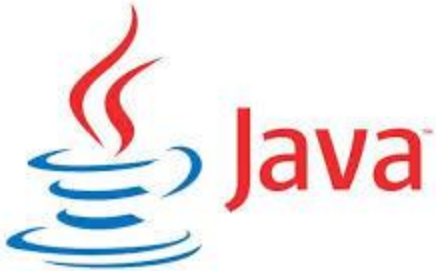
CS212: Object-Oriented Programming

Introduction to Classes and Objects

Instructor: Hirra Anwar



Objectives

In this chapter you will learn:

- What are classes, objects, methods and instance variables.
- How to declare a class and use it to create an object.
- How to declare methods in a class to implement the class's behaviors.
- How to declare instance variables in a class to implement the class's attributes.
- How to call an object's method to make that method perform its task.
- The differences between instance variables of a class and local variables of a method.
- How to use a constructor to ensure that an object's data is initialized when the object is created.
- The differences between primitive and reference types.

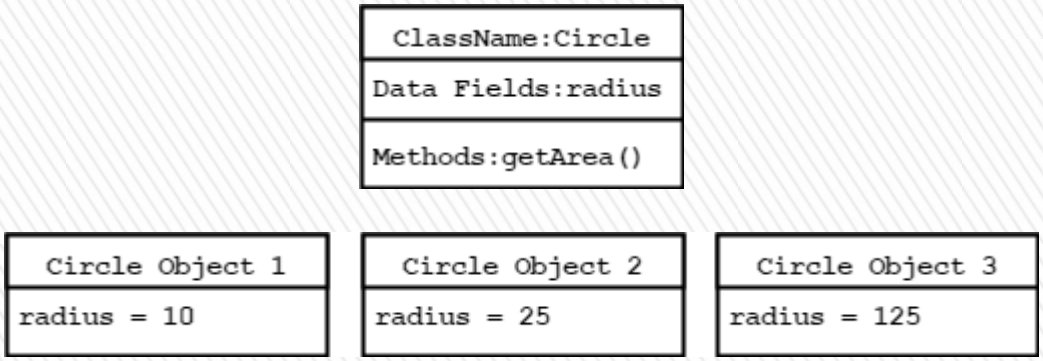
OO Programming Concepts

- 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.
  - state defines the object, and
  - behavior defines what the object does.

A CLASS TEMPLATE



Three objects of the Circle class

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

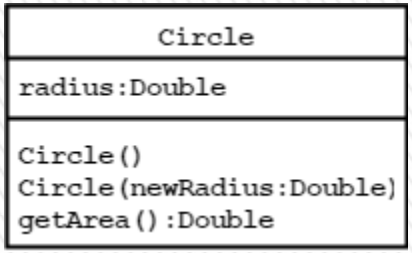
```
1.  class Circle {

2.      /** The radius of this circle */
3.      double radius = 1.0;

4.      /** Construct a circle object */
5.      Circle() {
6.      }

7.      /** Construct a circle object */
8.      Circle(double newRadius) {
9.              radius = newRadius;
10.     }

11.     /** Return the area of this circle */
12.     double getArea() {
13.             return radius * radius * 3.14159;
14.     }
15. }
```

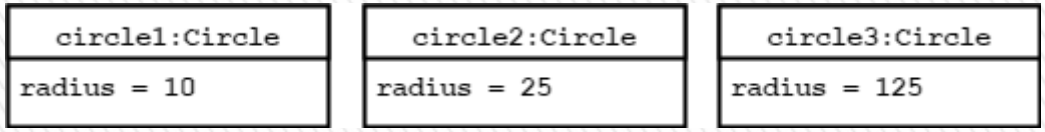
Data field

Constructors

Method

UML Class Diagram





UML notation for objects

CONSTRUCTORS

- Constructors are a special kind of methods that are invoked to construct objects.
- Constructors must have the same name as the class itself.
- Constructors do not have a return type—not even void.
- Java requires a constructor for every class.

```
1.  Circle() {
2.  }

3.  Circle(double newRadius) {
4.     radius = newRadius;
5.  }
```

Creating Objects Using Constructors

- Constructors are invoked using the new operator when an object is created.
- Constructors play the role of initializing objects.

```
new ClassName();
```

Example

```
new Circle();
new Circle(5.0);
```

Default Constructor

- A constructor with no parameters is referred to as a no-arg constructor.
- Java will provide a default no-argument constructor if none is provided.

