CIS2571 – Intro to Java

Chapter8 → Objects and Classes

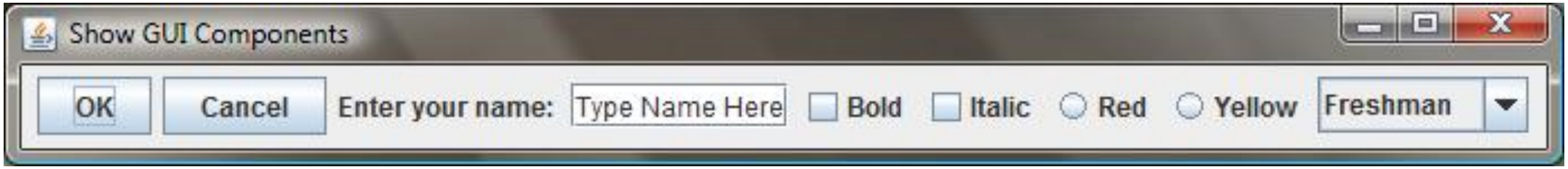
Information consolidated from a variety of textbook and online resources

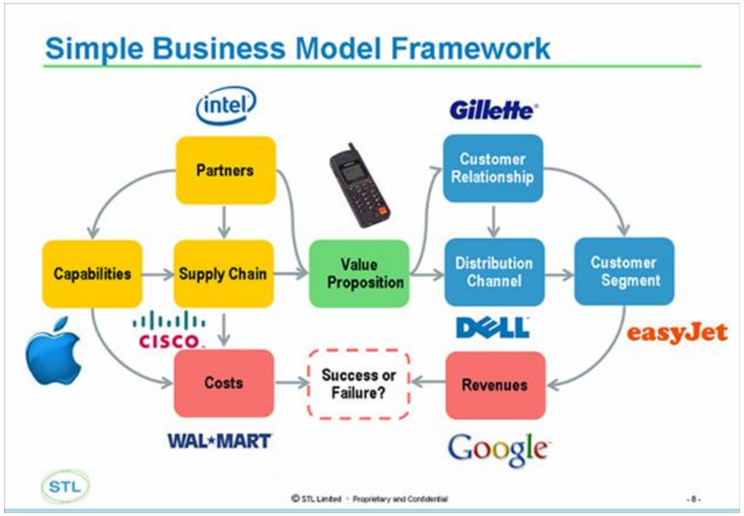
Topic Objectives

- OOP, Objects, Classes, and Packages
- Constructor Methods
- Using Objects
- Class Caution
- Static Variables, Constants, and Methods
- Visibility Modifiers
- Data Field Encapsulation
  - Private Data Members
- Passing Objects to Methods
- Array of Objects
- Java Library Classes

Why use classes?

- Development of GUI (graphical user interface) and large scale software systems require a level of modularity not achievable with methods alone





OOP Fundamentals

OOP, Objects, and Classes

- OOP (object-oriented programming) involves programming with objects
- Objects represent real-world entities and consist of
  - State (aka properties or attributes) represented by data fields with specific values
  - Behavior (aka actions) defined by methods
- Objects of **same** type are defined with a **common** class
- Class is a template, blueprint, or contract that defines
  - Fields
  - Methods
    - **Constructor** is special type of method used to create and initialize objects of the class type
- An object, or instance, is a variable of the class type

Packages

Supplement III G Packages

- Packages are
  - used to group classes
  - directories of Java class bytecode
  - hierarchical and Java expects a one-to-one mapping of package name and file system directory structure
- **Every** Java class belongs to package
  - Class added to package upon compilation
  - Current directory is default package
- All classes have access to the java.lang package without importing

```
int number = (int)(Math.random() * 2);
```

Packages

- To access another package's class and/or methods:
  - Use import statement
    - All classes in package

