

Topics to covered:

Constructor Overloading

Object and Instances


Static Variables, Constants, and Methods

Access to Data Encapsulation

Constructor Overloading in Java

- Overloaded constructor is called based upon the parameters specified when **new** is executed.
- Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading.

```
public class Demo {  
    Demo() {  
        ..  
    }  
    Demo(String s) {  
        ...  
    }  
    Demo(int i) {  
        ...  
    }  
    ....  
}
```



Three overloaded constructors -
They must have
different
Parameters list

Example:

Three objects of class Box are created. One is with default constructor (no dimensions specified), one with only single dimension given for cube, and another one providing 3 dimensions. All the constructors have different initialization code, similarly you can create any number of constructors with different initialization codes for different purposes

```
// Java program to illustrate Constructor Overloading
class Box
{
    double width, height, depth;
    // constructor used when all dimensions specified
    Box(double w, double h, double d)
    {
        width = w;
        height = h;
        depth = d;
    }
    // constructor used when no dimensions specified
    Box()
    {
        width = height = depth = 0;
    }
    // constructor used when cube is created
    Box(double len)
    {
        width = height = depth = len;
    }
    // compute and return volume
    double volume()
    {
        return width * height * depth;
    }
}
```

```
// Driver code
public class Test
{
    public static void main(String args[])
    {
        // create boxes using the various
        // constructors
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

        double vol;
        // get volume of first box
        vol = mybox1.volume();
        System.out.println(" Volume of mybox1 is " + vol);
        // get volume of second box
        vol = mybox2.volume();
        System.out.println(" Volume of mybox2 is " + vol);
        // get volume of cube
        vol = mycube.volume();
        System.out.println(" Volume of mycube is " + vol);
    }
}
```

Output:

```
Volume of mybox1 is 3000.0
Volume of mybox2 is 0.0
Volume of mycube is 343.0
```

Object and Instance

- Each object said to be an instance of its class but each instance of the class has its own value for each attributes instances shares the attribute name and operation with their instances of class but an object contains an implicit reference to his on class.
- Imagine a product like a computer.
 - THE xw6400 workstation is an object
 - YOUR xw6400 workstation, (or YOUR Friend's xw6400 workstation) is an instance of the xw6400 workstation object
- Class House --> Blueprint of the house. But you can't live in the blue print. You need a physical House which is the instance of the class to live in. i.e., actual address of the object is instance. Instances represent objects.

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 and final

- **static** means there is only one copy of the variable in memory shared by all instances of the class. The **final** keyword just means the value can't be changed. Without **final**, any object can change the value of the variable.
- Static and final both are the keywords used in Java. The **static member can be accessed before the class object is created**. Final has a different effect when applied to class, methods and variables.
- The main difference between a static and final keyword is that **static** is keyword is used to define the class member that can be used independently of any object of that class. **Final** keyword is used to declare, a constant variable, a method which can not be overridden and a class that can not be inherited.

Characteristics of Static methods

- A static method can be called using the class (className.methodName) as opposed to an instance reference:
 - `instanceOfClass xyz = new instanceOfClass();`
`xyz.methodName;`
- **Static methods cannot use non-static instance variables:** a static method can't refer to any instance variables of the class. The static method doesn't know which instance's variable value to use.
- **Static methods cannot call non-static methods:** non-static methods usually use instance variable state to affect their behaviour. Static methods can't see instance variable state, so if you try to call a non-static method from a static method the compiler will complain regardless if the non-static method uses an instance variable or not.