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

» Syntax:

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

» Example:

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

» Create an object
» Assign object reference

NOTE

- Normally, objects are created with new operator.
- One exception is a string literal that is contained in quotes, such as "hello".
- String literals are references to String objects that are implicitly created by Java.

Accessing Object's Members

» Referencing the object's data:

```
objectRefVar.data
```

» Example

```
myCircle.radius
```

» Invoking the object's method:

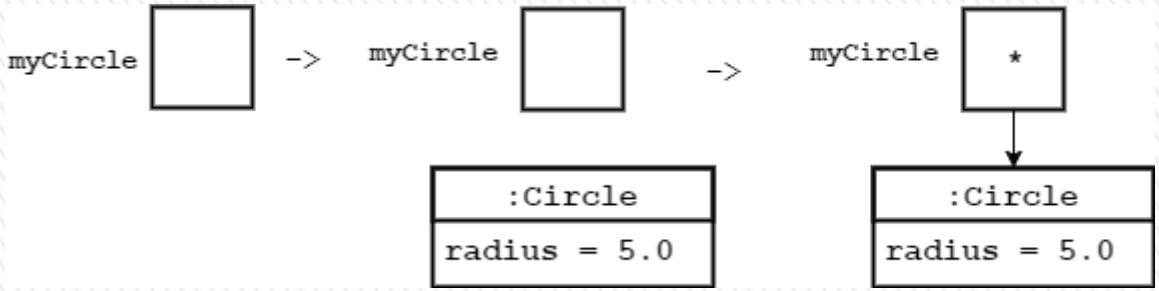
```
objectRefVar.methodName(arguments)
```

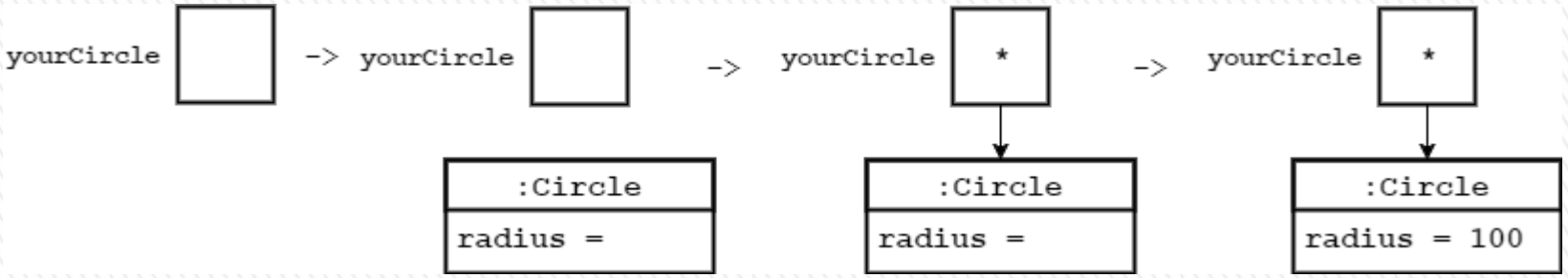
» Example

```
myCircle.getArea()
```

TRACE CODE

```
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
```





NOTE

- Recall that you use

```
Math.methodName(arguments)
```

- Example

```
Math.pow(3, 2.5)
```

- to invoke a method in the Math class.

- Q. Can you invoke getArea() using Circle.getArea()?
- A. 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)
```

- Example

```
myCircle.getArea()
```

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.

