

Lecture #8

CSC 200-04L Fall 2018

Ambrose Lewis
tjl274@email.vccs.edu

Agenda

- Introduction to Artificial Intelligence
 - Definition & Example Problems
 - Machine Learning
 - Neural Networks
 - More resources...
- Programming Languages
- Object Orientated Programming Concepts

Introduction to Artificial Intelligence

Definition and Example Problems

- AI is software that uses advanced algorithms and mathematical tools to tackle complex real-world problems
- Example problems:
 - web search
 - speech recognition
 - handwriting recognition
 - face recognition
 - machine translation
 - autonomous driving
- Characteristics of AI Problems:
 - High societal impact (affect billions of people)
 - Diverse (language, vision, robotics)
 - Complex (really hard)



IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

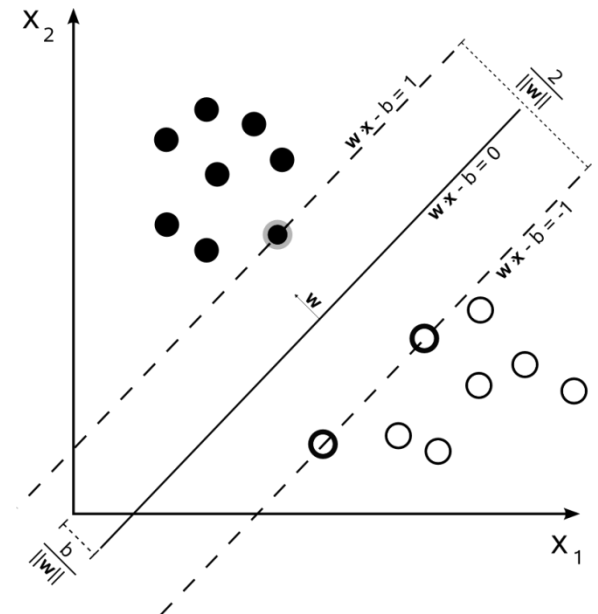
<http://xkcd.com/1425/>

Introduction to Artificial Intelligence

Machine Learning

- Field of study that gives computers the ability to learn without being explicitly programmed
- Types of problems suitable for Machine Learning approaches:
 - Classification (mapping a set of inputs into distinct pre-defined groups)
 - Clustering (mapping a set of inputs into groups that are not known beforehand)
- Two categories of Machine Learning:
 - Supervised Learning: Program is given example inputs and their desired outputs, goal is to have the program learn rules from these example inputs to determine how to map inputs into outputs for new examples [CLASSIFICATION]
 - Unsupervised Learning: Program searches for patterns in the input data, discovery of hidden data patterns [CLUSTERING]

Example of a Machine Learning classifier called a Support Vector Machine (SVM)
The SVM here is dividing circles into 2 classes: white & black circles



http://en.wikipedia.org/wiki/Machine_learning#mediaview/File:Svm_max_sep_hyperplane_with_margin.png

Introduction of Artificial Intelligence

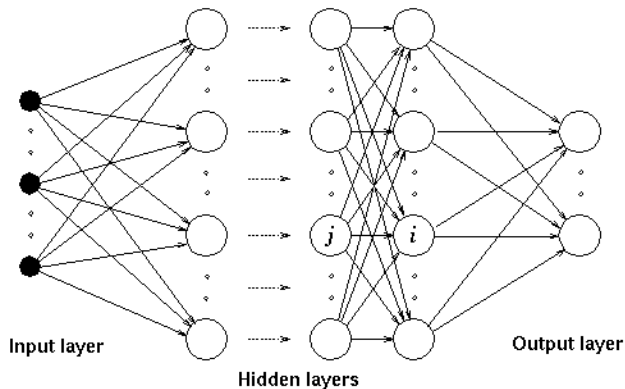
Neural Networks

- A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.
- Loosely modeled after the neural structure of the animal brains (but on much smaller scales)
- Structure:
 - Input layer (group of processing elements that receive the input data)
 - Hidden layer (group of PEs between the input & output, typically there are several hidden layers in a neural network)
 - Output layer (group of PEs that display the output data)
 - Connections (pathway between a PE in layer N and a PE in layer $N+1$, PEs have multiple connection between themselves and several PEs in the next layer)
 - Weights (each connection has a weight that defines how likely that connection will be taken for a given input, think of nerve cells firing)

Introduction to Artificial Intelligence

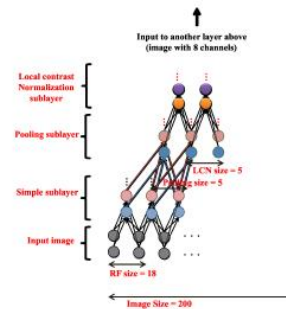
Neural Networks (Cont.)