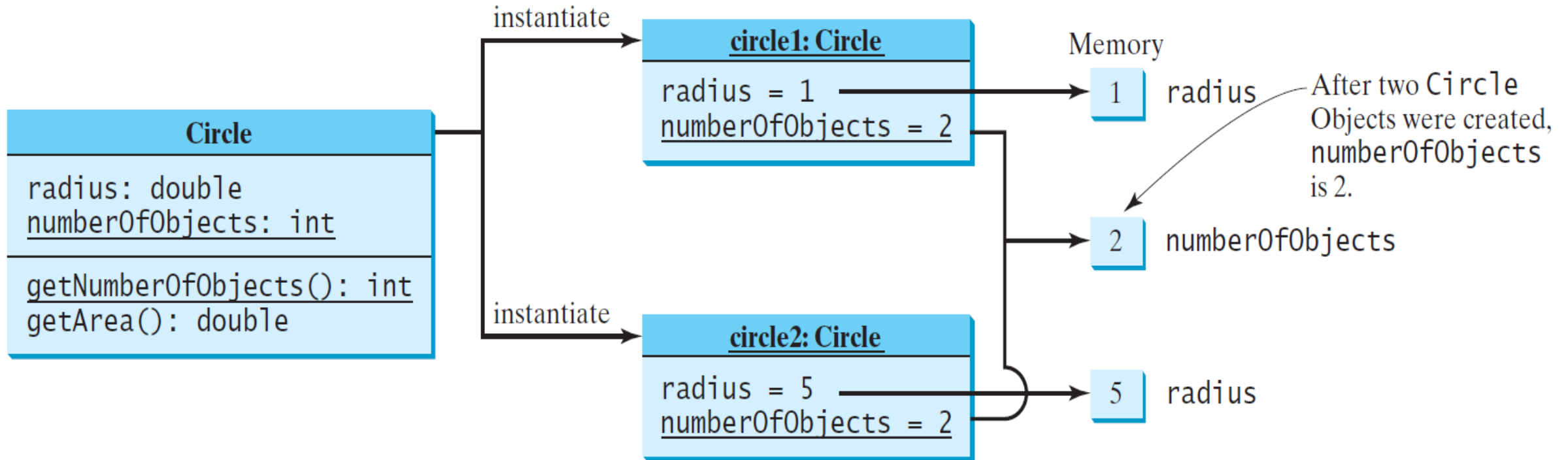
Characteristics of Non-Static methods

- A non-static method **does not have the keyword *static*** before the name of the method.
- A non-static method **belongs to an object of the class** and you have to **create an instance of the class to access it**.
- Non-static methods can access any static method and any static variable without creating an instance of the class.

Static Variables, Constants, and Methods (cont)

UML Notation:

underline: static variables or methods



Example of Using Instance and Static Variables and Method

Objective: Demonstrate the roles of instance and class variables and their uses. This example adds a class variable `numberOfObjects` to track the number of `Circle` objects created.

CircleWithStaticMembers

TestCircleWithStaticMembers

Run

```

public class CircleWithStaticMembers {
    /** The radius of the circle */
    double radius;

    /** The number of the objects created */
    static int numberOfObjects = 0;

    /** Construct a circle with radius 1 */
    CircleWithStaticMembers() {
        radius = 1.0;
        numberOfObjects++;
    }

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

    /** Return numberOfObjects */
    static int getNumberOfObjects() {
        return numberOfObjects;
    }

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

```

```

public class TestCircleWithStaticMembers {
    /** Main method */
    public static void main(String[] args) {
        System.out.println("Before creating objects");
        System.out.println("The number of Circle objects is " +
            CircleWithStaticMembers.numberOfObjects);

        // Create c1
        CircleWithStaticMembers c1 = new CircleWithStaticMembers();

        // Display c1 BEFORE c2 is created
        System.out.println("\nAfter creating c1");
        System.out.println("c1: radius (" + c1.radius +
            ") and number of Circle objects (" +
            c1.numberOfObjects + ")");

        // Create c2
        CircleWithStaticMembers c2 = new CircleWithStaticMembers(5);

        // Modify c1
        c1.radius = 9;

        // Display c1 and c2 AFTER c2 was created
        System.out.println("\nAfter creating c2 and modifying c1");
        System.out.println("c1: radius (" + c1.radius +
            ") and number of Circle objects (" +
            c1.numberOfObjects + ")");
        System.out.println("c2: radius (" + c2.radius +
            ") and number of Circle objects (" +
            c2.numberOfObjects + ")");
    }
}

```

Output

```
Before creating objects  
The number of Circle objects is 0
```

```
After creating c1  
c1: radius (1.0) and number of Circle objects (1)
```

```
After creating c2 and modifying c1  
c1: radius (9.0) and number of Circle objects (2)  
c2: radius (5.0) and number of Circle objects (2)
```

Why Data Fields Should Be private?