```
1. public class Student {
2.     String name; // name has default value null
3.     int age;     // age has default value 0
4.     boolean isScienceMajor; // isScienceMajor has
   default value false
5.     char gender; // c has default value '\u0000'
6. }
```

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
  - '\u0000' for a char type
- However, Java assigns no default value to a local variable inside a method.

```
1.  public class Test {
2.     public static void main(String[] args) {
3.        Student student = new Student();
4.        System.out.println("name? " + student.name);
5.        System.out.println("age? "  + student.age );
6.        System.out.println("isScienceMajor? " +
    student.isScienceMajor);
7.        System.out.println("gender? " + student.gender);
8.     }
9.  }
```

EXAMPLE

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

```
1. public class Test {
2.     public static void main(String[] args) {
3.         int x;    // x has no default value
4.         String y; // y has no default value
5.         System.out.println("x is " + x);
6.         System.out.println("y is " + y);
7.     }
8. }
```

» Compilation error: variables not initialized

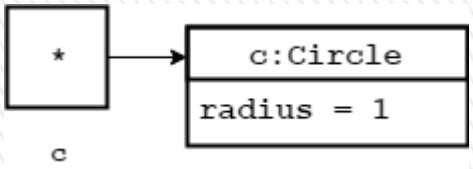
Differences between Variables of Primitive Data Types and Object Types

» Primitive type


```
int i = 1;
```

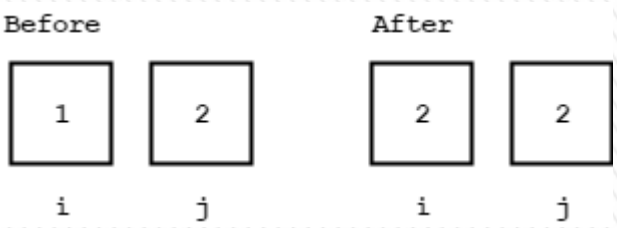
» Object type `Circle c = new Circle();`

» c is a reference created using new Circle()

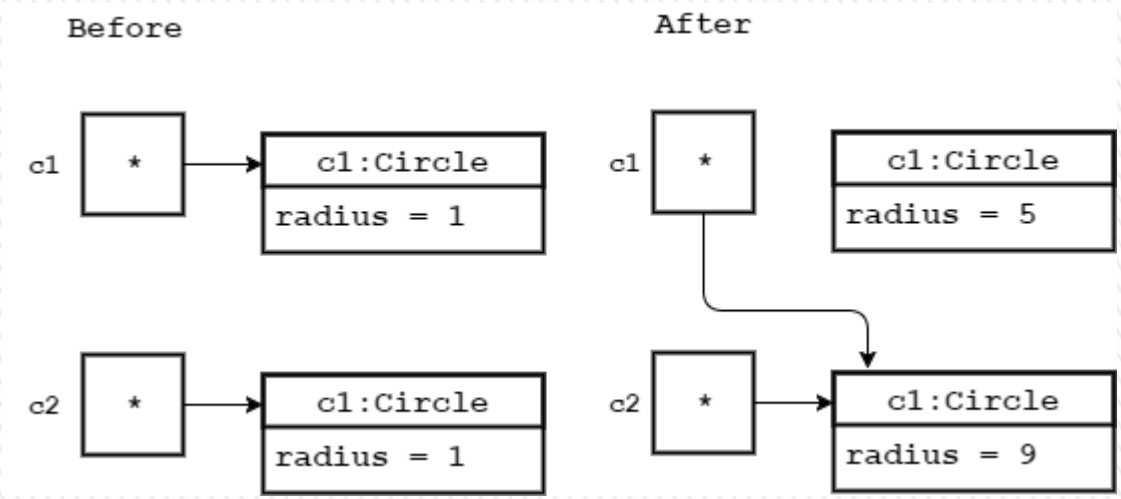


Copying Variables of Primitive Data Types and Object Types

» Primitive type assignment i = j



» Object type assignment c1 = c2



Garbage Collection

- After the assignment statement