Example of a Neural Network



<http://ecee.colorado.edu/~ecen4831/lectures/NNdemo.html>

Example Problem #1: Google identifying cats in frames of YouTube videos



<http://research.google.com/pubs/pub38115.html>

Example Problem #2: Number Recognition from Street View Imagery using Neural Networks



<http://research.google.com/pubs/pub42241.html>

Introduction to Artificial Intelligence

More Resources...

- Support Vector Machine example:
 - <http://cs.stanford.edu/people/karpathy/svmjs/demo/>
- Neural Network example:
 - <http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>
- On-line Courses:
 - Stanford: <http://web.stanford.edu/class/cs221/>
 - MIT: <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/>
 - Coursera
- Books from O'reilly:
 - “Machine Learning for Hackers” by Conway
 - “Thoughtful Machine Learning” by Kirk

Programming Languages

- Java is only one of many program languages in use (see http://en.wikipedia.org/wiki/List_of_programming_languages and http://en.wikibooks.org/wiki/Computer_Programming/Hello_world)
- A programming language is a formally constructed language that is meant to communicate instructions to a machine
 - Formally constructed languages are planned or constructed language, as opposed to natural evolving languages
- Programming Languages have syntax and semantics
 - Syntax: A programming language's surface form (the symbols you type)
 - Semantics: The meaning derived from the syntax

