Introduction to Java for C++ Programmers
JAC444

Week 04
Objects & Classes

Instructor

Hossein Pourmodheji

hossein.pourmodheji@senecacollege.ca

What are Objects?

- Object-oriented programming (OOP) involves programming using objects.
- An **object** represents an entity in the real world that can be distinctly identified
- Objects are reusable software components that model real world items.
- Humans think in terms of objects, for instance an animal, a plant, a car, 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 behavior.

Object State

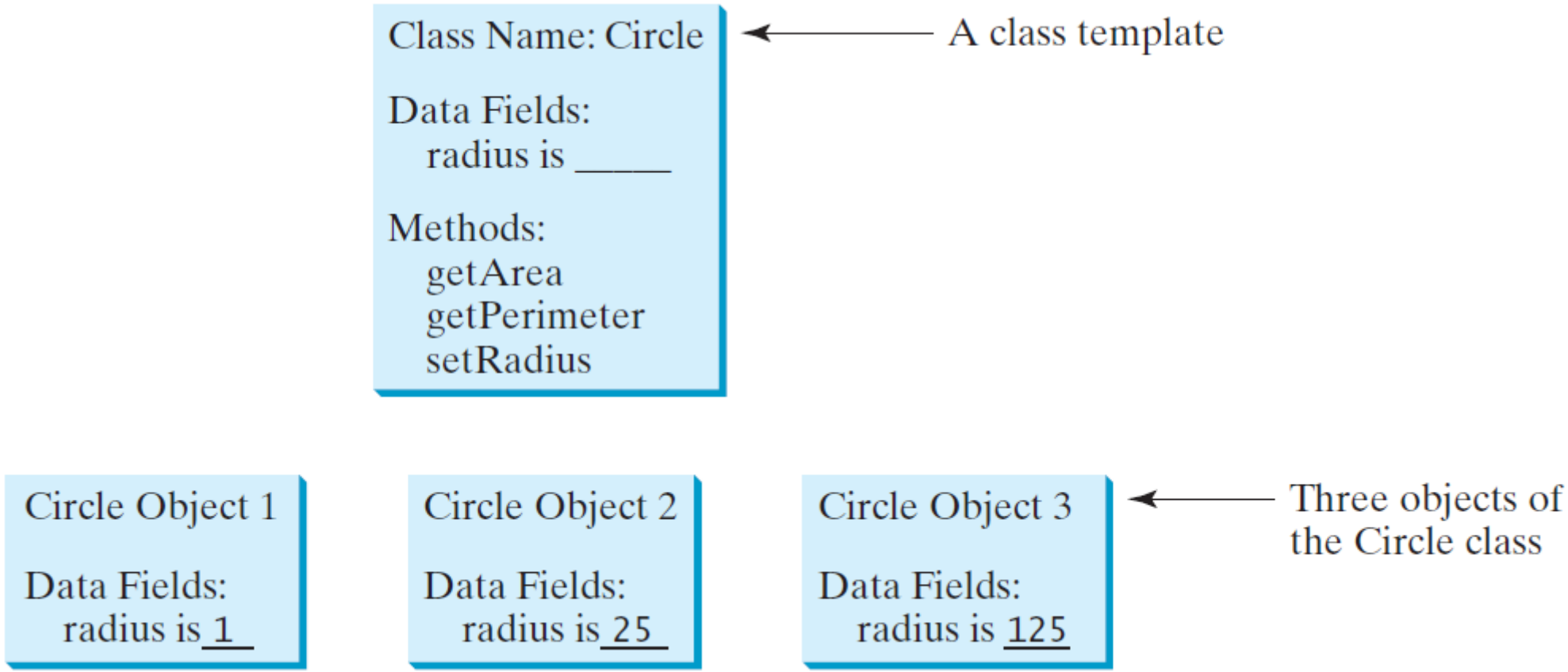
- The **state** of an object (also known as its properties or attributes) is represented by data fields with their current values. (e.g., size, shape, color and weight)
- A circle object, for example, has a data field **radius**, which is the property that characterizes a circle.
- A rectangle object has the data fields **width** and **height**, which are the properties that characterize a rectangle.

