Programming in the Large II: Objects and Classes (Part 1)

188230 Advanced Computer Programming

Asst. Prof. Dr. Kanda Runapongsa Saikaew
(krunapon@kku.ac.th)
Department of Computer Engineering
Khon Kaen University

Agenda

- OO Programming Concepts
- Declaring and Creating Objects
- Constructors
- Modifiers
- Instances and Class Variables and Methods
- Programming with Objects

OOP vs. Task

- Object-oriented programming (OOP) represents an attempt to make programs more closely model the way people think about the world
- In the older styles of programming, a programmer identifies a task that needs to be performed in order to solve the problem.
- But at the heart of OOP, instead of tasks we find objects—entities that have behaviors, that hold information, and that can interact with one another

OOP View

- We can think of an object in standard programming terms as nothing more than a set of variables together with some methods for manipulating those variables.
- What is a class?
- What is an object?
- What is an instance method?
- What is an instance variable?

What is a Class?

- A class is a kind of factory for constructing objects.
- Classes are used to create objects
- Objects are created and destroyed as the program runs
- There can be many objects with the same structure, if they are created using the same class

Class vs. Object

- A class is a type but the object is a value of that type
  - String message;
  - // String is a class and it is also a type
  - // message is an object that its type is String
- There can be many objects in the same class
- An object is a class instance.
  - String msg1, msg2;
  - // msg1 is a class instance, msg2 is also a class instance

Sample Class UserData

```
class UserData {
        static String name;
        static int age;
}
```

- In a program that uses this class, there is only one copy of each of the variables UserData.name and UserData.age.
- There can only be one “user,” since we only have memory space to store data about one user.

Sample Class PlayerData

```
class PlayerData {
        String name;
        int age;
}
```

- In this case, there is no such variable as PlayerData.name or PlayerData.age, since name and age are not static members of PlayerData.
- It can be used to create any number of objects!
- Each object will have its own variables called name and age

