UNIVERSITÉ Concordia UNIVERSITY

CENTRE FOR CONTINUING EDUCATION

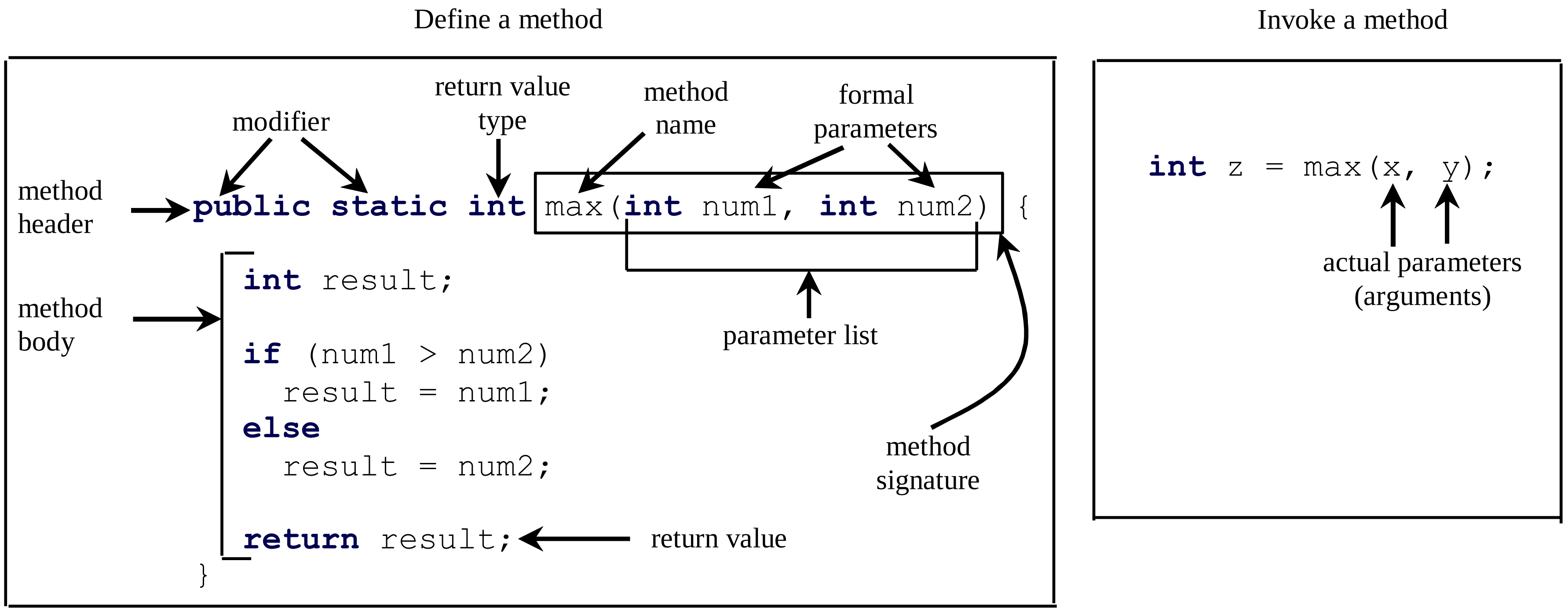
INTRODUCTION TO PROGRAMMING WITH JAVA - CEJV416

Lecture #8

Methods, Classes

Defining Methods

A method is a collection of statements that are grouped together to perform an operation.



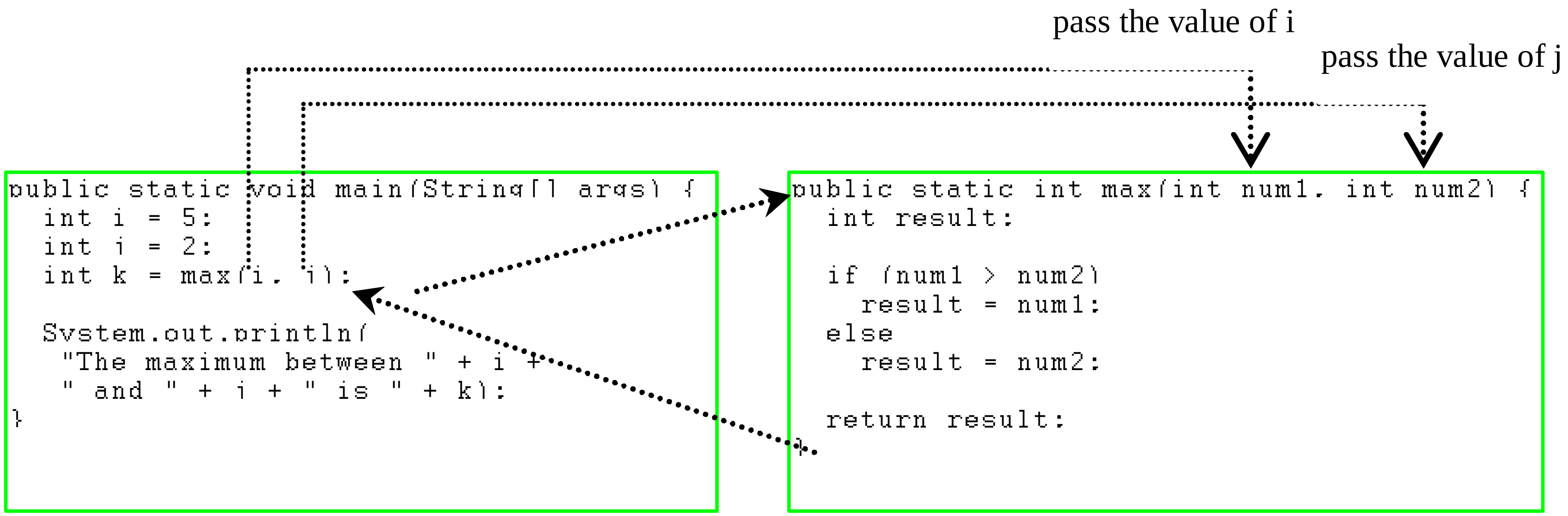
```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

```
int z = max(x, y);
```

Calling Methods, cont.