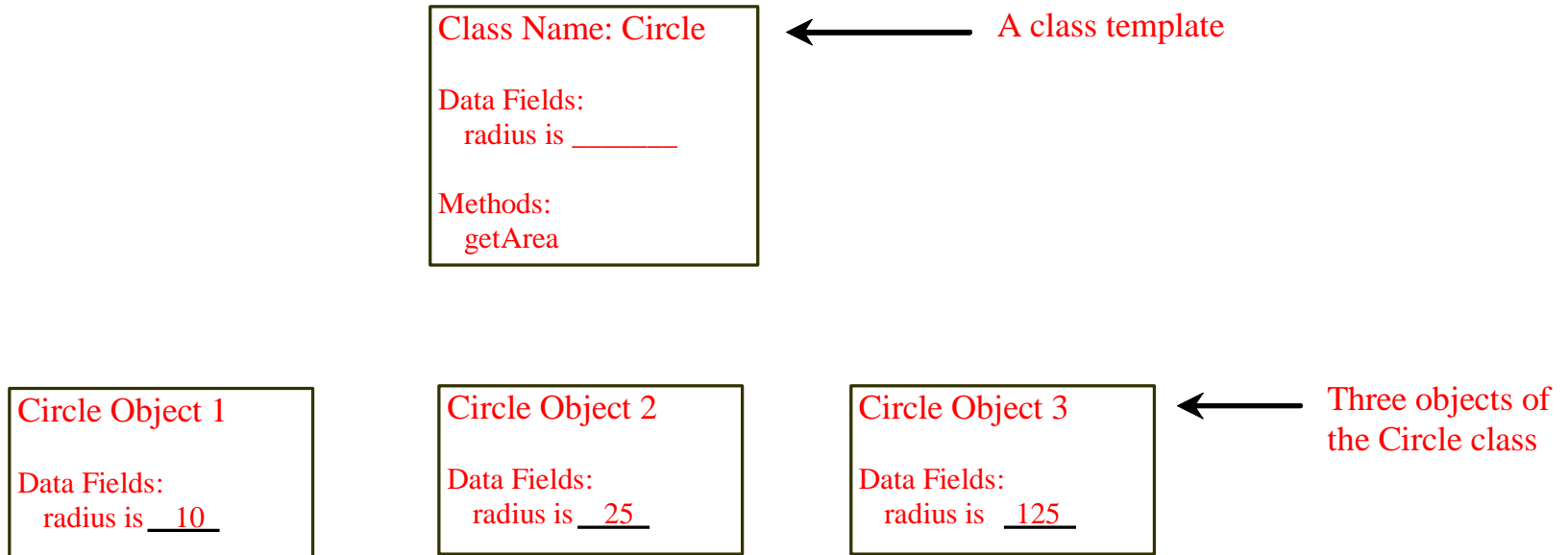
Programming Languages

- Types:
 - classifies values and expressions into data types, how it can manipulate those types and how they interact.
 - Strongly Typed: Java, variables are declared to be a single type
 - Weakly Typed: JavaScript, variables can be used as different data types within the same program (“1234” could be used as a String or an int for example)
- Compiled versus Interpreted:
 - An interpreted language executes instructions directly, without previously compiling a program into machine-language instructions. (Javascript, Basic, Python, LISP, etc.)
 - A compiled language is a programming language whose implementations are typically compilers (translators that generate machine code from source code), and not interpreters (step-by-step executors of source code, where no pre-runtime translation takes place). [Examples = C/C++, FORTRAN, etc.)

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. 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;  
    }  
}
```

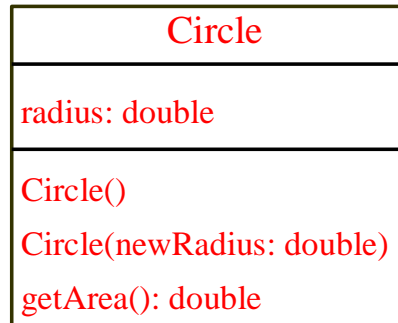
← Data field

← Constructors

← Method

UML Class Diagram

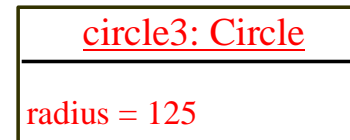
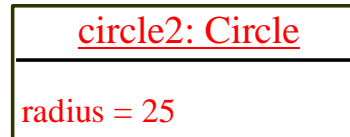
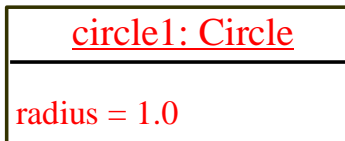
UML Class Diagram



← Class name

← Data fields

← Constructors and methods



← UML notation for objects

Example: Defining Classes and Creating Objects

Objective: Demonstrate creating objects, accessing data, and using methods. (next slide)

```

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());
    }
}

// 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

TV	
<pre>channel: int volumeLevel: int on: boolean</pre>	
<pre>+TV() +turnOn(): void +turnOff(): void +setChannel(newChannel: int): void +setVolume(newVolumeLevel: int): void +channelUp(): void +channelDown(): void +volumeUp(): void +volumeDown(): void</pre>	

The current channel (1 to 120) of this TV.
The current volume level (1 to 7) of this TV.
Indicates whether this TV is on/off.

Constructs a default TV object.
Turns on this TV.
Turns off this TV.
Sets a new channel for this TV.
Sets a new volume level for this TV.
Increases the channel number by 1.
Decreases the channel number by 1.
Increases the volume level by 1.
Decreases the volume level by 1.

The + sign indicates
a public modifier. →

```

public class TV {
    int channel = 1; // Default channel is 1
    int volumeLevel = 1; // Default volume level is 1
    boolean on = false; // By default TV is off

    public TV() {
    }

    public void turnOn() {
        on = true;
    }

    public void turnOff() {
        on = false;
    }

    public void setChannel(int newChannel) {
        if (on && newChannel >= 1 && newChannel <= 120)
            channel = newChannel;
    }

    public void setVolume(int newVolumeLevel) {
        if (on && newVolumeLevel >= 1 && newVolumeLevel <= 7)
            volumeLevel = newVolumeLevel;
    }

    public void channelUp() {
        if (on && channel < 120)
            channel++;
    }

    public void channelDown() {
        if (on && channel > 1)
            channel--;
    }

    public void volumeUp() {
        if (on && volumeLevel < 7)
            volumeLevel++;
    }

    public void volumeDown() {
        if (on && volumeLevel > 1)
            volumeLevel--;
    }
}

```

```
public class TestTV {  
    public static void main(String[] args) {  
        TV tv1 = new TV();  
        tv1.turnOn();  
        tv1.setChannel(30);  
        tv1.setVolume(3);  
  
        TV tv2 = new TV();  
        tv2.turnOn();  
        tv2.channelUp();  
        tv2.channelUp();  
        tv2.volumeUp();  
  
        System.out.println("tv1's channel is " + tv1.channel  
            + " and volume level is " + tv1.volumeLevel);  
        System.out.println("tv2's channel is " + tv2.channel  
            + " and volume level is " + tv2.volumeLevel);  
    }  
}
```

Constructors

Constructors are a special kind of methods that are invoked to construct objects.

```
Circle() {  
}
```

```
Circle(double newRadius) {  
    radius = newRadius;  
}
```

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 defined in the class. This constructor, called *a default constructor*, is provided automatically *only if no constructors are explicitly defined in the class*.