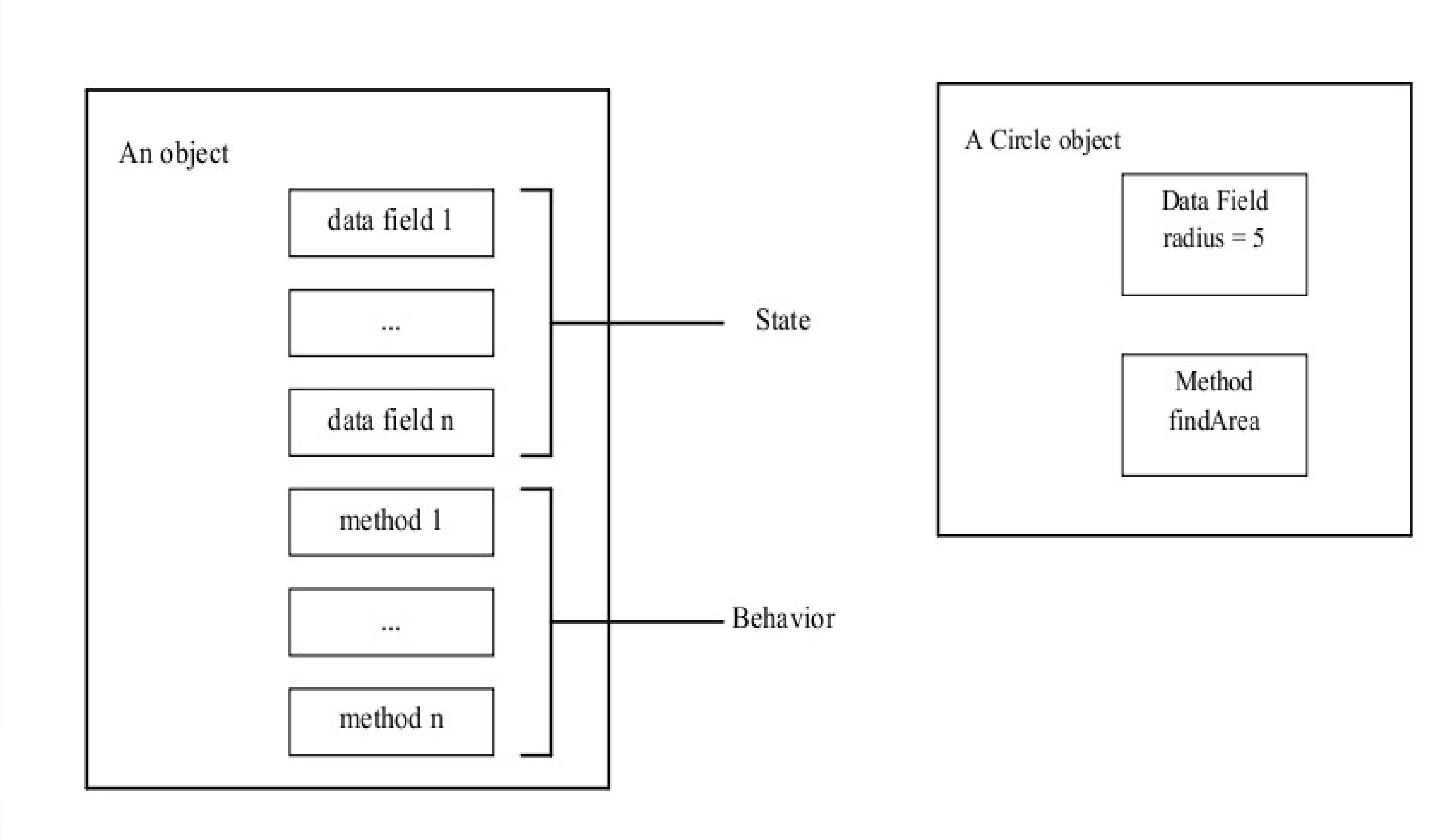
Instance Variables and Methods

- An object that belongs to a class is said to be an instance of that class
- The variables that the object contains are called instance variables
- The methods that the object contains are called instance methods.
- PlayerData jordan;
- jordan is an object or an instance of class PlayerData
- jordan.name and jordan.age are instance variables

Class Variables and Methods

- Static member variables are sometimes called class variables
- Static member methods in a class are sometimes called class methods
- Static member variables and methods are belong to the class itself, rather than to instances of that class
- UserData.name and UserData.age are static member variables or class variables

OO Programming Concepts



Object

- What is an object composed of?
  - Attribute or data: information about an object, different object can have different information
  - Behavior or method: what object can do
- Object examples:
  - A student is an object with attributes: ID, name, GPA and can perform actions: register, walk, run
  - A car is an object with attributes: color, model and can take actions: break, start, accelerate

Class

- A class is like a blueprint of objects
- An object created by a class sometimes called an instance of a class
- There can be any number of objects that are in the same class
- Example: there can be objects circle1, circle2, circle3, ... where these objects are belong to class Circle

Class Declaration

```
class Circle {
   double radius;
   double findArea() {
      return radius*radius*Math.PI;
}
```

- Is radius a class variable?
- Is findArea() a class method?

Object Declaration

- Syntax:
  - ClassName objectName;
- Example:
  - Circle c1;
- Declaring a variable does not create an object!
- In Java, no variable can ever hold an object
- A variable can only hold a reference or an address to an object
- Can we do this?
  - c1.radius = 10;

Creating an Object

- In a program, objects are created using an operator called new, which creates an object and returns a reference to that object
- Syntax
  - objectName = new ClassName();
- Example

```
c1 = new Circle();
```

  - Create a new object which is an instance of the class Circle
  - Store a reference to that object in the variable c1
  - The variable c1 refers to the object