      ```
      import javax.swing.*;
      ```

    - Only single class in package

      ```
      import javax.swing.JOptionPane;
      ```

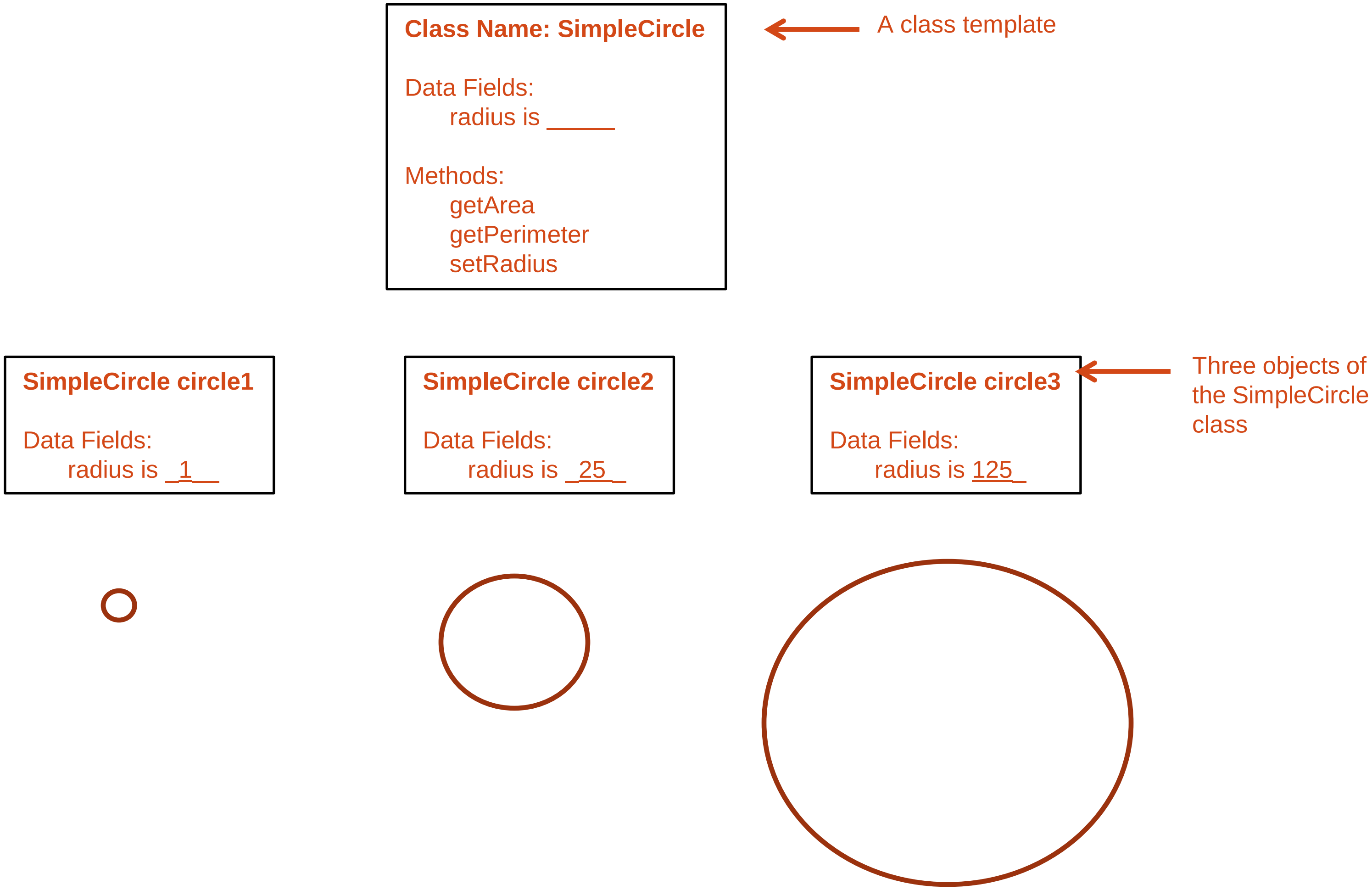
    - Access class method

      ```
      JOptionPane.showMessageDialog(null, "Welcome
        to Java!");
      ```

  - Use full class path:

    ```
    javax.swing.JOptionPane.showMessageDialog(null,
      "Welcome to Java!");
    ```