```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Ambiguous Invocation

Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match. This is referred to as *ambiguous invocation*. Ambiguous invocation is a compilation error.

Ambiguous Invocation

```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2));
  }

  public static double max(int num1, double num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }

  public static double max(double num1, int num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }
}
```

Scope of Local Variables

A local variable: a variable defined inside a method.

Scope: the part of the program where the variable can be referenced.

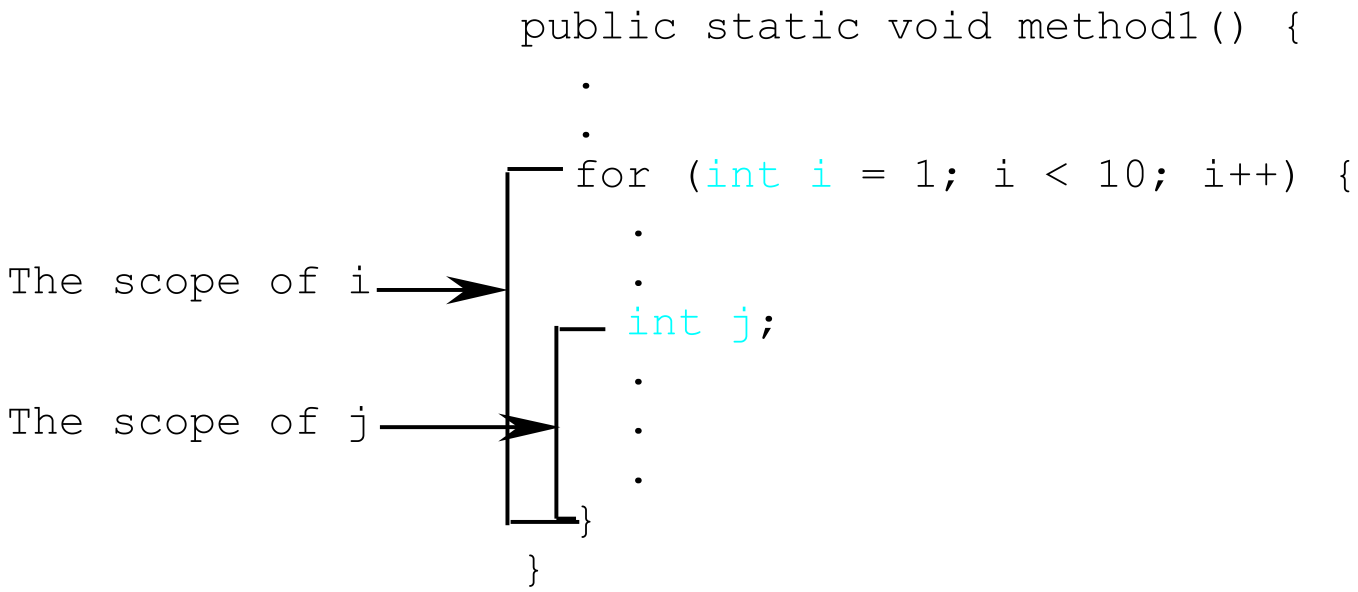
The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.

Scope of Local Variables, cont.

You can declare a local variable with the same name multiple times in different non-nesting blocks in a method, but you cannot declare a local variable twice in nested blocks.

Scope of Local Variables, cont.

A variable declared in the initial action part of a for loop header has its scope in the entire loop. But a variable declared inside a for loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.



```
public static void method1() {
  .
  .
  for (int i = 1; i < 10; i++) {
    .
    .
    int j;
    .
    .
    .
  }
}
```

Scope of Local Variables, cont.

It is fine to declare i in two non-nesting blocks

```
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x += i;
  }

  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

It is wrong to declare i in two nesting blocks

```
 public static void method2() {

   int i = 1;
   int sum = 0;

   for (int i = 1; i < 10; i++)
     sum += i;
   }

 }
```

Scope of Local Variables, cont.

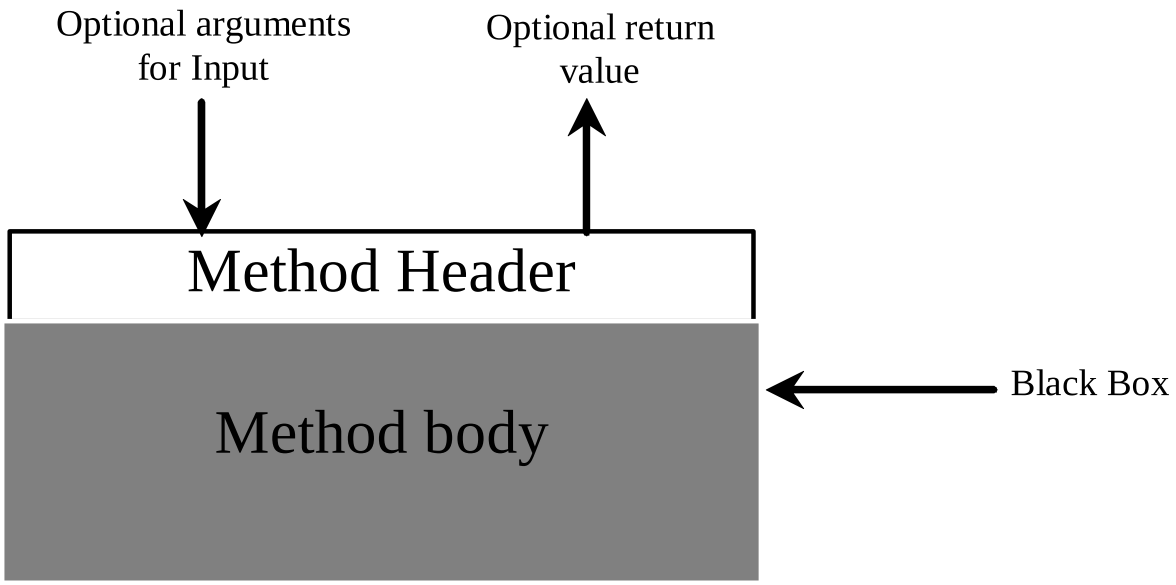
```
// With errors
public static void incorrectMethod() {
  int x = 1;
  int y = 1;
  for (int i = 1; i < 10; i++) {
    int x = 0;
    x += i;
  }
}
```

Scope of Local Variables, cont.