Object Behavior

- The **behavior** of an object (also known as its **actions**) is defined by **methods**.
- To invoke a method on an object is to ask the object to perform an action.
- For example, you may define methods named **getArea()** and **getPerimeter()** for circle objects.
- A circle object may invoke **getArea()** to return its area and **getPerimeter()** to return its perimeter.
- You may also define the **setRadius(radius)** method. A circle object can invoke this method to change its radius.

Defining Classes for Objects

- Objects of the same type are defined using a common class.
- A **class** is a template, blueprint, or **contract** that defines what an object's data fields and methods will be.
- Additionally, a class provides methods of a special type, known as **constructors**, which are invoked to create a **new object**.
- An object is an **instance** of a class. You can create many instances of a class. Creating an instance is referred to as **instantiation**. The terms object and instance are often interchangeable.
- The relationship between classes and objects is analogous to that between an apple-pie recipe and apple pies:
  - You can make as many apple pies as you want from a single recipe.


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

```java
class Circle {
  /** The radius of this circle */
  double radius = 1;

  /** 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 * 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;
  }
}
```

Data fields

Constructors

Methods

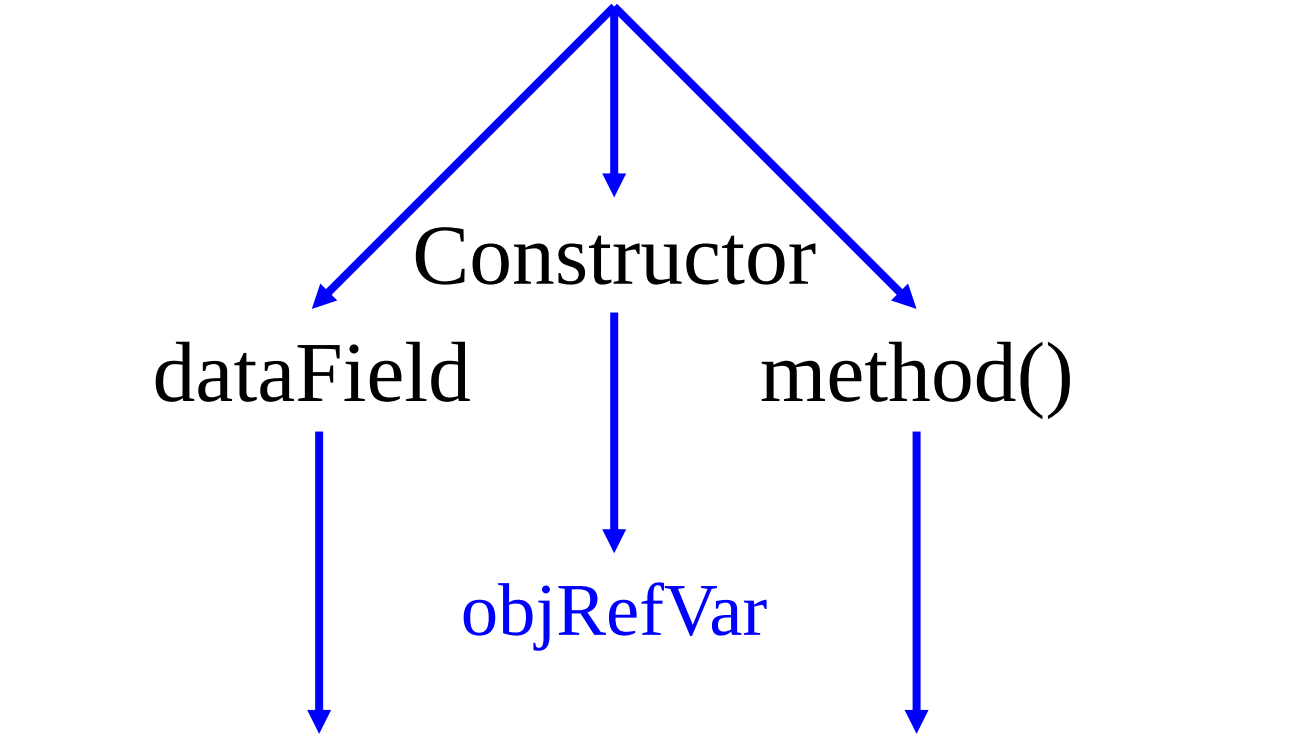
Constructing Objects Using Constructors

- A constructor is invoked to create an object using the new operator.
  - A constructor 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.

```
ClassName objRefVar;
objRefVar = new ClassName();

ClassName objRefVar = new ClassName();
```

class members



Accessing Objects via Reference Variables

- An object's data and methods can be accessed through the dot (.) operator via the object's reference variable.
  - **`objectRefVar.dataField`** references a data field in the object.
  - **`objectRefVar.method(arguments)`** invokes a method on the object.