To protect data.

To make code easy to maintain.

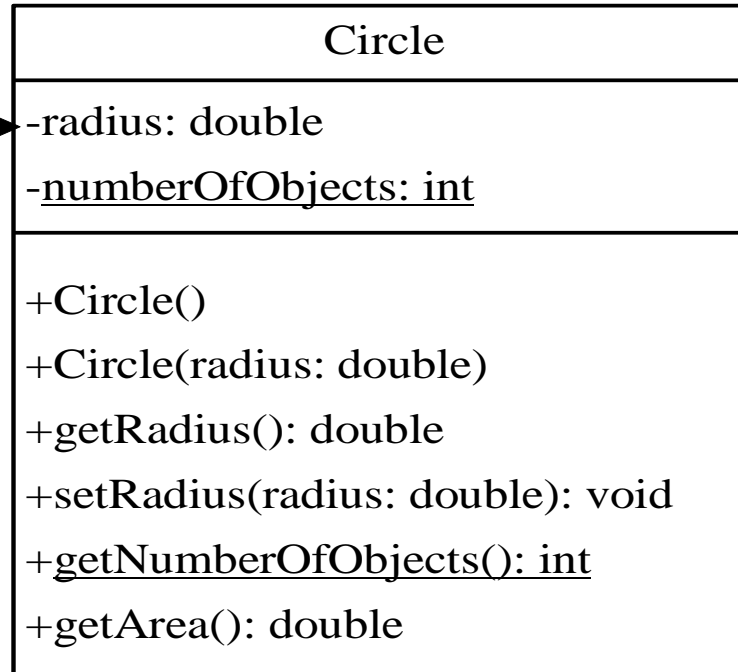
Advantage of Setter and Getter

- **Getter** and **Setter** Methods in **Java**.

If a data member is declared "private", then it can only be accessed within the same class. No outside class can access data member of that class. If you need to access these variables, you have to use public "**getter**" and "**setter**" methods

Example of Data Field Encapsulation

The - sign indicates
private modifier



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.

CircleWithPrivateDataFields

TestCircleWithPrivateDataFields

Run

```
public class CircleWithPrivateDataFields {
    /** The radius of the circle */
    private double radius = 1;

    /** The number of the objects created */
    private static int numberOfObjects = 0;

    /** Construct a circle with radius 1 */
    public CircleWithPrivateDataFields() {
        numberOfObjects++;
    }

    /** Construct a circle with a specified radius */
    public CircleWithPrivateDataFields(double newRadius) {
        radius = newRadius;
        numberOfObjects++;
    }

    /** Return radius */
    public double getRadius() {
        return radius;
    }

    /** Set a new radius */
    public void setRadius(double newRadius) {
        radius = (newRadius >= 0) ? newRadius : 0;
    }
}
```

```
/** Return numberOfObjects */
public static int getNumberOfObjects() {
    return numberOfObjects;
}

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

```
public class TestCircleWithPrivateDataFields {  
    /** Main method */  
    public static void main(String[] args) {  
        // Create a Circle with radius 5.0  
        CircleWithPrivateDataFields myCircle =  
            new CircleWithPrivateDataFields(5.0);  
        System.out.println("The area of the circle of radius "  
            + myCircle.getRadius() + " is " + myCircle.getArea());  
  
        // Increase myCircle's radius by 10%  
        myCircle.setRadius(myCircle.getRadius() * 1.1);  
        System.out.println("The area of the circle of radius "  
            + myCircle.getRadius() + " is " + myCircle.getArea());  
  
        System.out.println("The number of objects created is "  
            + CircleWithPrivateDataFields.getNumberOfObjects());  
    }  
}
```

Output