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


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

Assign object reference

Create an object

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()`

Trace Code

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

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

Declare myCircle

myCircle

no value

Trace Code, cont.

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

myCircle

no value

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

<u>: Circle</u>
radius: 5.0

Create a circle

Trace Code, cont.

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

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

Assign object reference
to myCircle

myCircle

reference value

: Circle

radius: 5.0

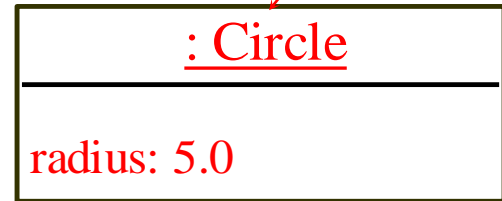
Trace Code, cont.

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

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle reference value



yourCircle no value

Declare yourCircle

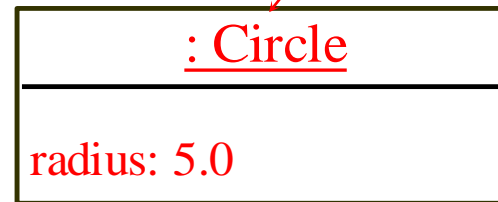
Trace Code, cont.

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

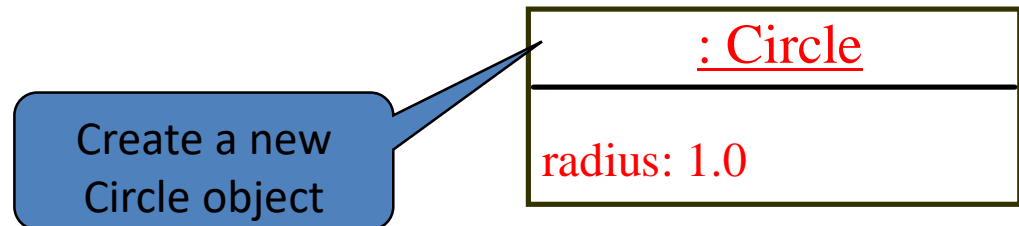
```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle reference value



yourCircle no value



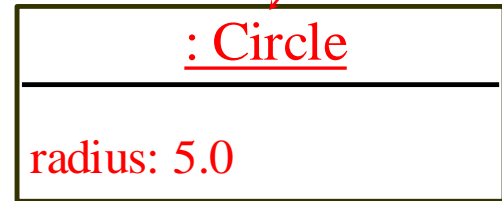
Trace Code, cont.

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

```
Circle yourCircle = new Circle();
```

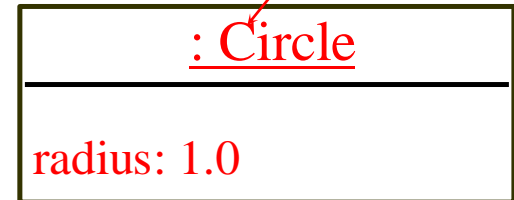
```
yourCircle.radius = 100;
```

myCircle reference value



yourCircle reference value

Assign object reference
to yourCircle



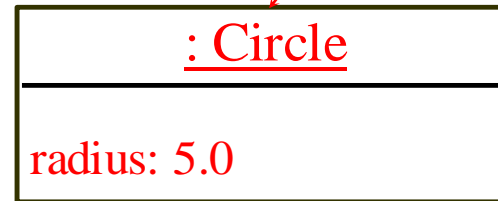
Trace Code, cont.

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

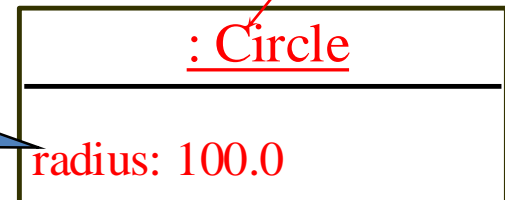
```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle reference value



yourCircle reference value



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 `SimpleCircle.getArea()`? 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)` (e.g., `myCircle.getArea()`).