Declaring/Creating an Object in a Single Step

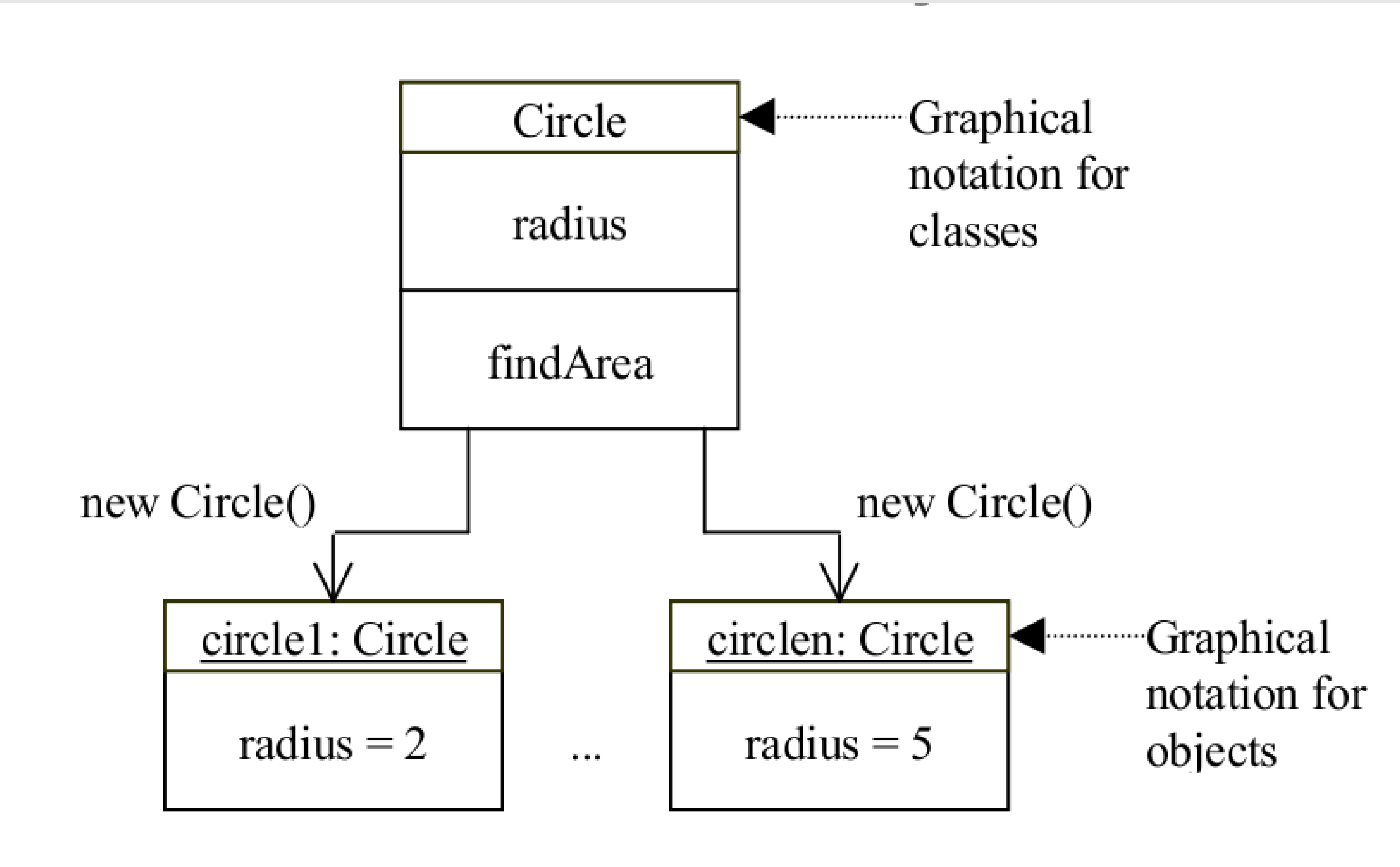
- Syntax:

```
ClassName objectName = new className();
```