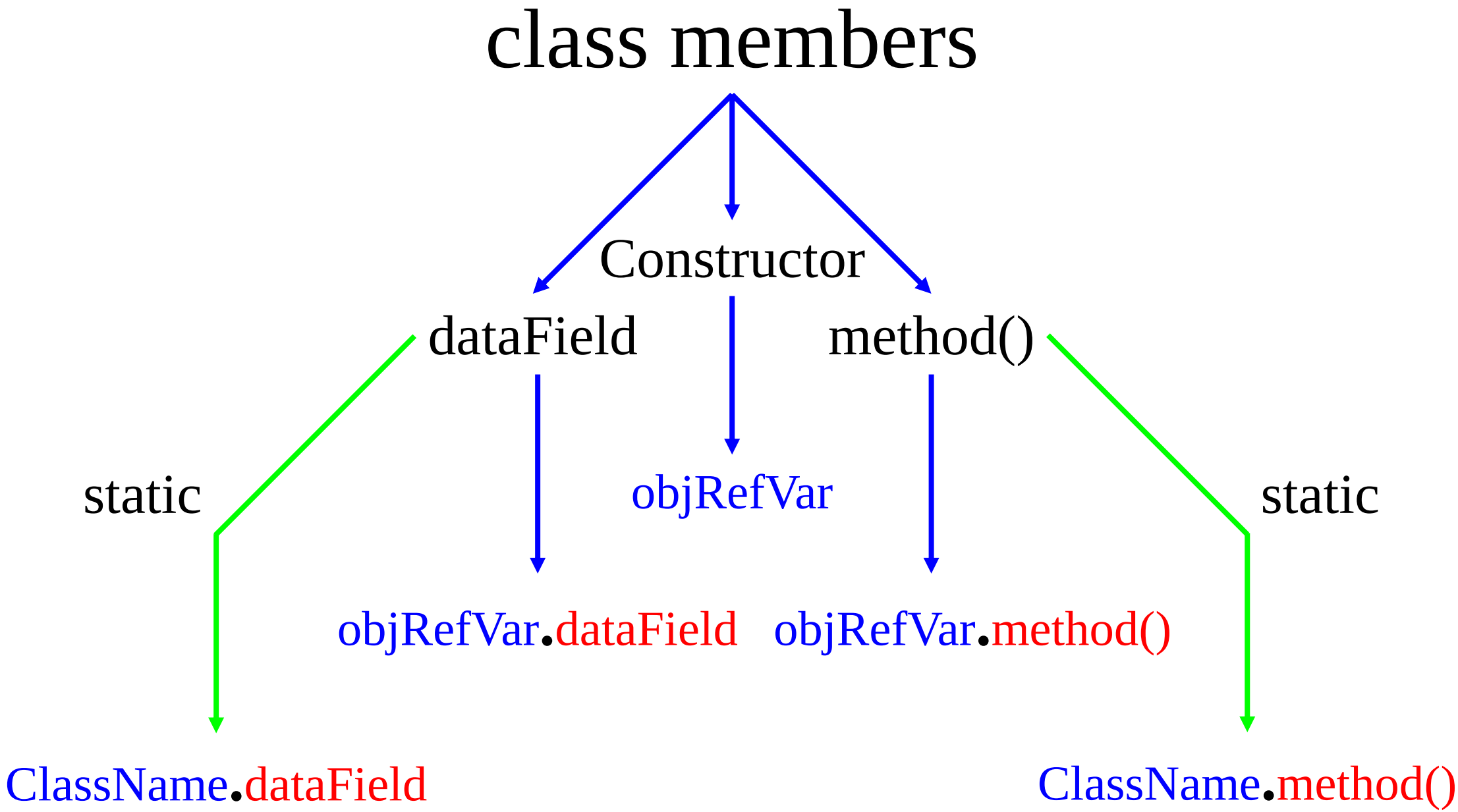
```
Scanner input = new Scanner(System.in);
int i = input.nextInt();
```