More explanations will be given in the section on “Static Variables, Constants, and Methods.”

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.

```
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; // c has default value '\u0000'  
}
```

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

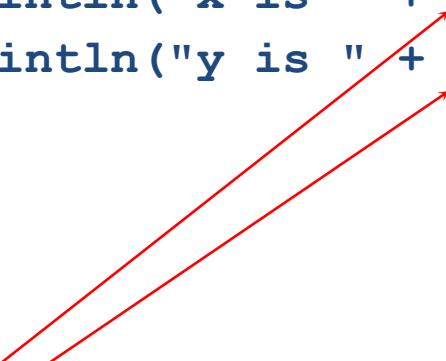
The default value of a data field is null for a reference type, 0 for a numeric type, false for a boolean type, and '\u0000' for a char type. However, Java assigns no default value to a local variable inside a method.

```
public class Test {  
    public static void main(String[] args) {  
        Student student = new Student();  
        System.out.println("name? " + student.name);  
        System.out.println("age? " + student.age);  
        System.out.println("isScienceMajor? " + student.isScienceMajor);  
        System.out.println("gender? " + student.gender);  
    }  
}
```

Example

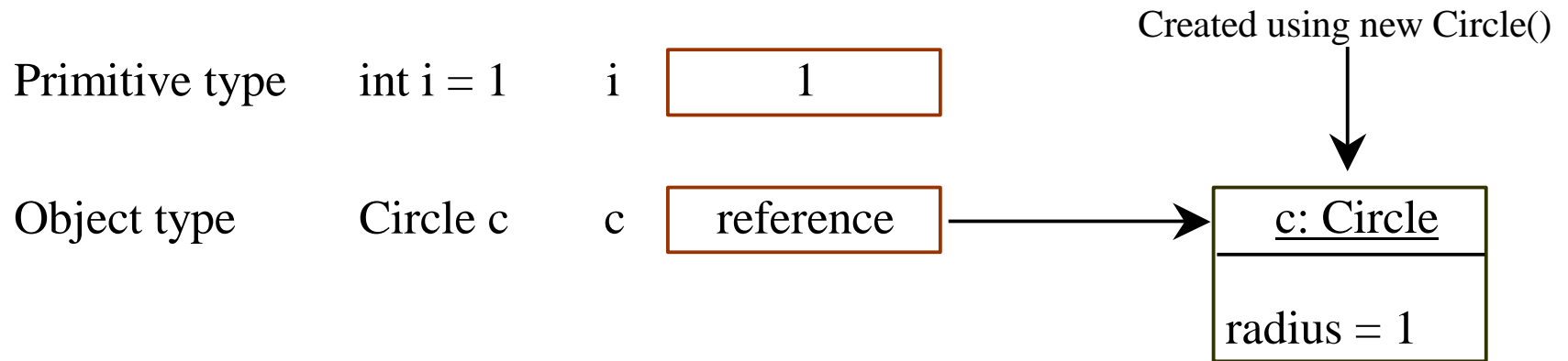
Java assigns no default value to a local variable inside a method.

```
public class Test {  
    public static void main(String[] args) {  
        int x; // x has no default value  
        String y; // y has no default value  
        System.out.println("x is " + x);  
        System.out.println("y is " + y);  
    }  
}
```



Compile error: variable not
initialized

Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types

Primitive type assignment $i = j$

Before:

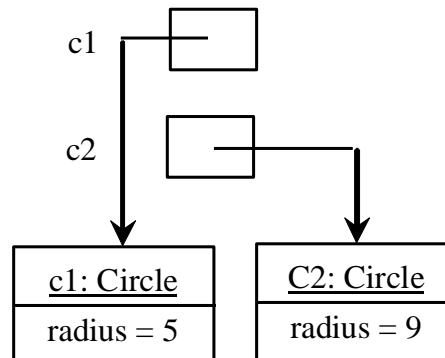


After:

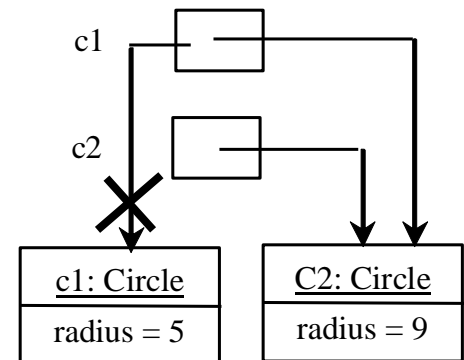


Object type assignment $c1 = c2$

Before:



After:



Garbage Collection

As shown in the previous figure, 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.