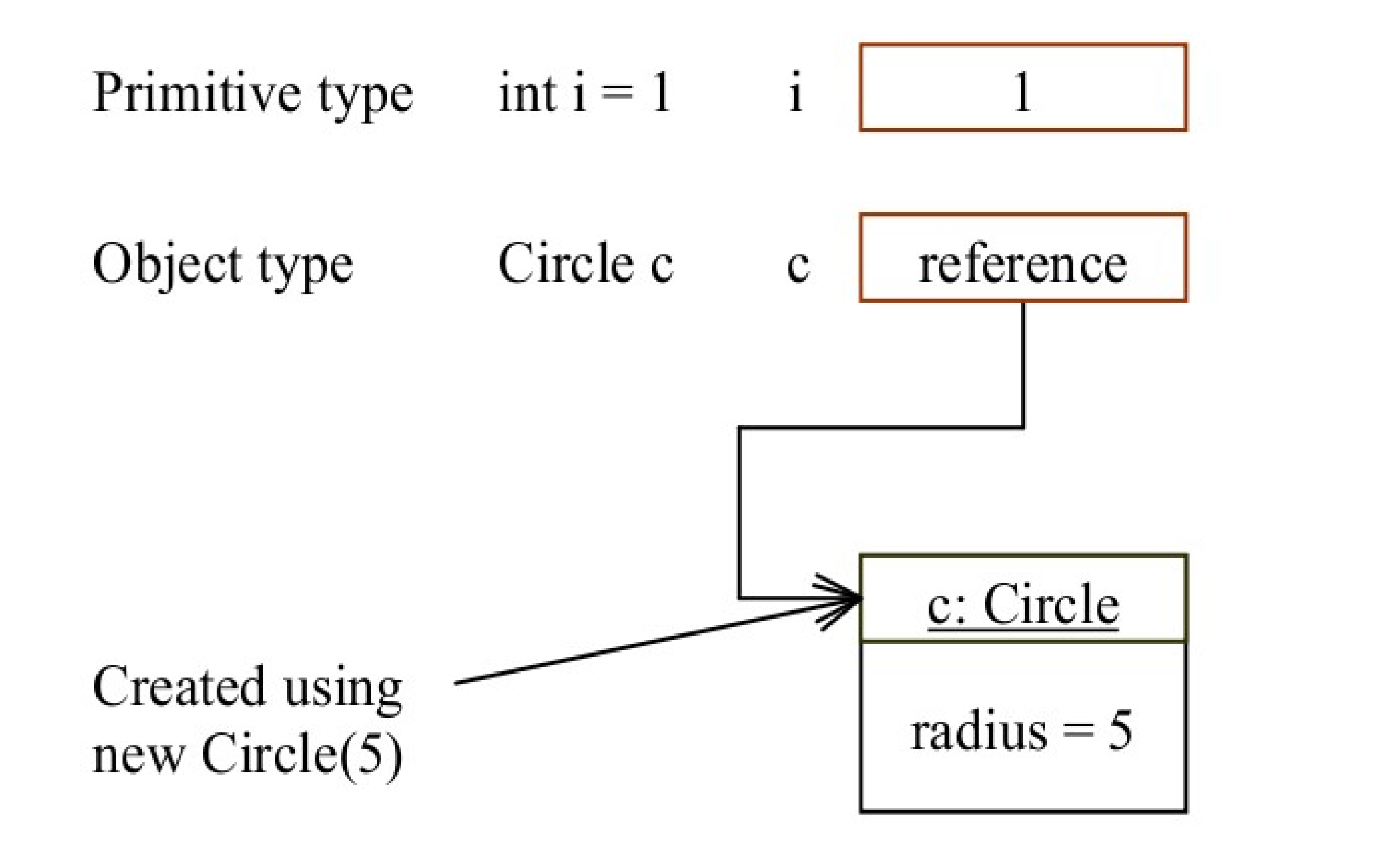
- Example

```
Circle c1 = new Circle();
// Now we can do
c1.radius = 10;
```

Class and Objects



Differences between Variables of Primitive Types and Object Types



Class Student

```
public class Student {
    public String name; // Student’s name.
    public double test1, test2, test3;   // Grades on three tests
    public double getAverage() { // compute average test grade
            return (test1 + test2 + test3) / 3;
    }
}  // end of class Student
```

The null Reference

- It is possible for a variable like std, whose type is given by a class, to refer to no object
- The null reference is written in Java as “null”
  - std = null
- You could test whether the value of std is null by testing
  - if (std == null) . . .

