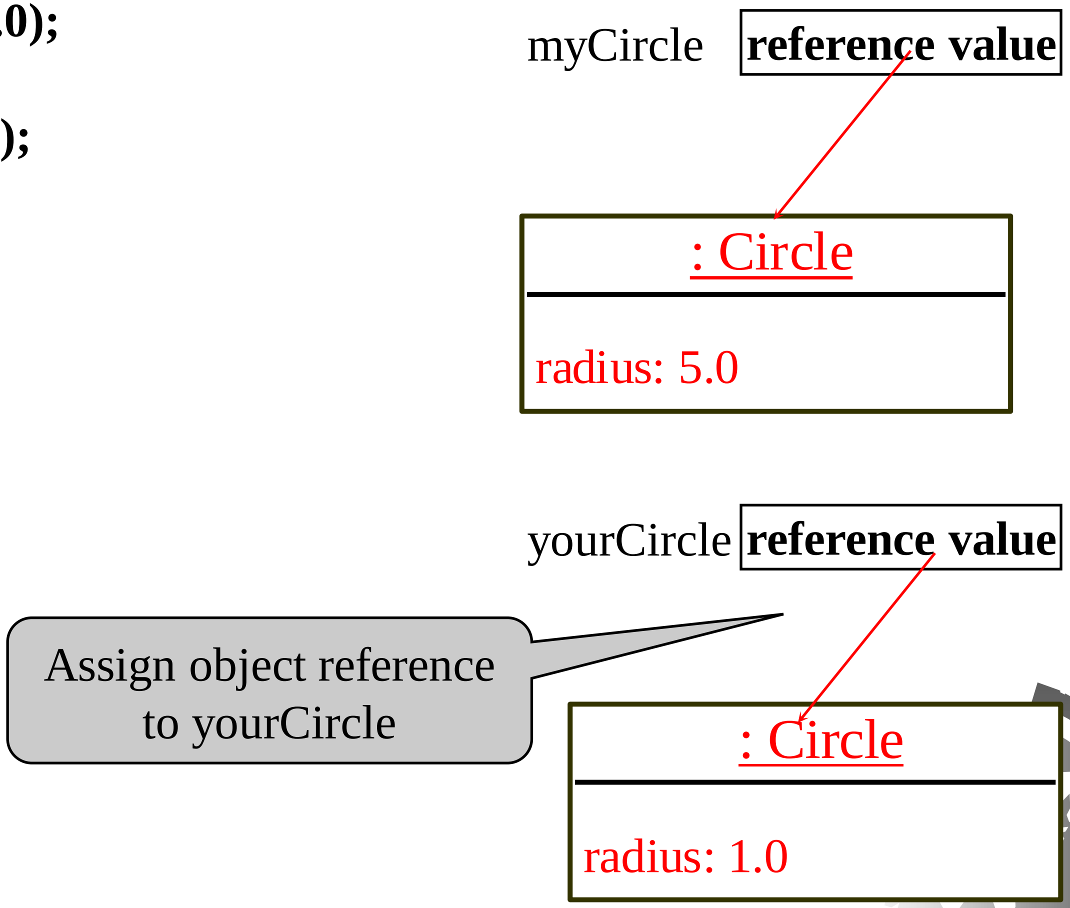
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



myCircle

reference value

: Circle

radius: 5.0

yourCircle

reference value

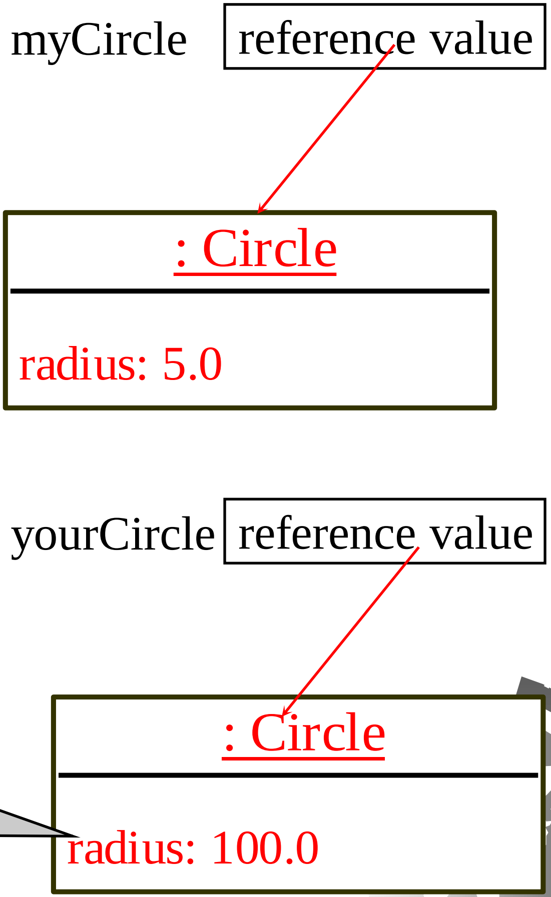
Assign object reference to yourCircle

: Circle

radius: 1.0

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Change radius in yourCircle

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.”

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'
}
```