Classes and Objects Example



Class Name: SimpleCircle

Data Fields:
radius is ______

Methods:
getArea
getPerimeter
setRadius

A class template

SimpleCircle circle1

Data Fields:
radius is _1__

SimpleCircle circle2

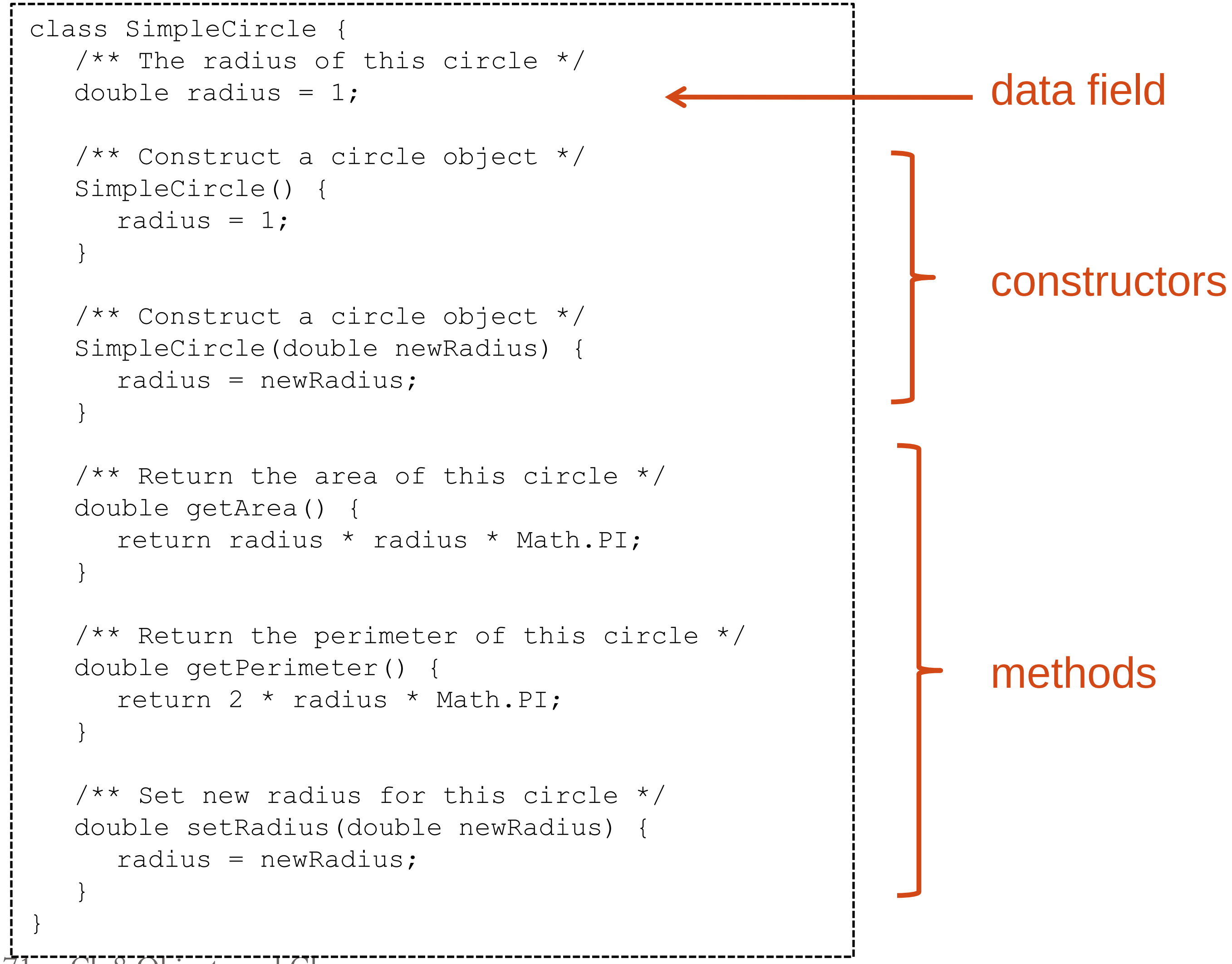
Data Fields:
radius is _25_

SimpleCircle circle3

Data Fields:
radius is 125_

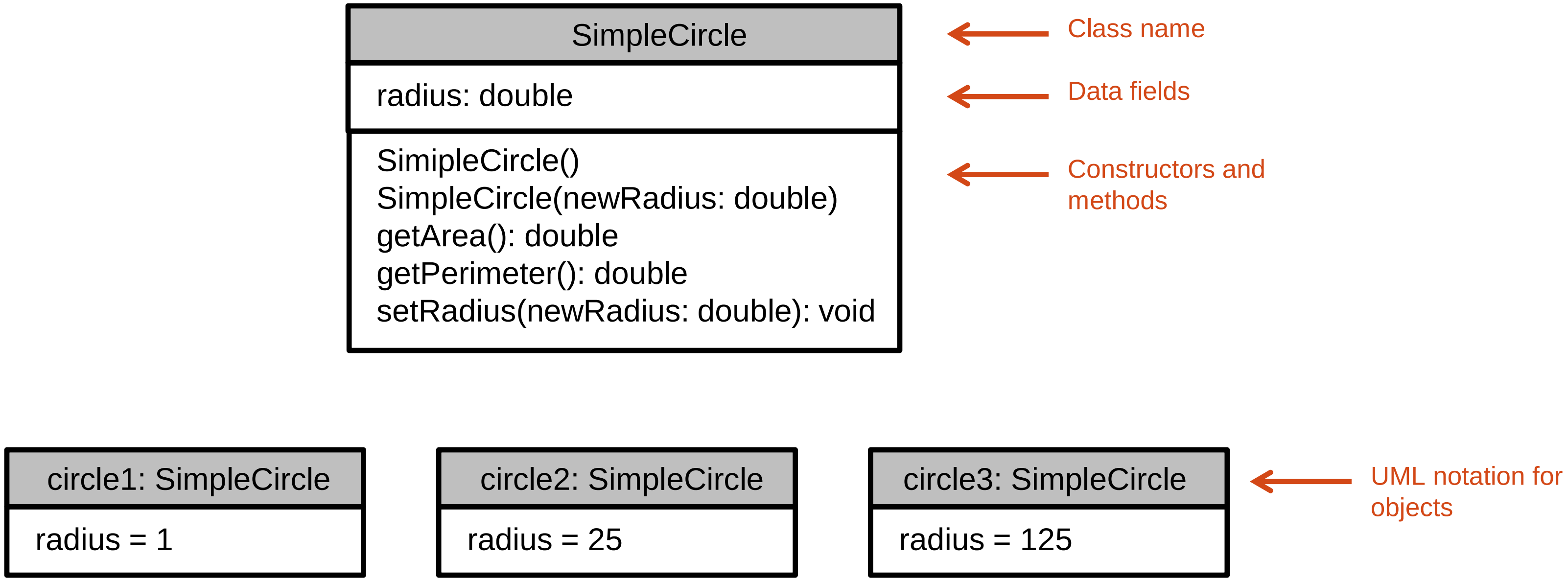
Three objects of the SimpleCircle class

Classes and Objects Example



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

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

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

Classes and Objects Example



SimpleCircle

radius: double

SimipleCircle()
SimpleCircle(newRadius: double)
getArea(): double
getPerimeter(): double
setRadius(newRadius: double): void