```
// Fine with no errors
public static void correctMethod() {
  int x = 1;
  int y = 1;
  // i is declared
  for (int i = 1; i < 10; i++) {
    x += i;
  }
  // i is declared again
  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

Method Abstraction

You can think of the method body as a black box that contains the detailed implementation for the method.



Benefits of Methods

- Write a method once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

Basic Project

- Open the basic project
  - The main method is where a program starts.
  - While you can write any code that you want in the main method you should not.
  - This method just gets the ball rolling and waits for the program to end.
- From now on, you should use base project as guide line for all futures exercises.

Exercise

- Write a program to print the even and odd numbers.
- Your program prompts the user to provide 2 different integers (one for odd numbers and one for even numbers) and then display the odd or the even numbers less than the inputs by calling the printOdd and printEven methods.
- Your program should have at least 3 methods:
  - getting the input and check if the inputs are valid
  - printOdd method (public void printOdd(int odd))
  - printEven method (public void printEven(int even))

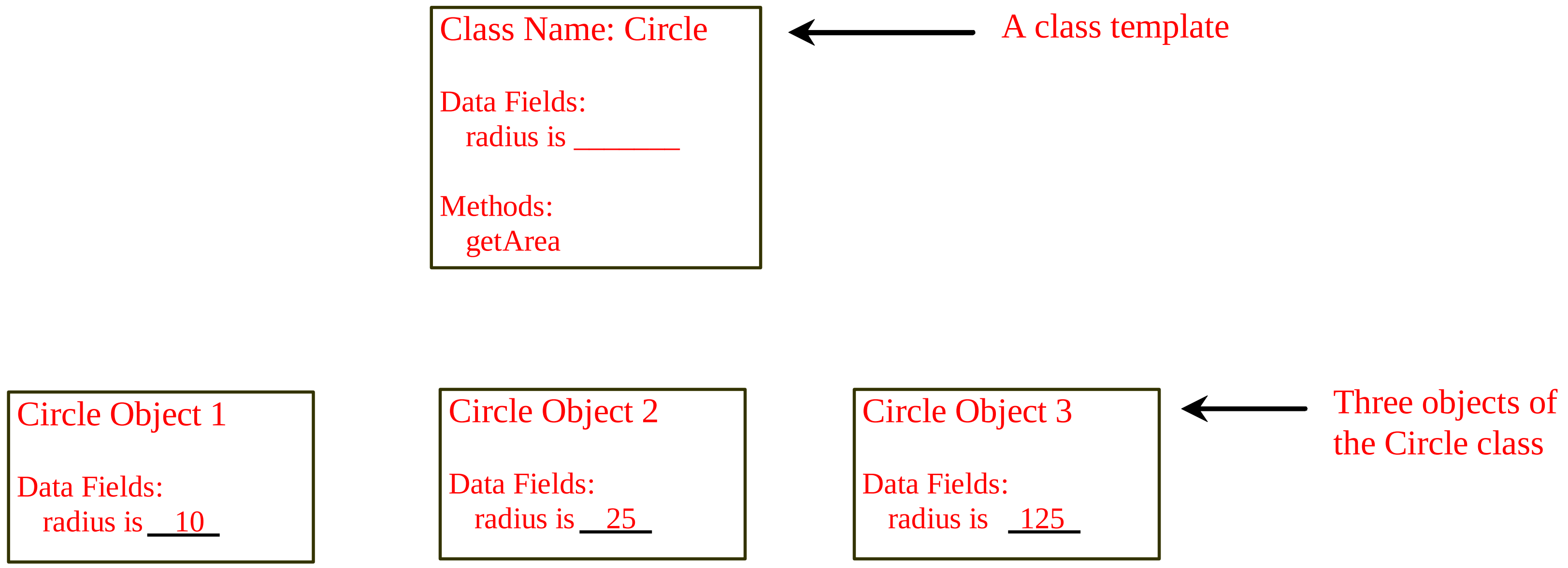
OO Programming Concepts

- 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.

The Class

- Creating abstract data types (classes) is a fundamental concept in object-oriented programming.
- What we really do in object-oriented programming is create new data types.
- When you see the word ‘class’ think ‘type’ and vice versa.
- A class describes a set of objects that have identical characteristics (data elements) and behaviours (functionality).