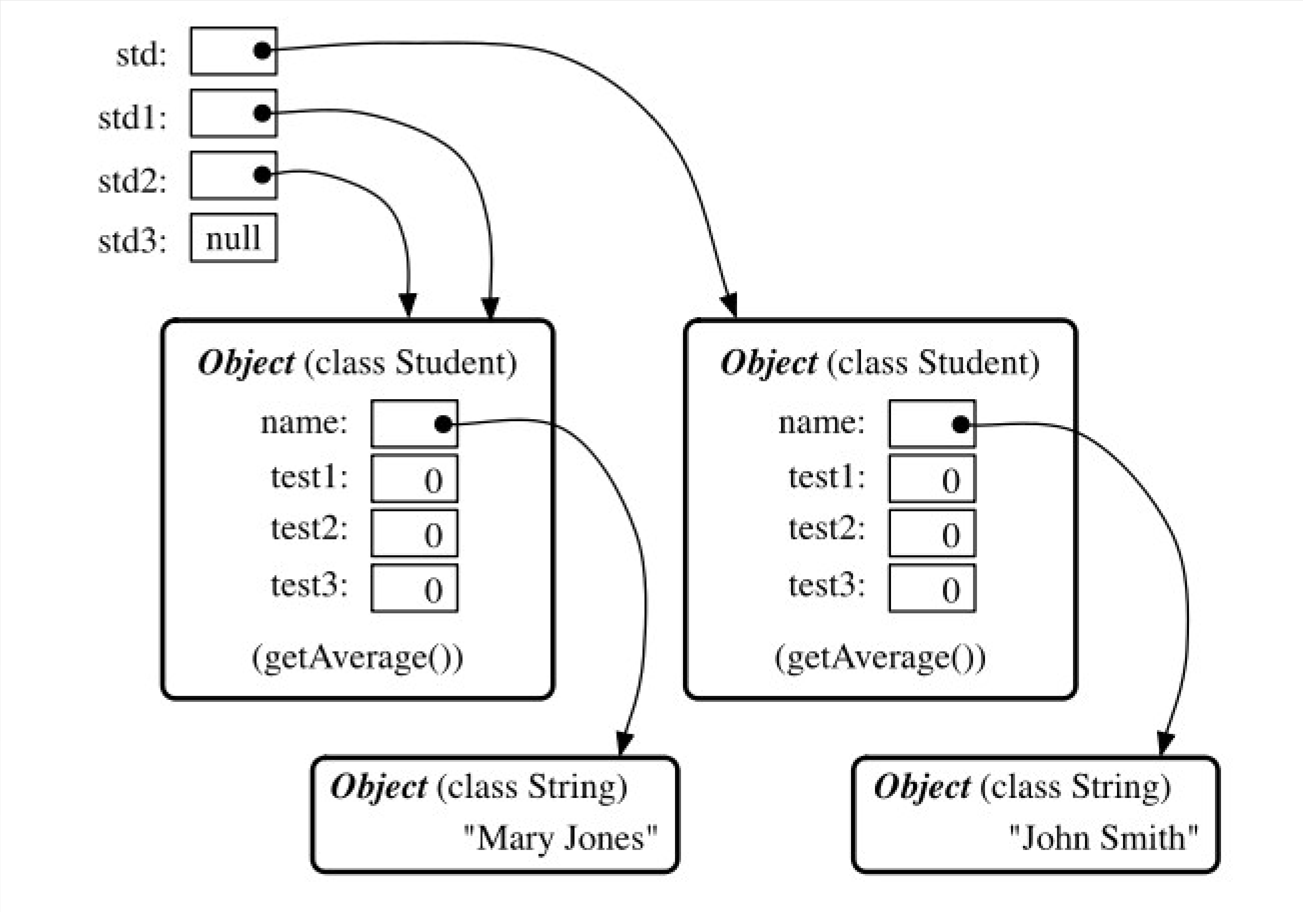
Sample Code in Class Student

```
// Declare four variables of  type Student.
Student std, std1,  std2, std3;
/* Create a new object belonging to the class
Student, and store a reference to that object in the
variable std. */
std = new Student();
/* Create a second Student object and store a
reference to it in the variable std1. */
std1 = new Student();
```