Class name

Data fields

Constructors and methods

circle1: SimpleCircle

radius = 1

circle2: SimpleCircle

radius = 25

circle3: SimpleCircle

radius = 125

UML notation for objects

UML Notation

dataFieldName: dataFieldType
methodName(parameterName: parameterType): returnType

Classes and Objects Example

```
public class TestSimpleCircle {
  /** Main method */
  public static void main(String[] args) {

    // create a circle with radius 1.0
    SimpleCircle circle1 = new SimpleCircle();
    System.out.println(“The area of the circle of radius ”
      + circle1.radius + “ is ” circle1.getArea());

    // create a circle with radius 25.0
    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 125.0
    SimpleCircle circle3 = new SimpleCircle(125);
    System.out.println(“The area of the circle of radius ”
      + circle3.radius + “ is ” circle3.getArea());
  }
}
```

client of SimpleCircle class

(main class contains main method)

See 8.1 TestSimpleCircle.java
See 8.2 SimpleCircle.java

Constructor Methods

- Constructors are special class methods
  - Must have **same name as class**
  - Do not have a return type (not even void)
  - Invoked using new operator when object is created
    - Used to initialize objects
- No-arg, or no-argument, constructor is constructor without parameters
- When class defined without explicit constructors
  - No-arg constructor automatically created
  - Also known as default constructor

Using Objects

- Declaring object reference variable

```
ClassName objectRefVar;
SimpleCircle circle1;
```

- Creating object and assigning object reference variable

```
ClassName objectRefVar = new ClassName();
SimpleCircle circle1 = new SimpleCircle();
```

- Dot operator (.), aka object member access operator
- Accessing object’s data (aka instance variable)

```
objectRefVar.dataField
circle1.radius
```

- Invoking object’s method (aka instance method)

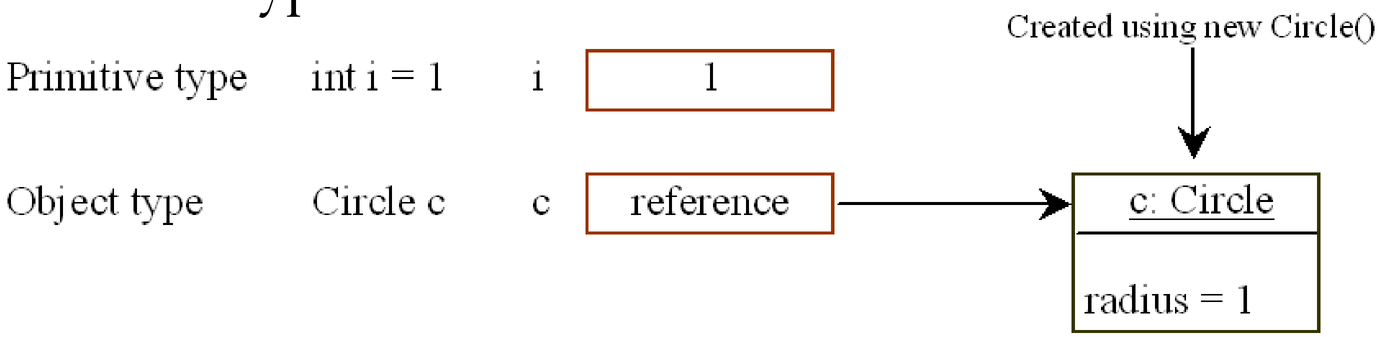
```
objectRefVar.methodName(arguments)
circle1.getArea() // circle1 is calling object
```

Class Cautions

- Use of reference types for data fields

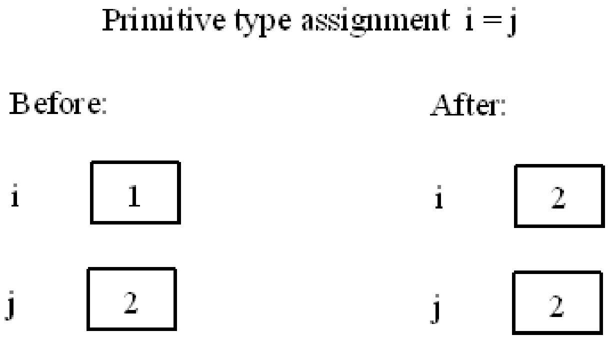
```
public class Student {
   String name;  // name has default value null
   int age;  // age has default value 0
   boolean isScienceMajor;  //isSciencemajor has default value false
   char gender;  // gender has default value ‘\u0000’ (nul)
}
```