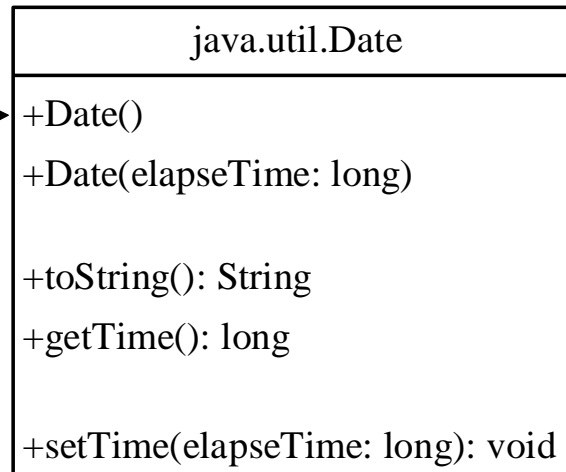
Garbage Collection, cont

TIP: 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 .

The Date Class

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

The + sign indicates
public modifier



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

For example, the following code

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

displays a string like Sun Mar 09 13:50:19
EST 2003.

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.

java.util.Random	
+Random()	Constructs a Random object with the current time as its seed.
+Random(seed: long)	Constructs a Random object with a specified seed.
+nextInt(): int	Returns a random int value.
+nextInt(n: int): int	Returns a random int value between 0 and n (exclusive).
+nextLong(): long	Returns a random long value.
+nextDouble(): double	Returns a random double value between 0.0 and 1.0 (exclusive).
+nextFloat(): float	Returns a random float value between 0.0F and 1.0F (exclusive).
+nextBoolean(): boolean	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.

```
Random random1 = new Random(3);  
System.out.print("From random1: ");  
for (int i = 0; i < 10; i++)  
    System.out.print(random1.nextInt(1000) + " ");  
Random random2 = new Random(3);  
System.out.print("\nFrom random2: ");  
for (int i = 0; i < 10; i++)  
    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

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.

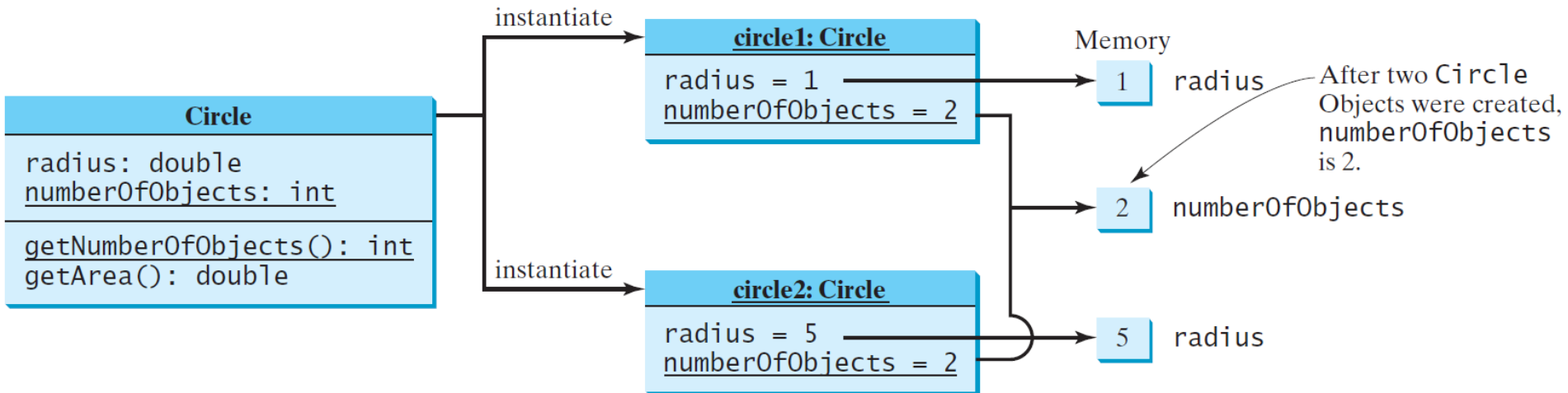
Static Variables, Constants, and Methods, cont.

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

Static Variables, Constants, and Methods, cont.

UML Notation:

underline: static variables or methods



Example of Using Instance and Class 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.