Objects



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

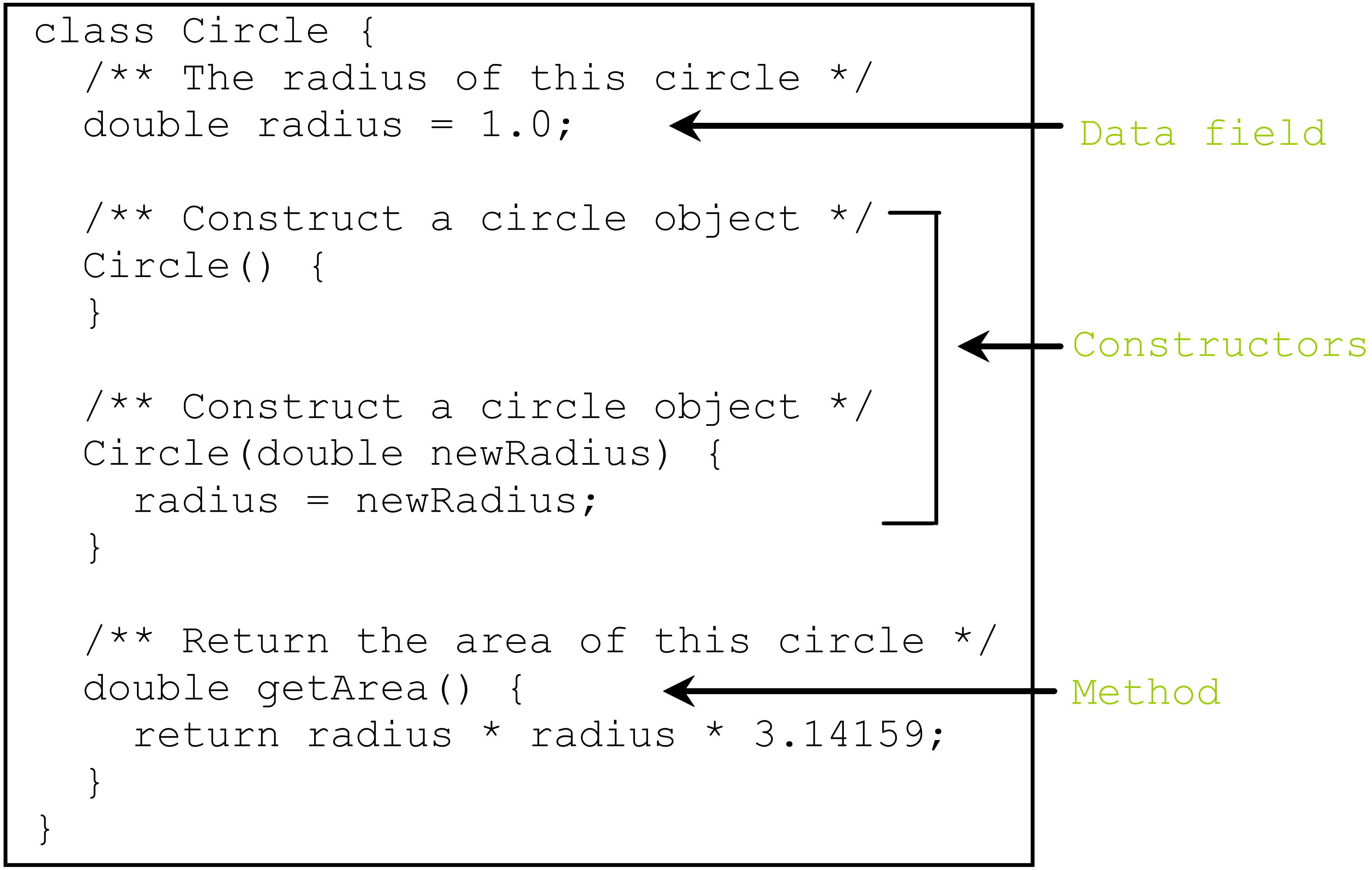
Classes

•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



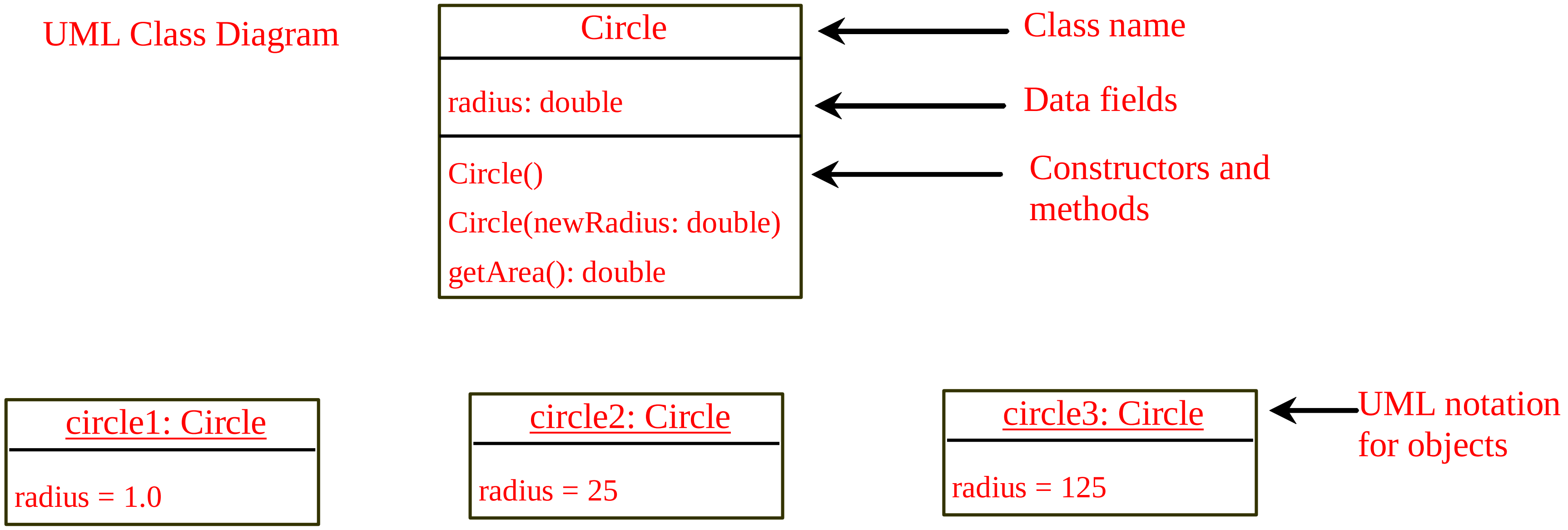
```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** 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



```
// Define the circle class with two constructors
class SimpleCircle {
  double radius;

  /** Construct a circle with radius 1 */
  SimpleCircle() {
    radius = 1;
  }

  /** Construct a circle with a specified radius */
  SimpleCircle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * Math.PI;
  }

  /** Return the perimeter of this circle */
  double getPerimeter() {
    return 2 * radius * Math.PI;
  }