- Differences between variables of primitive data types and reference types



Class Cautions

- Copying variables of primitive types



Primitive type assignment i = j

Before:

i 1

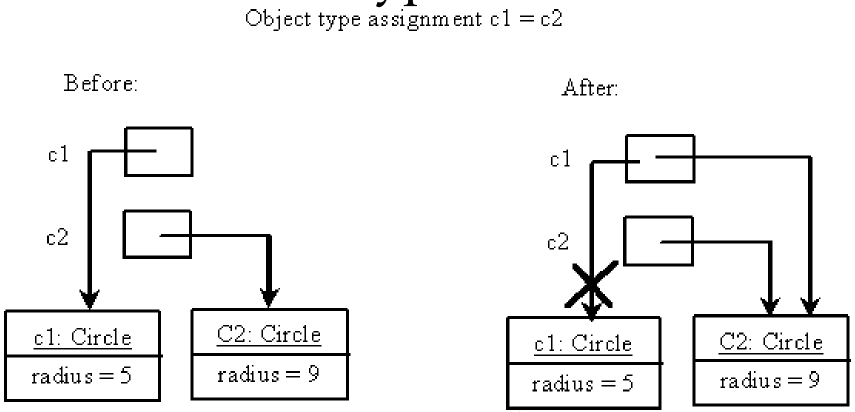
j 2

After:

i 2

j 2

- Copying variables of reference types



Object type assignment c1 = c2

Before:

c1

c2

c1: Circle
radius = 5

C2: Circle
radius = 9

After:

c1

c2

c1: Circle
radius = 5

C2: Circle
radius = 9

JVM
Garbage Collection

Static Variables, Constants, and Methods

- Instance variables/methods are tied to an instance of the class
- Static, or class, variables are shared by all instances
- Static methods can be invoked without a created object
  - Class name is used

```
Math.random()
JOptionPane.showMessageDialog(null,
  "Welcome to Java!");
```

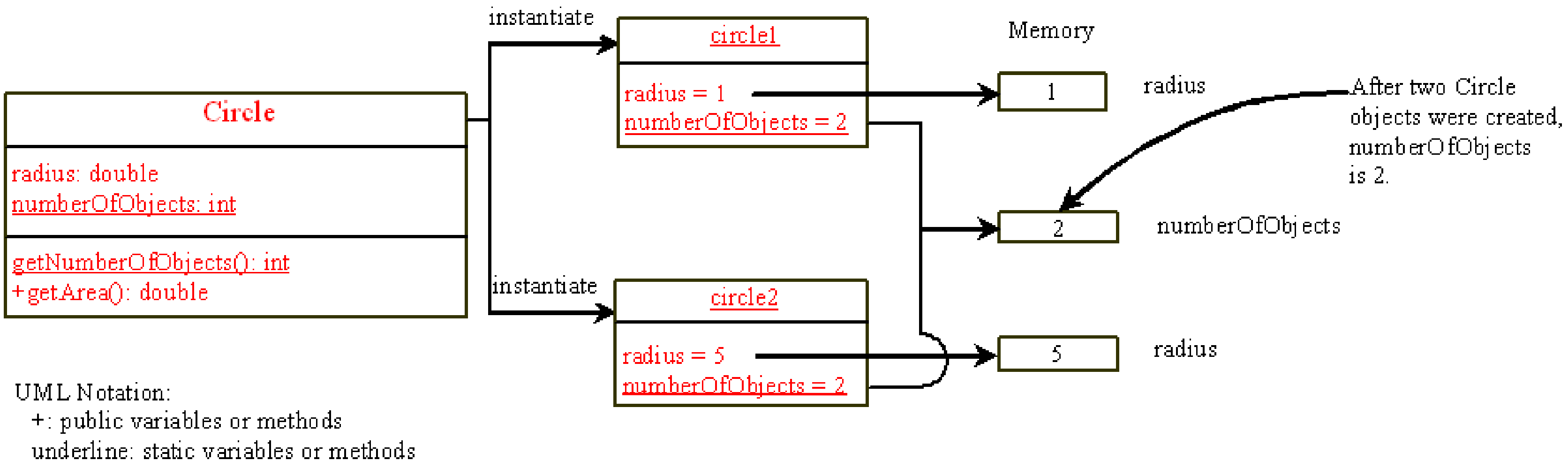
- Modifier, static, should be placed before variable or method declaration

```
static int numberOfObjects;
static int getNumberObjects() {
   return numberOfObjects;
}
```

- Constants should have static modifier after final

```
final static double PI = 3.141592653589;
```