```
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 + ")");
    }
}

```

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.

```

package p1;

public class C1 {
    public int x;
    int y;
    private int z;

    public void m1() {
    }
    void m2() {
    }
    private void m3() {
    }
}

```

```

package p1;

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 {
    ...
}

```

```

package p1;

public class C2 {
    can access C1
}

```

```

package p2;

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

```

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.

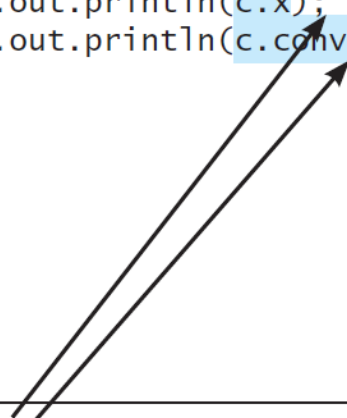
NOTE

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).

```
public class C {  
    private boolean x;  
  
    public static void main(String[] args) {  
        C c = new C();  
        System.out.println(c.x);  
        System.out.println(c.convert());  
    }  
  
    private int convert() {  
        return x ? 1 : -1;  
    }  
}
```

(a) This is okay because object **c** is used inside the class **C**.

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



(b) This is wrong because **x** and **convert** are private in class **C**.

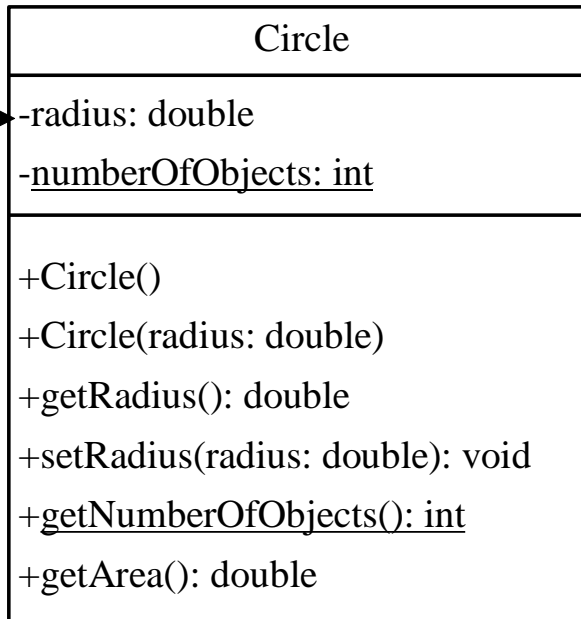
Why Data Fields Should Be private?

To protect data.

To make code easy to maintain.

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());
    }
}
```