```
c1 = c2
```

- c1 points to the same object referenced by c2.
- The object previously referenced by c1 is no longer referenced.
- This object is known as garbage.
- Garbage is automatically collected by JVM.

Garbage Collection, cont.

- If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object.
- The JVM will automatically collect the space if the object is not referenced by any variable.

NOTE

- Each class declaration that begins with keyword public must be stored in a file that has the same name as the class and ends with the .java filename extension.
- Declaring more than one public class in the same file is a compilation error.

Discussion

- Instantiation
- Packages in Java
- Instance variables/methods
- Static variables/methods
- This ->reference variable
- Getter and Setter methods
- Date Class, Random Class

PACKAGES

Built-in Packages

These packages consist of a large number of classes which are a part of Java **API**.Some of the commonly used built-in packages are:

1) **java.lang:** Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.

2) **java.io:** Contains classed for supporting input / output operations.

3) **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.

4) **java.applet:** Contains classes for creating Applets.

5) **java.awt:** Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).

6) **java.net:** Contain classes for supporting networking operations.

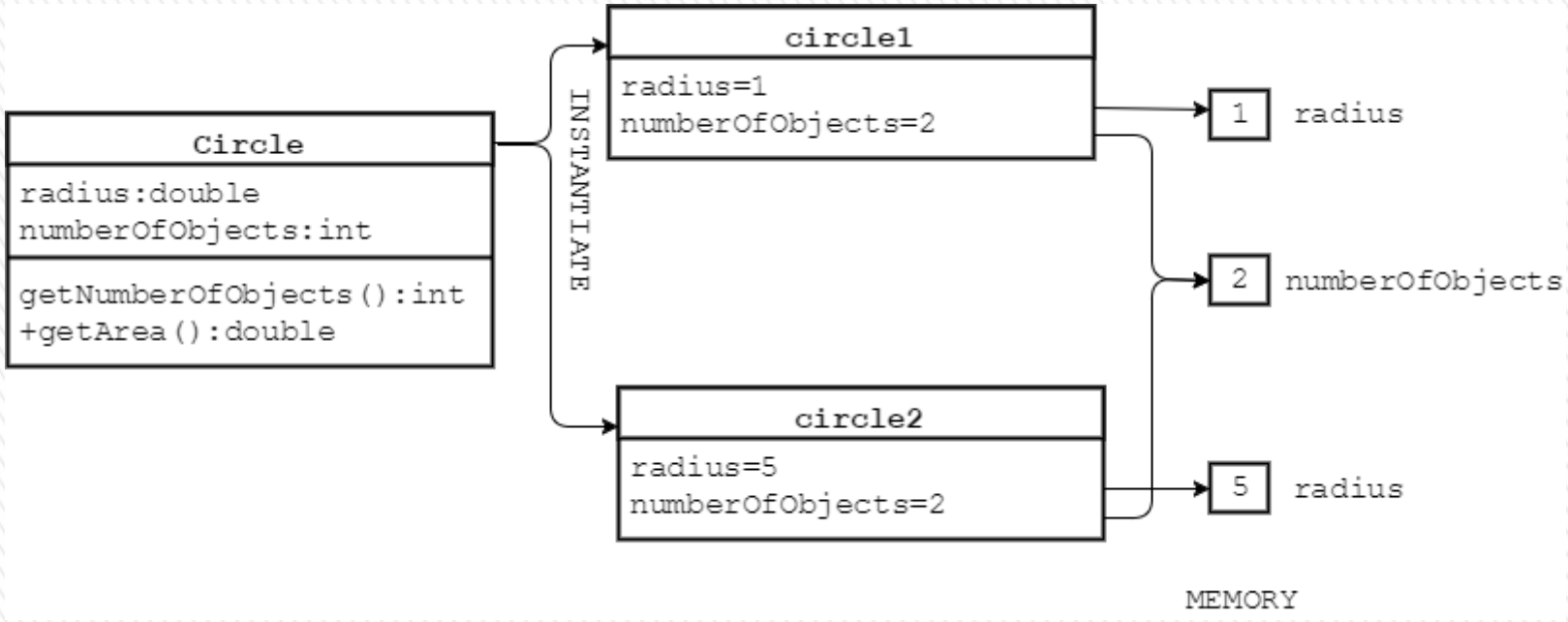
Instance Variables and Methods

- 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.
- To declare static variables, constants, and methods, use the static modifier.

STATIC VARIABLES, CONSTANTS, AND METHODS, CONT.

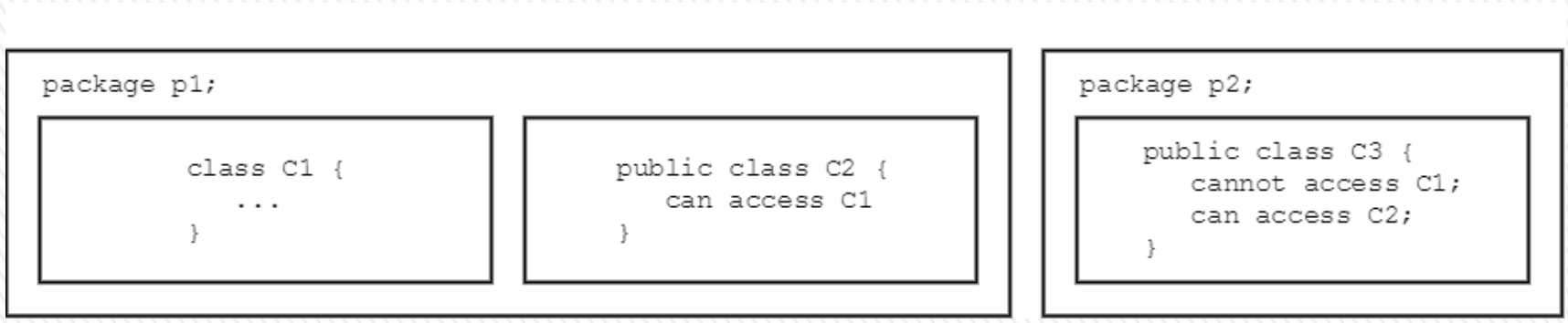


After two Circle objects were created, numberOfObjects is 2.

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.
- **Private**
  - Package access

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



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).