Static Variables, Constants, and Methods Example



See 8.7 CircleWithStaticMembers.java
See 8.8 TestCircleWithStaticMembers.java

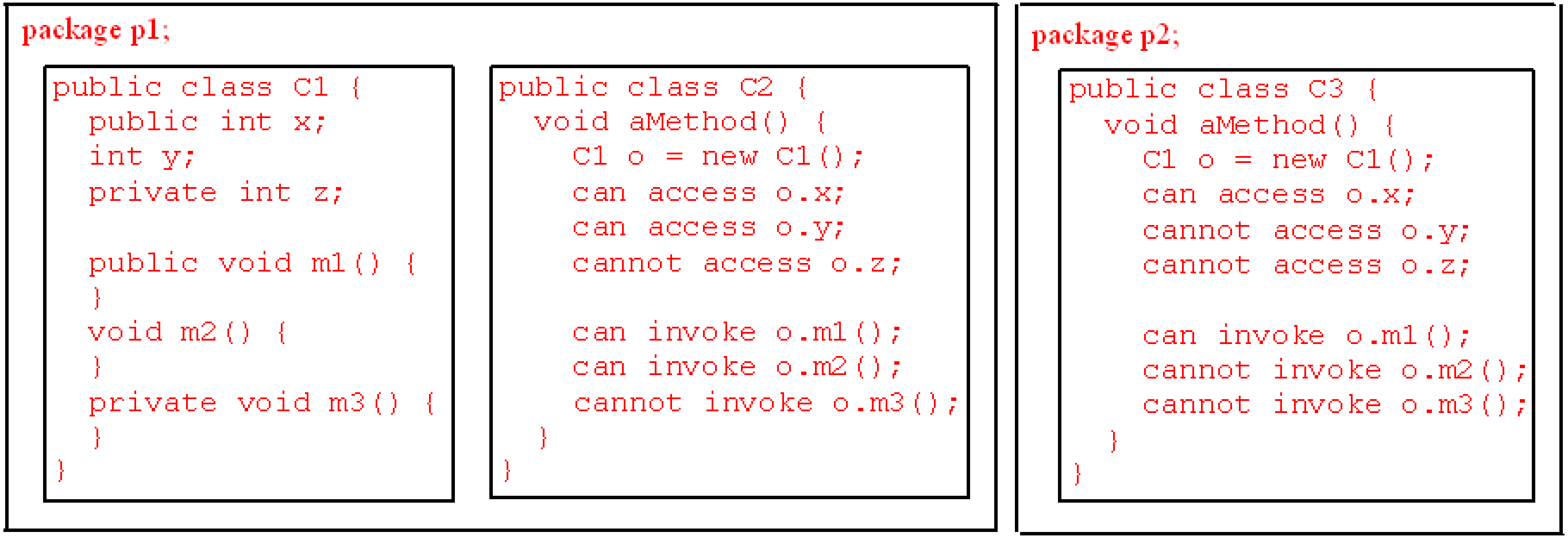
Visibility Modifiers

- If no package is specified, class is in default package:

  `package packageName;`

- By default, the class, variable, or method can be accessed by any class in the same package.
  - package-private or package-access
- `public`
  - Class, data, or method is visible to any class
  - In most cases, constructor should be public; however, a *private constructor will prohibit user from creating instances of a class*.
- `private`
  - Data or methods can only be accessed by the declaring class
  - get and set methods are used to read and modify private properties

Visibility Modifiers Example



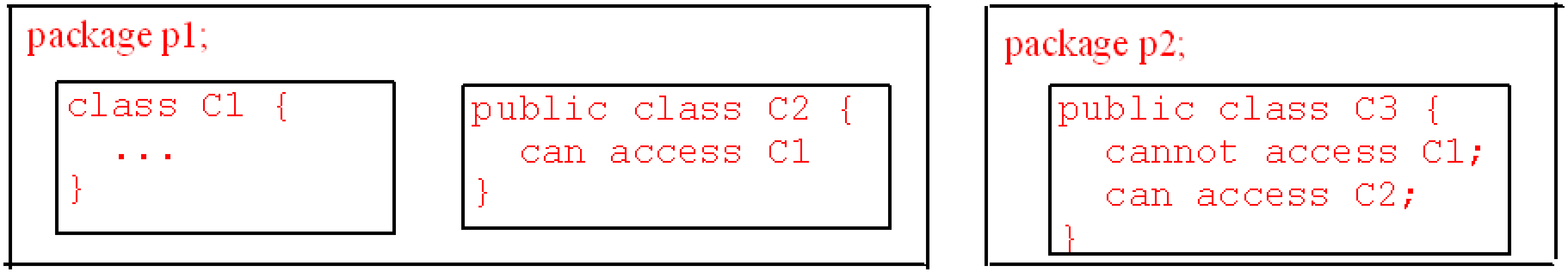
package p1;

```
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 {
  ...
}
```

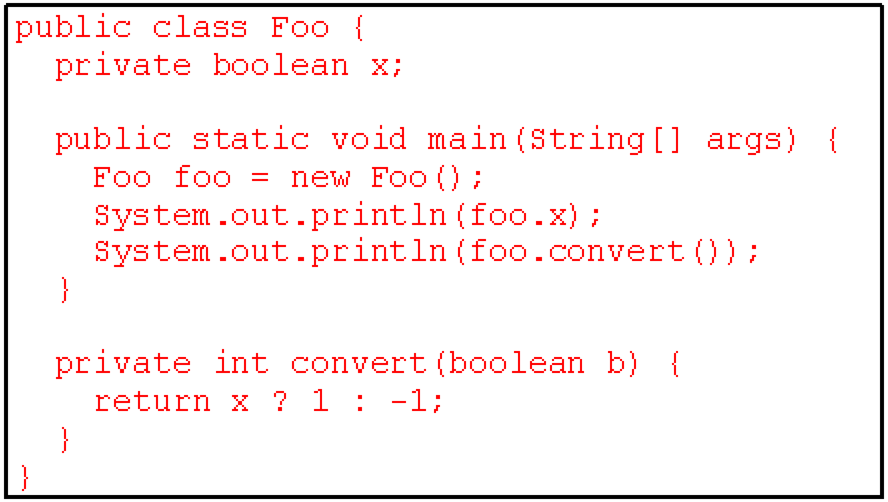
```
public class C2 {
  can access C1
}
```