Sample Code in Class Student

```
/* Copy the reference value in std1 into the
variable std2. */
std2 = std1;
// Store a null reference in the variable std3.
std3 = null;
std.name = "John Smith"; // Set values of some instance variables.
std1.name = "Mary Jones";
// (Other instance variables have default  initial values of zero.)
```

Object References in Memory



Object Assignments

- When one object variable is assigned to another, only a reference is copied.
- The object referred to is not copied.
- When the assignment “std2 = std1;” was executed, no new object was created
- Instead, std2 was set to refer to the very same object that std1 refers to

Variables and Objects

- For example, std1.name and std2.name are two different names for the same variable, namely the instance variable in the object that both std1 and std2 refer to
- After the string "Mary Jones" is assigned to the variable std1.name, it is also true that the

  value of std2.name is "Mary Jones"The object is not in the variable.
- The object is not in the variable.
- The variable just holds a pointer to the object

Testing Equality Operator

- You can test objects for equality and inequality using the operators == and !=
- The semantics are different from what you are used to
- When you make a test “if (std1 == std2)”
  - You thought that you were testing whether the values stored in std1 and std2 are the same
  - But the values are references to objects, not objects
  - It tests whether they point to the same location

Method Parameters

- Let’s consider what happens when obj is passed as an actual parameter to a subroutine. is executed
- The subroutine has no power to change the value stored in the variable
- It only has a copy of that value
- However, it can change the data stored in the object
- After the subroutine ends, obj still points to the same object, but the data stored in the object might have changed.

Sample Method Parameters

```
static void method1(int z) {
    z = 42;
}

static void method2(Person p) {
    p.name = "Chanapat";
}
```

Sample Method Parameters

```
public static void main(String[] args) {
    int x = 17;
    method1(x);
    System.out.println(x);
    Person p = new Person();
    p.name = "Ta";
    method2(p);
    System.out.println(p.name);
}
```

Public and Private Members

- When writing new classes, it’s a good idea to pay attention to the issue of access control.
- Making a member of a class public makes it accessible from anywhere
- A private member can only be used in the class where it is defined.
- In the opinion of many programmers, almost all member variables should be declared private
  - This gives you complete control over what can be done with the variable.

Getter Method

- You can allow other classes to find out what its value is by providing a public accessor method that returns the value of the variable.
- Accessor methods are more often referred to as getter methods
- A getter method provides “read access” to a variable