  /** Set a new radius for this circle */
  void setRadius(double newRadius) {
    radius = newRadius;
  }
}
```

Example: Defining Classes and Creating Objects

```
public class TestSimpleCircle {
  /** Main method */
  public static void main(String[] args) {
    // Create a circle with radius 1
    SimpleCircle circle1 = new SimpleCircle();
    System.out.println("The area of the circle of radius "
      + circle1.radius + " is " + circle1.getArea());

    // Create a circle with radius 25
    SimpleCircle circle2 = new SimpleCircle(25);
    System.out.println("The area of the circle of radius "
      + circle2.radius + " is " + circle2.getArea());

    // Create a circle with radius 125
    SimpleCircle circle3 = new SimpleCircle(125);
    System.out.println("The area of the circle of radius "
      + circle3.radius + " is " + circle3.getArea());

    // Modify circle radius
    circle2.radius = 100; // or circle2.setRadius(100)
    System.out.println("The area of the circle of radius "
      + circle2.radius + " is " + circle2.getArea());
  }
}
```

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);
```

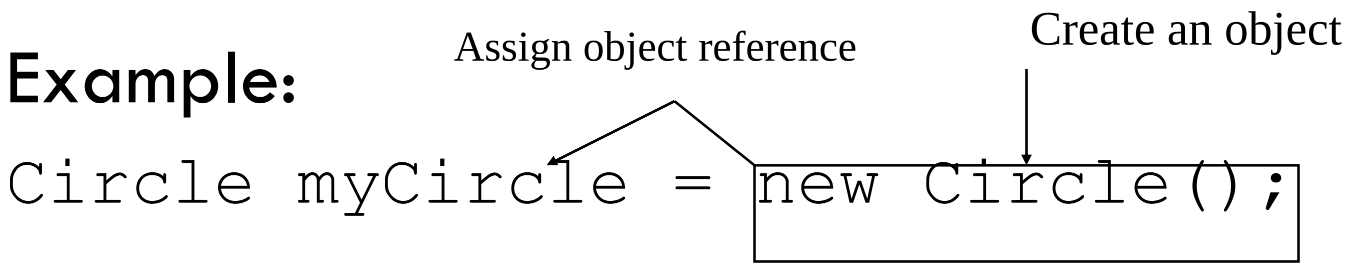
Default Constructor

- A class may be defined 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 defined in the class.

Declaring/Creating Objects in a Single Step

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

Example:



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

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()`

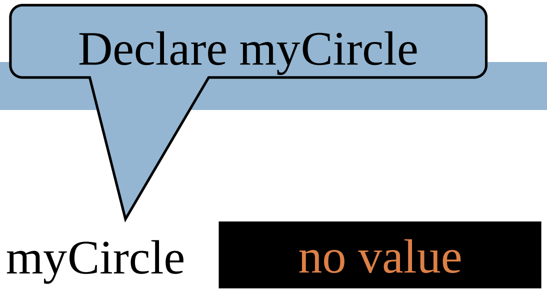
animation

Trace Code

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

SCircle yourCircle = new Circle();