```
1.    public class F {
2.       private boolean x;
3.       private int convert(boolean b) {
4.          return x ? 1 : -1;
5.       }
6.       public static void main(String[] args) {
7.          F f = new F ();
8.          System.out.println(f.x);
9.          System.out.println(f.convert());
10.      }
11.   }
```

- (a) This is OK because object f is used inside the F class

```
1.    public class Test {
2.       public static void main(String[] args) {
3.          F f = new F();
4.          System.out.println(f.x);
5.          System.out.println(f.convert(f.x));
6.       }
7.    }
```

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

Why Data Fields Should Be private?

- To protect data.
- To make class easy to maintain.

GETTERS AND SETTERS

```
1  class Person
2  {
3      private int id;
4      private String name;
5      private int age;
6      private String address;
7
8      //Constructor
9      public Person(int id)
10     {
11         this.id = id;
12         this.name = "Unknowm";
13         this.age = -1;
14         this.address = "Unknown";
15     }
16
```

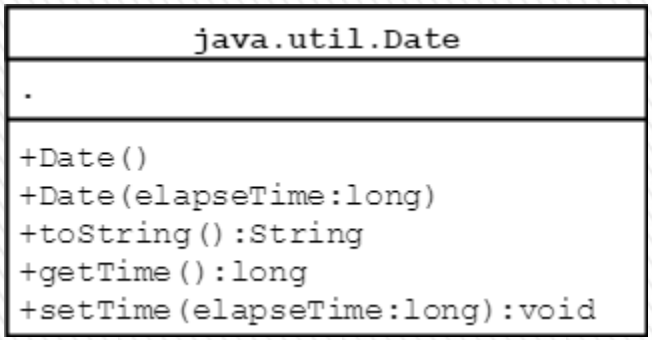
GETTERS AND SETTERS