```
The area of the circle of radius 5.0 is 78.53981633974483  
The area of the circle of radius 5.5 is 95.03317777109125  
The number of objects created is 1
```

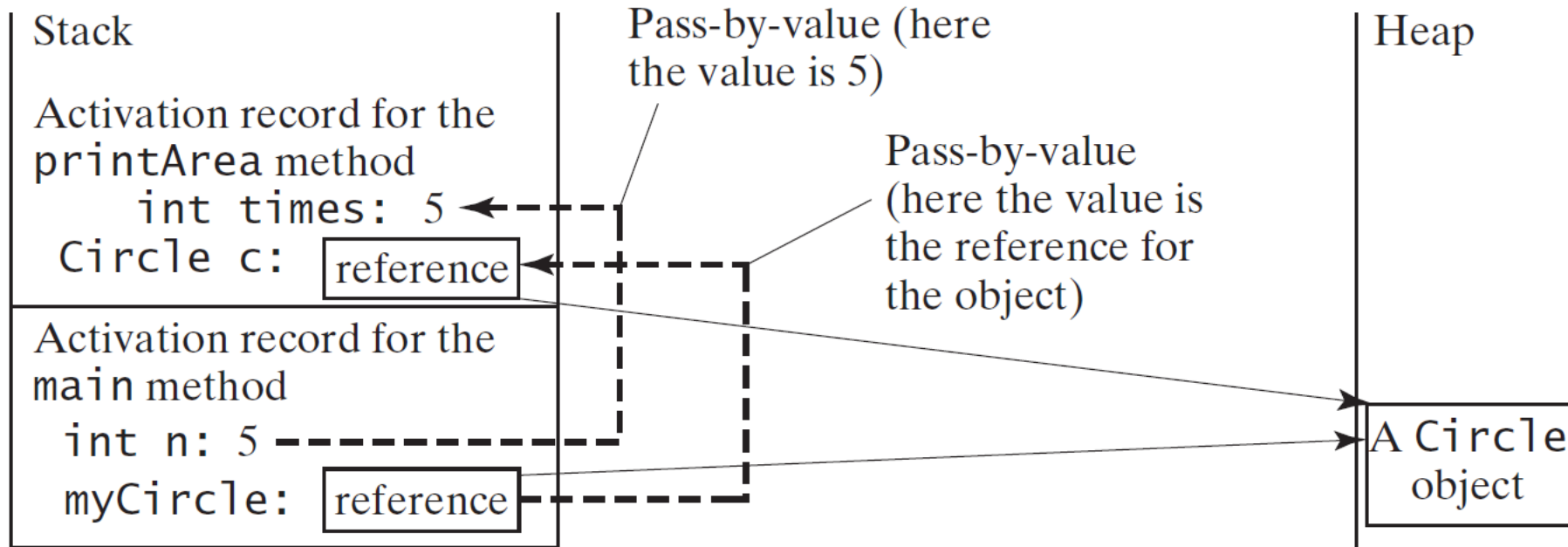
Passing Objects to Methods

- ❑ Passing by value for primitive type value (the value is passed to the parameter)
- ❑ Passing by value for reference type value (the value is the reference to the object)

TestPassObject

Run

Passing Objects to Methods, cont.



```
public class TestPassObject {
    /** Main method */
    public static void main(String[] args) {
        // Create a Circle object with radius 1
        CircleWithPrivateDataFields myCircle =
            new CircleWithPrivateDataFields(1);

        // Print areas for radius 1, 2, 3, 4, and 5.
        int n = 5;
        printAreas(myCircle, n);

        // See myCircle.radius and times
        System.out.println("\n" + "Radius is " + myCircle.getRadius());
        System.out.println("n is " + n);
    }

    /** Print a table of areas for radius */
    public static void printAreas(
        CircleWithPrivateDataFields c, int times) {
        System.out.println("Radius \t\tArea");
        while (times >= 1) {
            System.out.println(c.getRadius() + "\t\t" + c.getArea());
            c.setRadius(c.getRadius() + 1);
            times--;
        }
    }
}
```

Output

Radius	Area
1.0	3.141592653589793
2.0	12.566370614359172
3.0	28.274333882308138
4.0	50.26548245743669
5.0	78.53981633974483

Radius is 6.0
n is 5

Array of Objects

```
Circle[] circleArray = new Circle[10];
```

An array of objects is actually an *array of reference variables*. So invoking `circleArray[1].getArea()` involves two levels of referencing as shown in the next figure.

`circleArray` references to the **entire array**.
`circleArray[1]` references to a **Circle object**.

Array of Objects, cont.

```
Circle[] circleArray = new Circle[10];
```