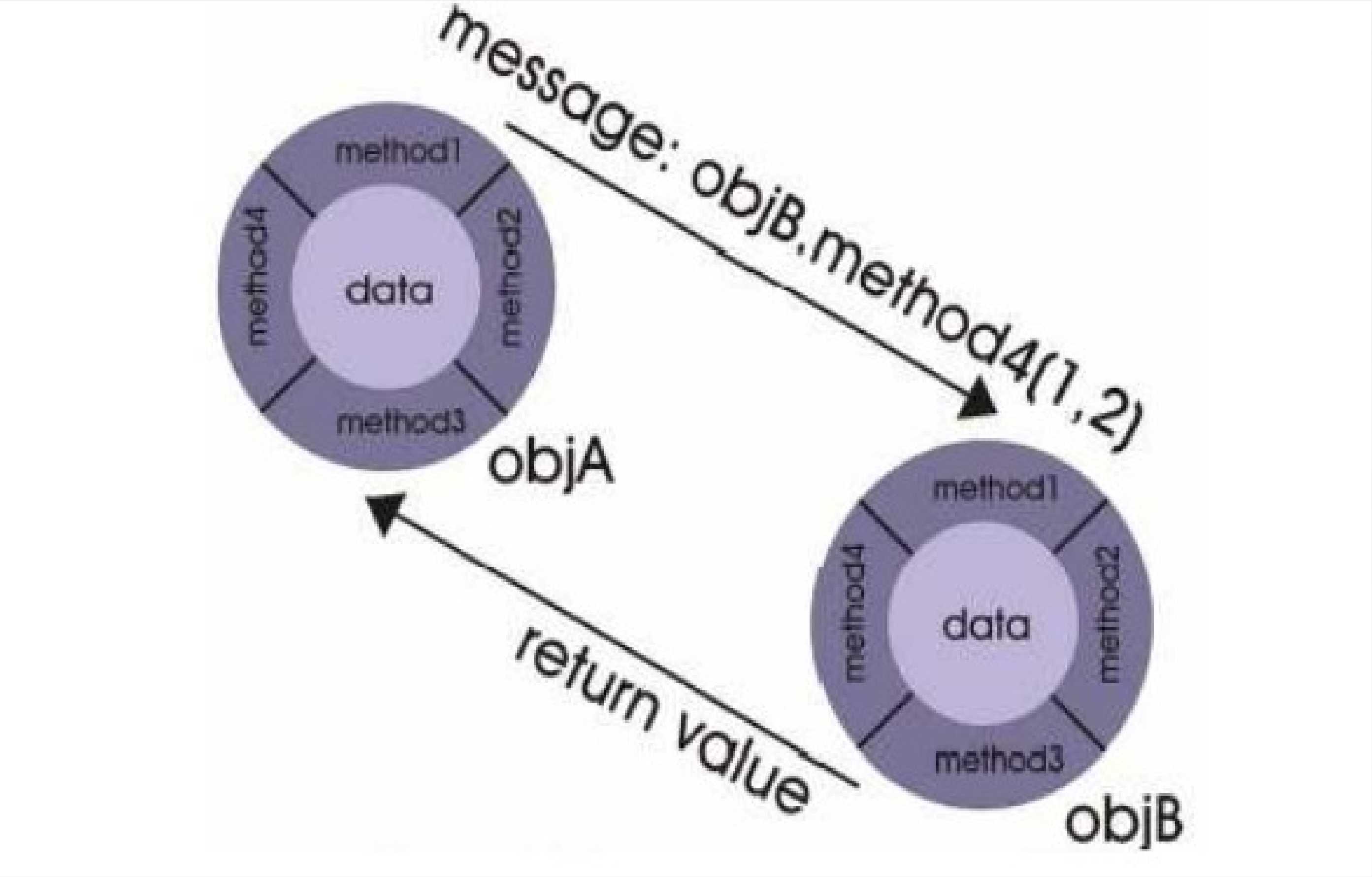
```
public String getTitle() {
   return title;
}
```

Setter Method

- You might also want to allow “write access” to a private variable
- That is, you might want to make it possible for other classes to specify a new value for the variable
- This is done with a setter method