```
17      //Overloaded Constructor
18      public Person(int id,String name,int age,String address)
19      {
20          this.id = id;
21          this.name = name
22          this.age = age;
23          this.address = address;
24      }
25
26      //Getter method for name
27      public String getName()
28      {
29          return name;
30      }
31
32      //Setter method for name
33      public String setName()
34      {
35          this.name = name;
36      }
```

The Date Class

- Java provides a system-independent encapsulation of date and time in the java.util.Date class.
- Used to create an instance for the current date and time.
- Use toString method to return the date and time as a string.

The Date Class, Cont.



- Constructs a Date object for the current time.
- Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.
- Returns a string representing the date and time.
- Returns the number of milliseconds since January 1, 1970, GMT.
- Sets a new elapse time in the object.

The Date Class Example

- Example

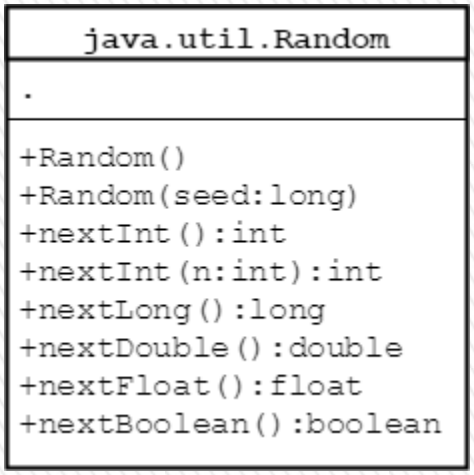
```
java.util.Date date = new java.util.Date();
System.out.println(date.toString());
```

- displays a string like Mon Feb 27 11:15:30 PKT 2017

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.
- We have seen both of them earlier!

The Random Class



1. Constructs a Random object with the current time as its seed.
2. Constructs a Random object with a specified seed.
3. Returns a random int value.
4. Returns a random int value between 0 (inclusive) and n (exclusive).
5. Returns a random long value.
6. Returns a random double value between 0.0 and 1.0 (exclusive).
7. Returns a random float value between 0.0F and 1.0F (exclusive).
8. Returns a random boolean value.

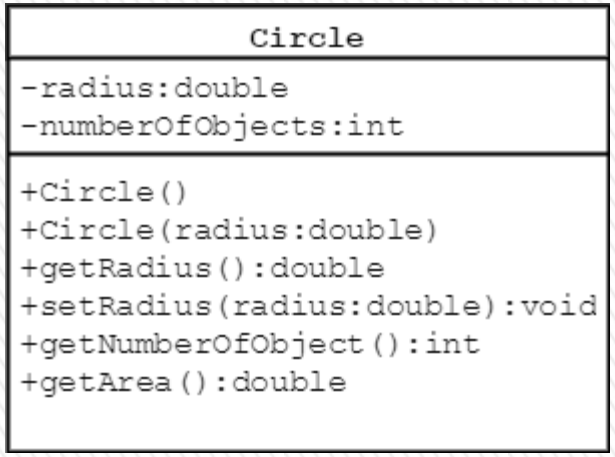
The Random Class Example

- 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.

```
1.  Random random1 = new Random(3);
2.  System.out.print("From random1: ");
3.  for (int i = 0; i < 10; i++)
4.     System.out.print(random1.nextInt(1000) + " ");

5.  Random random2 = new Random(3);
6.  System.out.print("\nFrom random2: ");
7.  for (int i = 0; i < 10; i++)
8.     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

Example of Data Field Encapsulation



1. The radius of this circle (default: 1.0).
2. The number of circle objects created.
3. Constructs a default circle object.
4. Constructs a circle object with the specified radius.
5. Returns the radius of this circle.
6. Sets a new radius for this circle.
7. Returns the number of circle objects created.
8. Returns the area of this circle.

Questions/Answers & Discussion