package p2;

```
public class C3 {
  cannot access C1;
  can access C2;
}
```

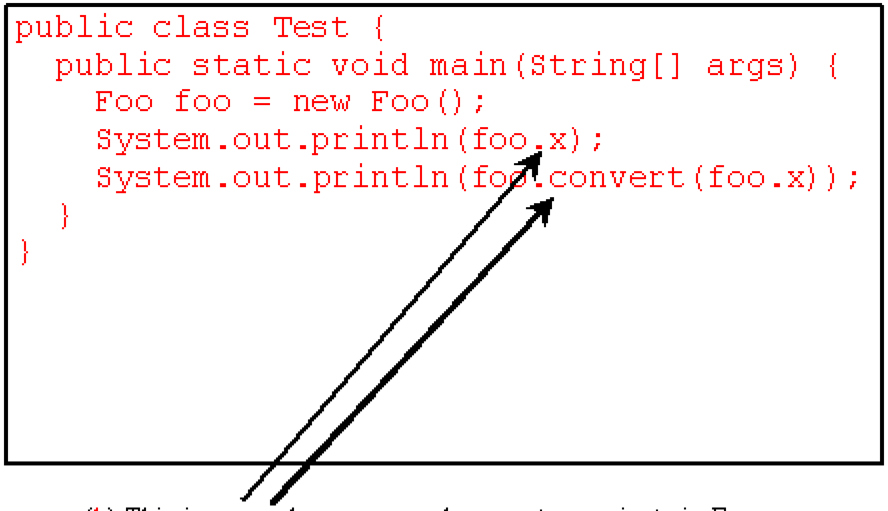
Private Members of a Class

- Object used in client of class code cannot directly access private members
- Object in own class code can directly access private members



```
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

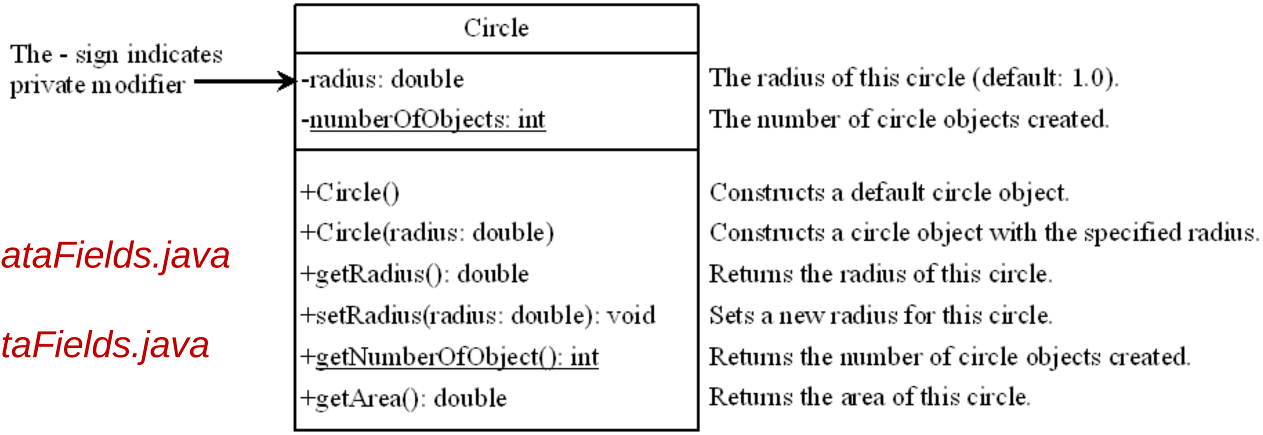