Import Declaration

- Helps the compiler locate a class that is used in this program.
- Rich set of predefined classes that you can reuse rather than "reinventing the wheel."
- Classes are grouped into packages—named groups of related classes—and are collectively referred to as the Java class library, or the Java Application Programming Interface (Java API).
- You use keyword import to identify the predefined classes used in a Java program.

Static Variables and Methods

- A static variable is shared by **all** objects of the class. A static method **cannot** access instance members of the class.
- If you want all the instances of a class to share data, use static variables, also known as class variables.
  - `ClassName.dataField` references a static data field in the objects.
- Static methods can be called **without** creating an instance of the class.
  - `ClassName.method(arguments)` invokes a static method in the class.
- To declare a static variable or define a static method, put the modifier **static** in the variable or method declaration.


class members
Constructor
dataField
method()
static
objRefVar
static
objRefVar.dataField
objRefVar.method()
ClassName.dataField
ClassName.method()

Visibility Modifiers

- Visibility modifiers can be used to specify the visibility of a class and its members.
- You can use the **public** visibility modifier for classes, methods, and data fields to denote that they can be **accessed** from any other classes.
- The **private** modifier makes methods and data fields accessible only from within its own class.
- If no visibility modifier is used, then by default the classes, methods, and data fields are accessible by any class in the same package. This is known as **package-private** or **package-access**.
- To **prevent** direct modifications of data fields, you should declare the data fields private, using the **private** modifier. This is known as **data field encapsulation**.

CAUTION

- A return statement is required for a value-returning method. The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it's possible that this method does not return any value.