yourCircle.radius = 100;
```



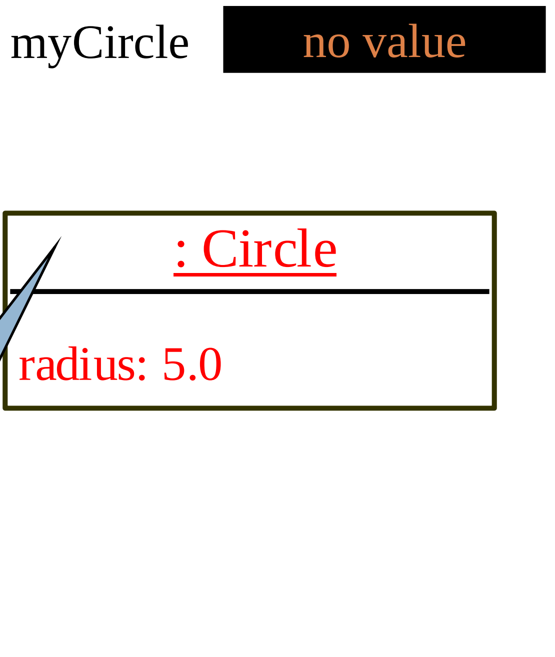
animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

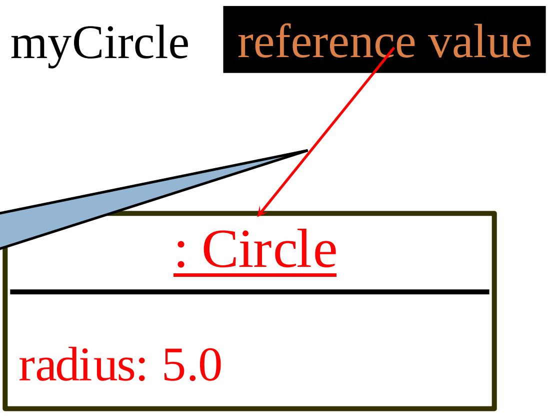
Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

Assign object reference to myCircle



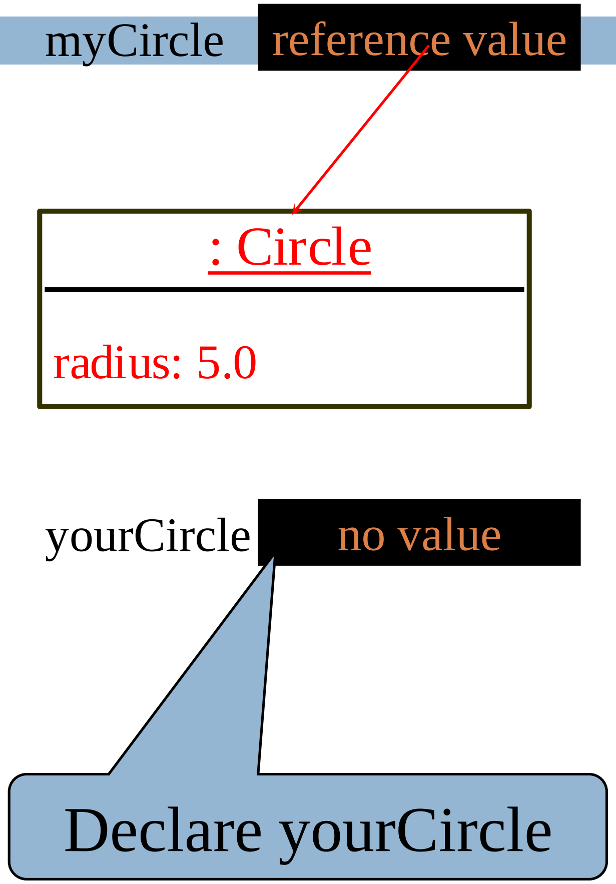
animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



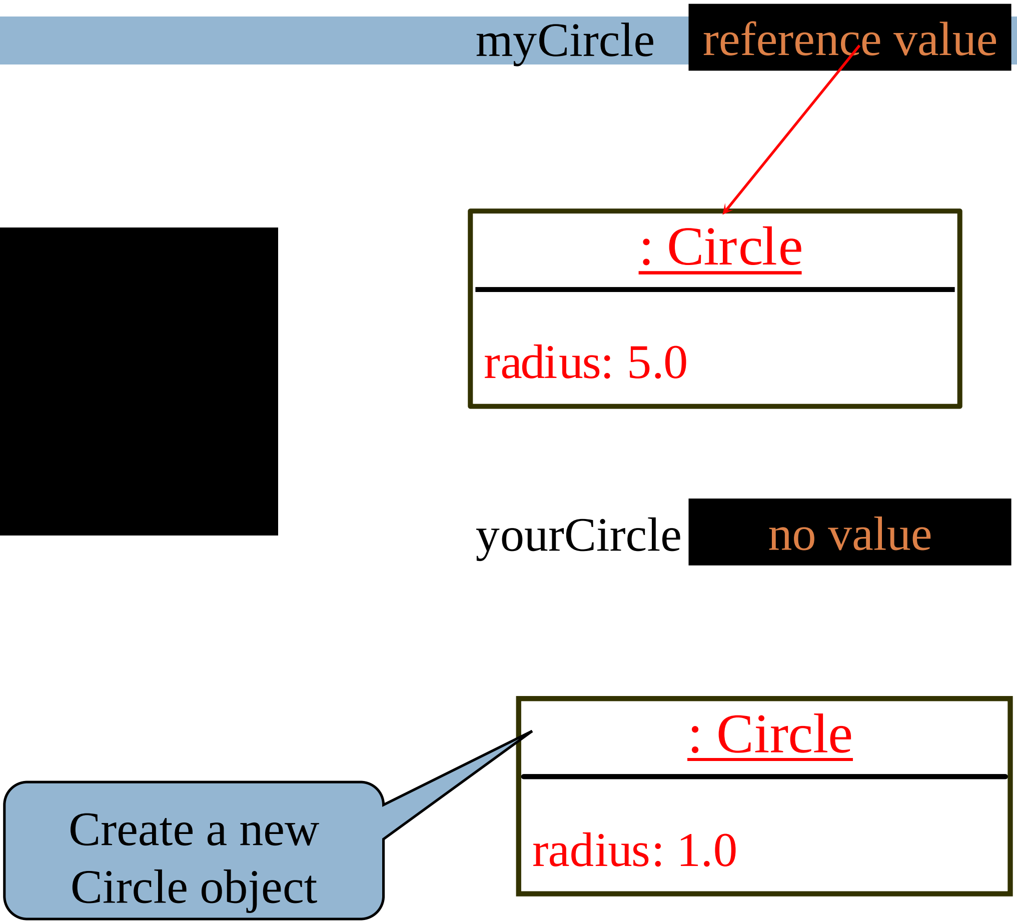
animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



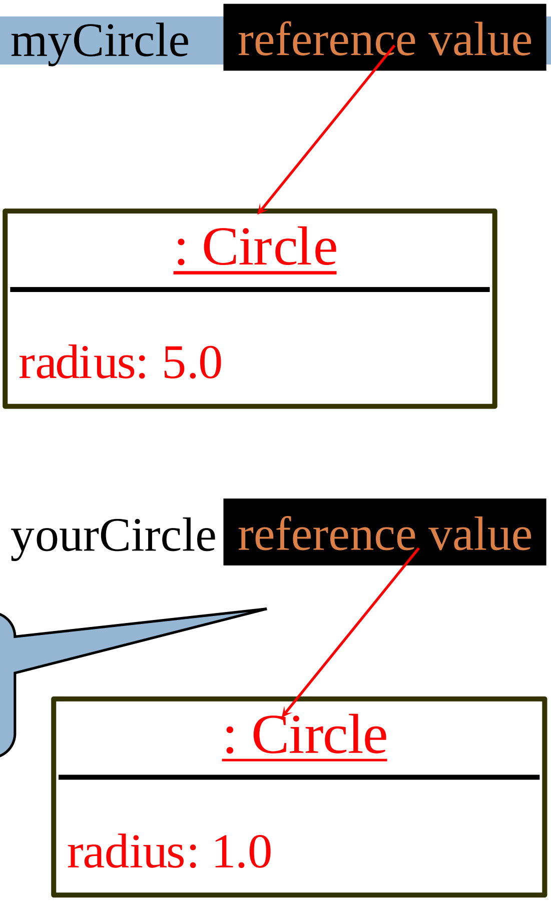
animation

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



Assign object reference to yourCircle

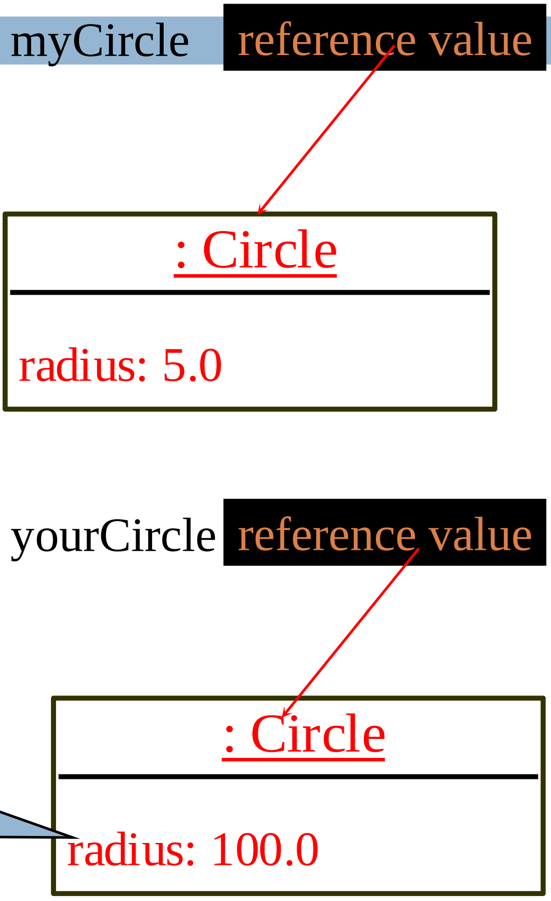
animation