```
public class Test {
  public static void main(String[] args) {
    Foo foo = new Foo();
    System.out.println(foo.x);
    System.out.println(foo.convert(foo.x));
  }
}
```

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

Data Field Encapsulation

- Prevents direct modification of data fields by declaring them private
  - get method (aka getter or accessor) is used to **retrieve** private data from outside the class definition
    - if return type is boolean, accessor method is defined using '*isFieldName*'
  - set method (aka setter or mutator) is used to **modify** private data from outside the class definition



The - sign indicates private modifier

Circle

-radius: double — The radius of this circle (default: 1.0).

-numberOfObjects: int — The number of circle objects created.

+Circle() — Constructs a default circle object.

+Circle(radius: double) — Constructs a circle object with the specified radius.

+getRadius(): double — Returns the radius of this circle.

+setRadius(radius: double): void — Sets a new radius for this circle.

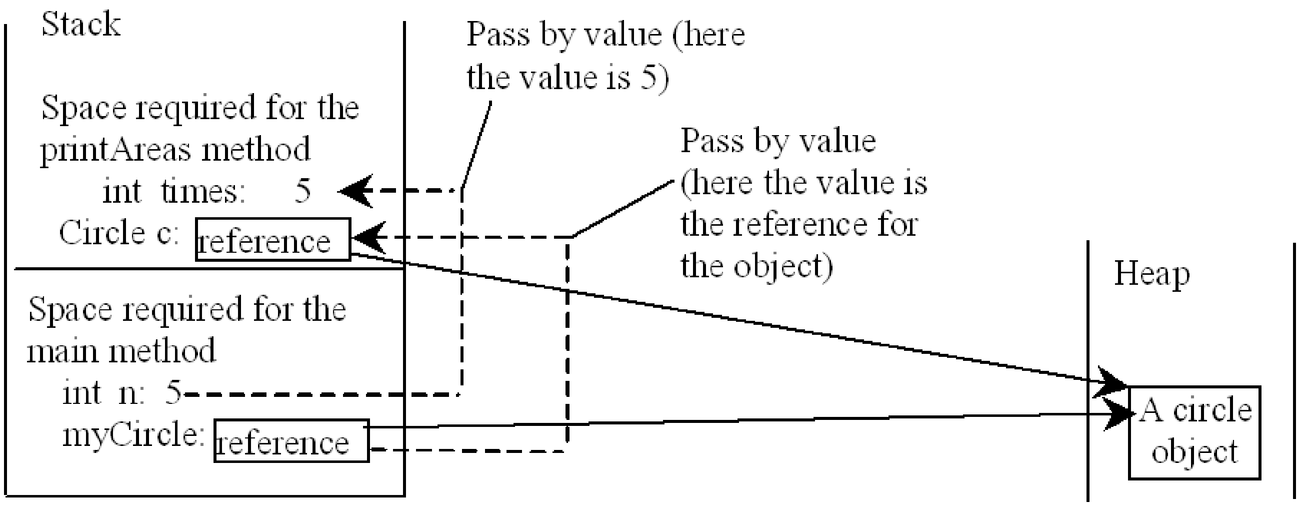
+getNumberOfObject(): int — Returns the number of circle objects created.

+getArea(): double — Returns the area of this circle.

See 8.9 CircleWithPrivateDataFields.java
See 8.10 TestCircleWithPrivateDataFields.java

Passing Objects to Methods

- Similar to passing an array to a method, passing an object is actually passing the reference of the object


Stack
Space required for the printAreas method
int times: 5
Circle c: reference
Space required for the main method
int n: 5
myCircle: reference
Pass by value (here the value is 5)
Pass by value (here the value is the reference for the object)
Heap
A circle object

See 8.11 TestPassObject.java

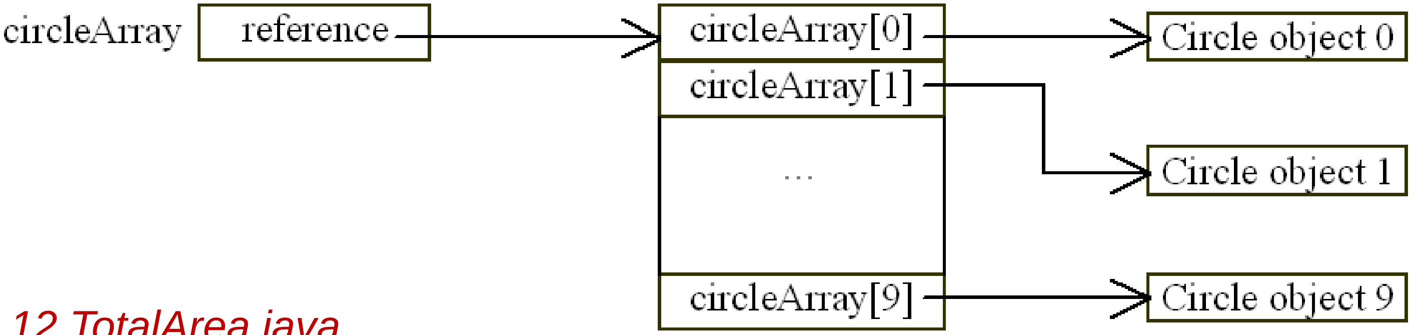
Array of Objects

- An array of objects is actually an array of reference variables with a default value of null

```
className[] arrayRefVar = new className[arraySize];
```

- Each element in the array should then create a class object

```
for (int i = 0; i < arrayRefVar.length; i++) {
   arrayRefVar[i] = new className();
}
```



See 8.12 TotalArea.java

Java Library Classes

- Date
  - System-independent encapsulation of date and time
  - Can use with format string
    - See Date/Time Conversions
- Random
  - Generates a stream of pseudorandom numbers from a given seed value
- JFrame
  - Creates top-level window with title and border
- JButton
  - Creates a common push button graphical widget

Java Library Classes

- JRadioButton
  - Creates one of a group of radio buttons
- JComboBox
  - Creates button with drop-down list of values or text field and drop-down list where user can select a value
- JList
  - Creates a group of items, displayed in one or more columns, where user can make one or more selections

See 8.5 TestFrame.java
See 8.6 GUIComponents.java