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else if (n < 0)
    return -1;
}
```

(a)

Should be

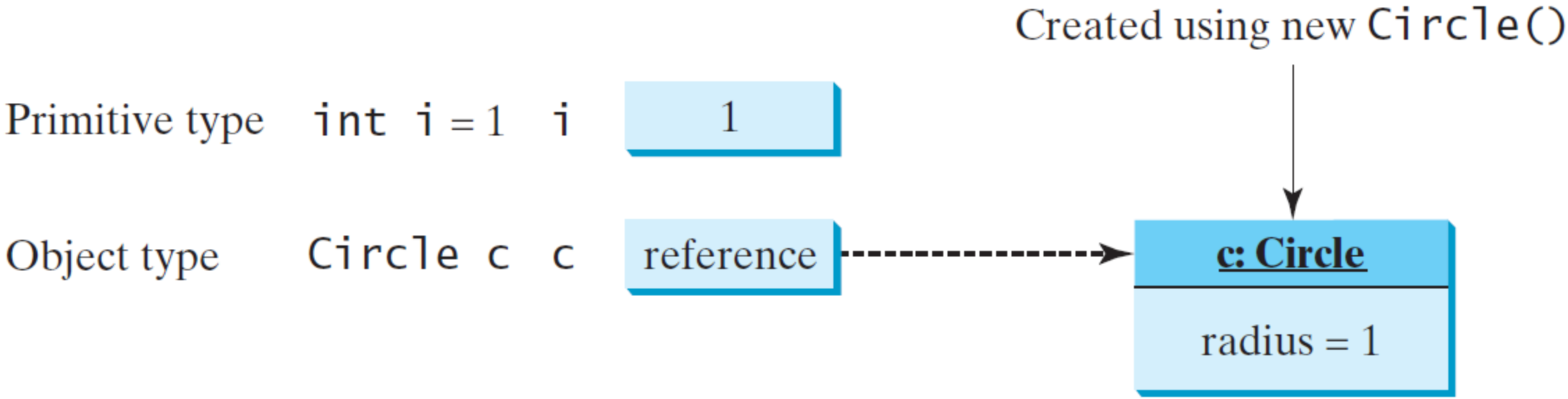
```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else
    return -1;
}
```

(b)

- To fix this problem, delete **if (n < 0)** in (a), so that the compiler will see a return statement to be reached regardless of how the if statement is evaluated.

Differences between Variables of Primitive Types and Reference Types

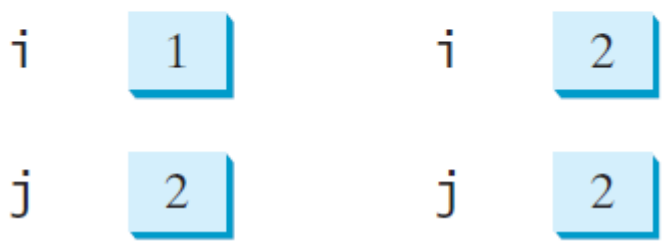
- Every variable represents a memory location that holds a value. When you declare a variable, you are telling the compiler what type of value the variable can hold.
- For a variable of a primitive type, the value is of the primitive type.
- For a variable of a reference type, the value is a reference to where an object is located.
- For example the value of **int** variable **i** is **int** value **1**, and the value of **Circle** object **c** holds a reference to where the contents of the **Circle** object are stored in memory.


Created using new Circle()
Primitive type
int i = 1
i
1
Object type
Circle c
c
reference
c: Circle
radius = 1

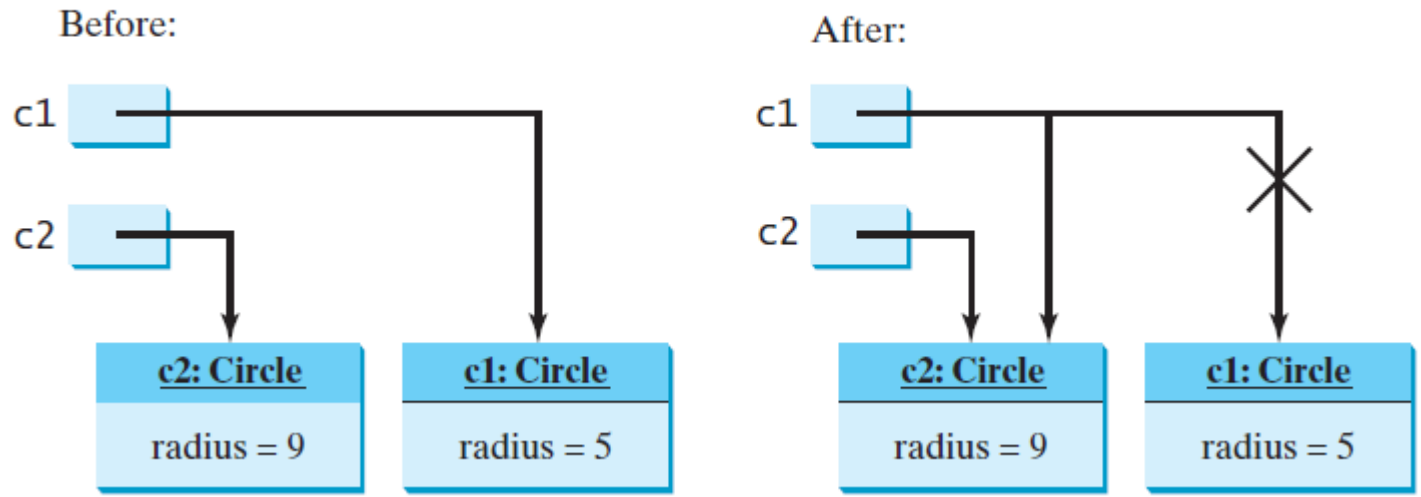
Differences between Variables of Primitive Types and Reference Types (Cont.)

- When you assign one variable to another, the other variable is set to the same value.
- For a variable of a primitive type, the real value of one variable is assigned to the other variable.
- For a variable of a reference type, the reference of one variable is assigned to the other variable.

Primitive type assignment `i = j`



Object type assignment `c1 = c2`



Analyzing Our First Java Program

- What is System?
  - **class**
- What is System.out?
  - **PrintStream object**
  - Standard output object.
  - Allows Java applications to display strings in the command window from which the Java application executes.
- What about System.out.println()?!
  - **A method within PrintStream class**