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 Circle.getArea()?

The answer is ***no***, because getArea() is non-static. It must be invoked from an object using

objectRefVar.methodName(arguments) (e.g., myCircle.getArea()).

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.

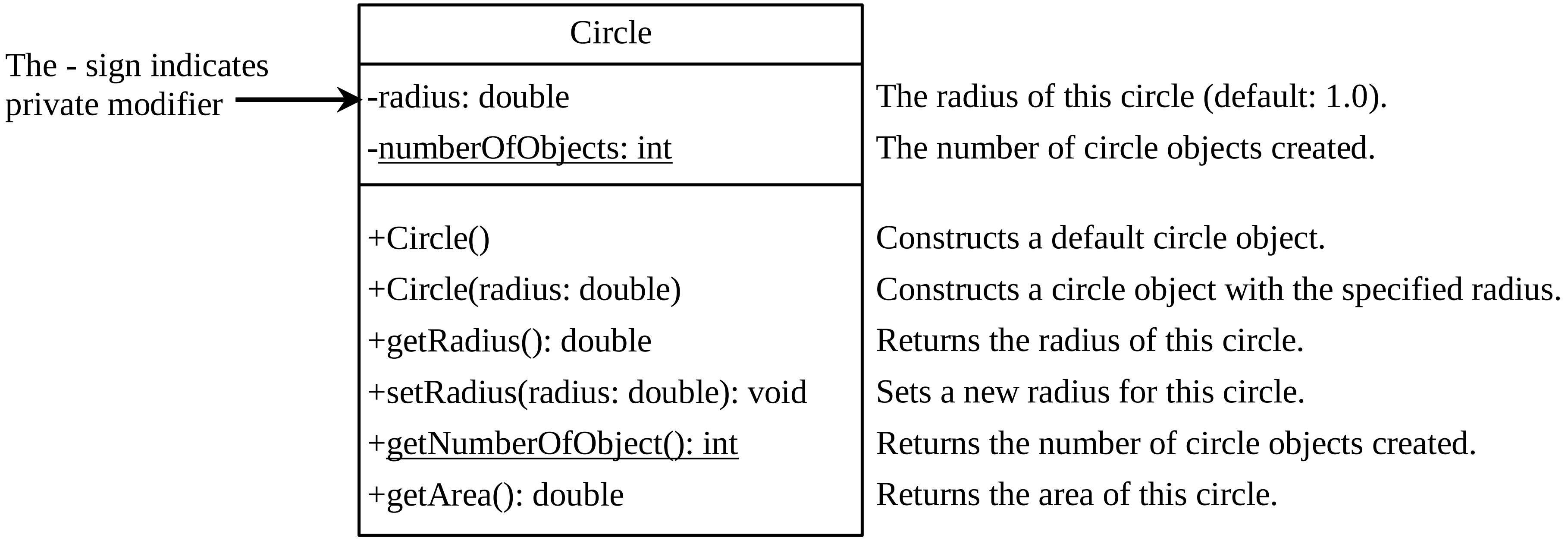
The get and set methods are used to read and modify private properties.

Why Data Fields Should Be private?

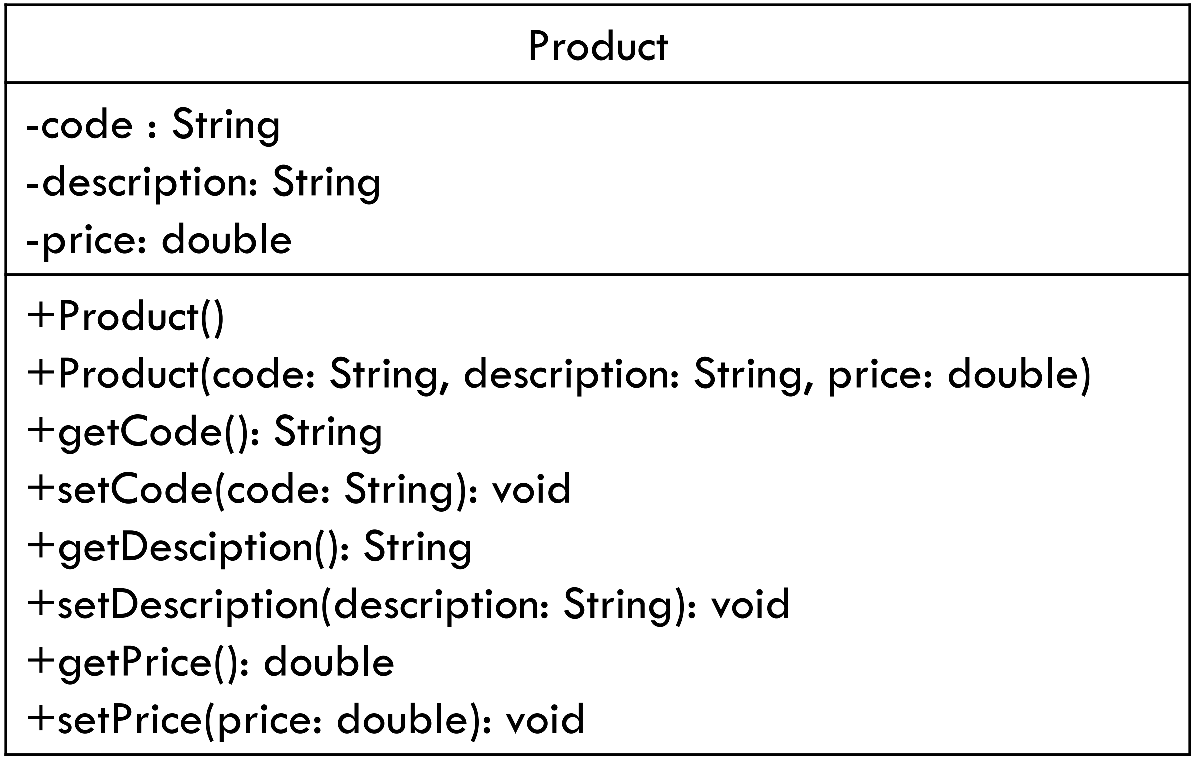
To protect data.

To make class easy to maintain.

Example of Data Field Encapsulation


The - sign indicates private modifier
Circle
-radius: double
-numberOfObjects: int
+Circle()
+Circle(radius: double)
+getRadius(): double
+setRadius(radius: double): void