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)

```

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

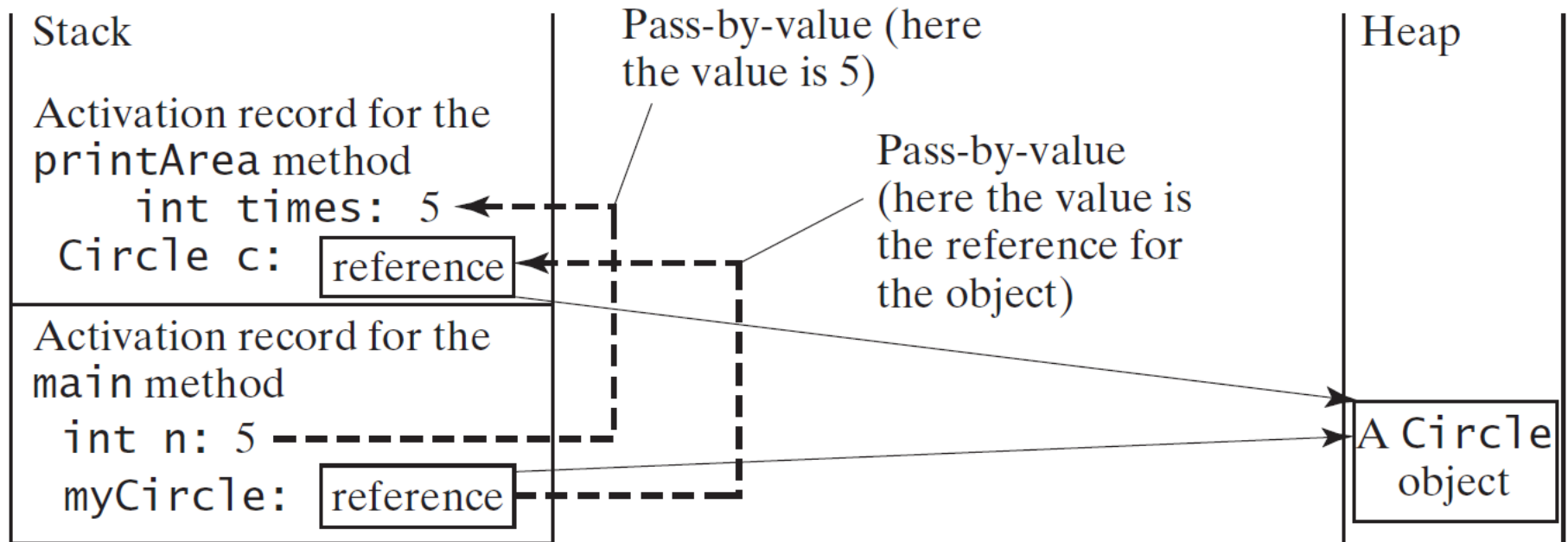
        // Print areas for radius 1, 2, 3, 4, and 5.
        int n = 5;
        printAreas(myCircle, n);

        // 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--;
        }
    }
}

```

Passing Objects to Methods, cont.



Immutable Objects and Classes

If the contents of an object cannot be changed once the object is created, the object is called an *immutable object* and its class is called an *immutable class*. If you delete the set method in the Circle class in Listing 8.10, the class would be immutable because radius is private and cannot be changed without a set method.

A class with all private data fields and without mutators is not necessarily immutable. For example, the following class Student has all private data fields and no mutators, but it is mutable.

Example

```
public class Student {
    private int id;
    private BirthDate birthDate;

    public Student(int ssn,
        int year, int month, int day) {
        id = ssn;
        birthDate = new BirthDate(year, month, day);
    }

    public int getId() {
        return id;
    }

    public BirthDate getBirthDate() {
        return birthDate;
    }
}
```

```
public class BirthDate {
    private int year;
    private int month;
    private int day;

    public BirthDate(int newYear,
        int newMonth, int newDay) {
        year = newYear;
        month = newMonth;
        day = newDay;
    }

    public void setYear(int newYear) {
        year = newYear;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Student student = new Student(111223333, 1970, 5, 3);
        BirthDate date = student.getBirthDate();
        date.setYear(2010); // Now the student birth year is changed!
    }
}
```

What Class is Immutable?

For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.

Scope of Variables

- ❑ The scope of instance and static variables is the entire class. They can be declared anywhere inside a class.
- ❑ 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 initialized explicitly before it can be used.

The this Keyword

- ❑ The this keyword is the name of a reference that refers to an object itself. One common use of the this keyword is reference a class's *hidden data fields*.
- ❑ Another common use of the this keyword to enable a constructor to invoke another constructor of the same class.

Reference the Hidden Data Fields

```
public class F {  
    private int i = 5;  
    private static double k = 0;  
  
    void setI(int i) {  
        this.i = i;  
    }  
  
    static void setK(double k) {  
        F.k = k;  
    }  
}
```

Suppose that f1 and f2 are two objects of F.
F f1 = new F(); F f2 = new F();

Invoking f1.setI(10) is to execute
 this.i = 10, where **this** refers f1

Invoking f2.setI(45) is to execute
 this.i = 45, where **this** refers f2

Calling Overloaded Constructor

```
public class Circle {  
    private double radius;
```

```
    public Circle(double radius) {  
        this.radius = radius;  
    }
```

→ this must be explicitly used to reference the data field radius of the object being constructed

```
    public Circle() {  
        this(1.0);  
    }
```

→ this is used to invoke another constructor

```
    public double getArea() {  
        return this.radius * this.radius * Math.PI;  
    }  
}
```

↓
Every instance variable belongs to an instance represented by this, which is normally omitted