+getNumberOfObject(): int
+getArea(): double
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.

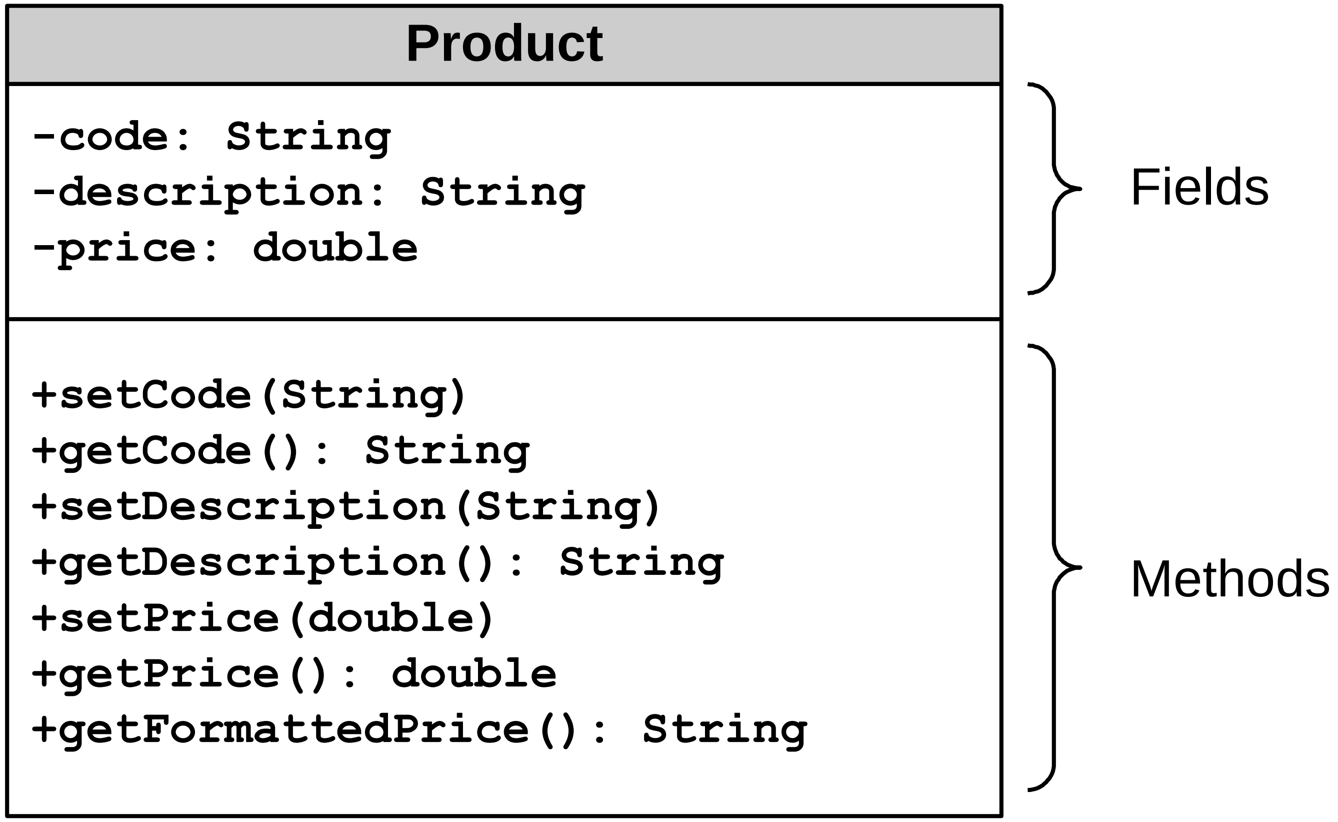
Class diagram for the Product class



Product

-code : String
-description: String
-price: double

+Product()
+Product(code: String, description: String, price: double)
+getCode(): String
+setCode(code: String): void
+getDesciption(): String
+setDescription(description: String): void
+getPrice(): double
+setPrice(price: double): void


Product
-code: String
-description: String
-price: double
+setCode(String)
+getCode(): String
+setDescription(String)
+getDescription(): String
+setPrice(double)
+getPrice(): double
+getFormattedPrice(): String
Fields
Methods