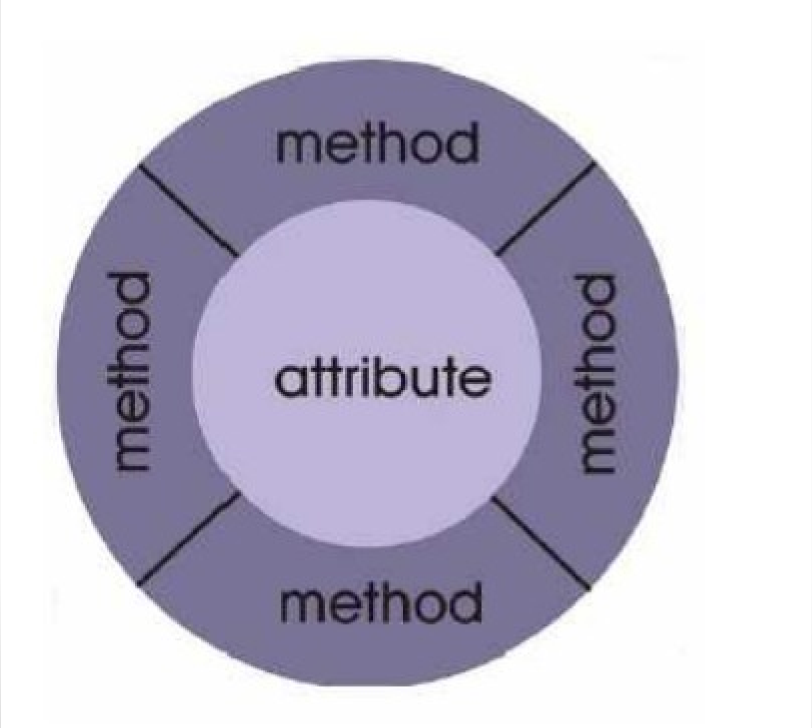
```
public void setTitle( String newTitle ) {
   title = newTitle;
}
```

Communications Done by Sending Messages to Each Other



Encapsulation

- Use object's functions by calling methods
- Encapsulation is done by
  - Declaring attributes as private
  - Declaring methods as public



Sample Code TestCircle

```
class Circle2 {
private double radius = 1.0;
// Find the area of this circle
public double findArea() { return radius*radius*Math.PI; }
public double getRadius() { return radius; }
public void setRadius(double newR) { radius = newR; }}
public class TestCircle {
public static void main(String[] args) {
Circle2 c = new Circle2();
System.out.println("The area of the circle " + " of radius " +
c.getRadius() + " is " + c.findArea()); }}
```

Constructing Objects

- Object types in Java are very different from the primitive types
- Simply declaring a variable whose type is given as a class does not automatically create an object of that class
- Objects must be explicitly constructed
  - Finding some unused memory in the heap that can be used to hold the object
  - Filling in the object’s instance variables

Constructors

- Objects are created with the operator, new

```
// Declare a variable of type Circle
Circle myCircle;
/* Allocates memory for the object
   Initializes the object’s instance variables
   Returns a reference to the object. */
myCircle = new Circle();
```

Default Constructor

- Every class has at least one constructor
- If the programmer doesn’t write a constructor definition in a class
  - The system will provide a default constructor for that class
  - This default constructor does nothing beyond the basics: allocate memory and initialize instance variables
- But you can include one or more constructors in the class definition

Definition of a Constructor

- The definition of a constructor looks much like the definition of any other methods excepts
  - A constructor does not have any return type (not even void)
  - The name of the constructor must be the same as the name of the class in which it is defined
  - A constructor can’t be declared static

Sample Code Using Constructors

```
class Circle3 {
private double radius;
Circle3(double r) {
radius = r;
}
Circle3() {
radius = 1.0;
}...
}
```

Sample Code Using Constructors

```
public static void main(String[] args) {
Circle3 c1 = new Circle3(5.0);
System.out.println("The area of the circle" +
c1.getRadius() + " is " +
c1.findArea());
Circle3 c2 = new Circle3();
System.out.println("The area of the circle" +
c2.getRadius() + " is " +
c2.findArea());}
```

Access Modifiers and Levels

- 3 access modifiers: public, protected, private
- 4 access levels: public, protected, default, private
- Class can be declared as only public or default
- Class declared as public

  public class HelloWorld
- Class declared as default

  class HelloWorld

Access Modifiers

- By default, the class, variable, or data 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 method can be accessed only by the declaring class
- protected
  - The data or method is visible to any subclass

Access Modifiers Diagram

Modifier	Same class	Same package	Subclass	Any class
public	/	/	/	/
protected	/	/	/	
default	/	/		
private	/			

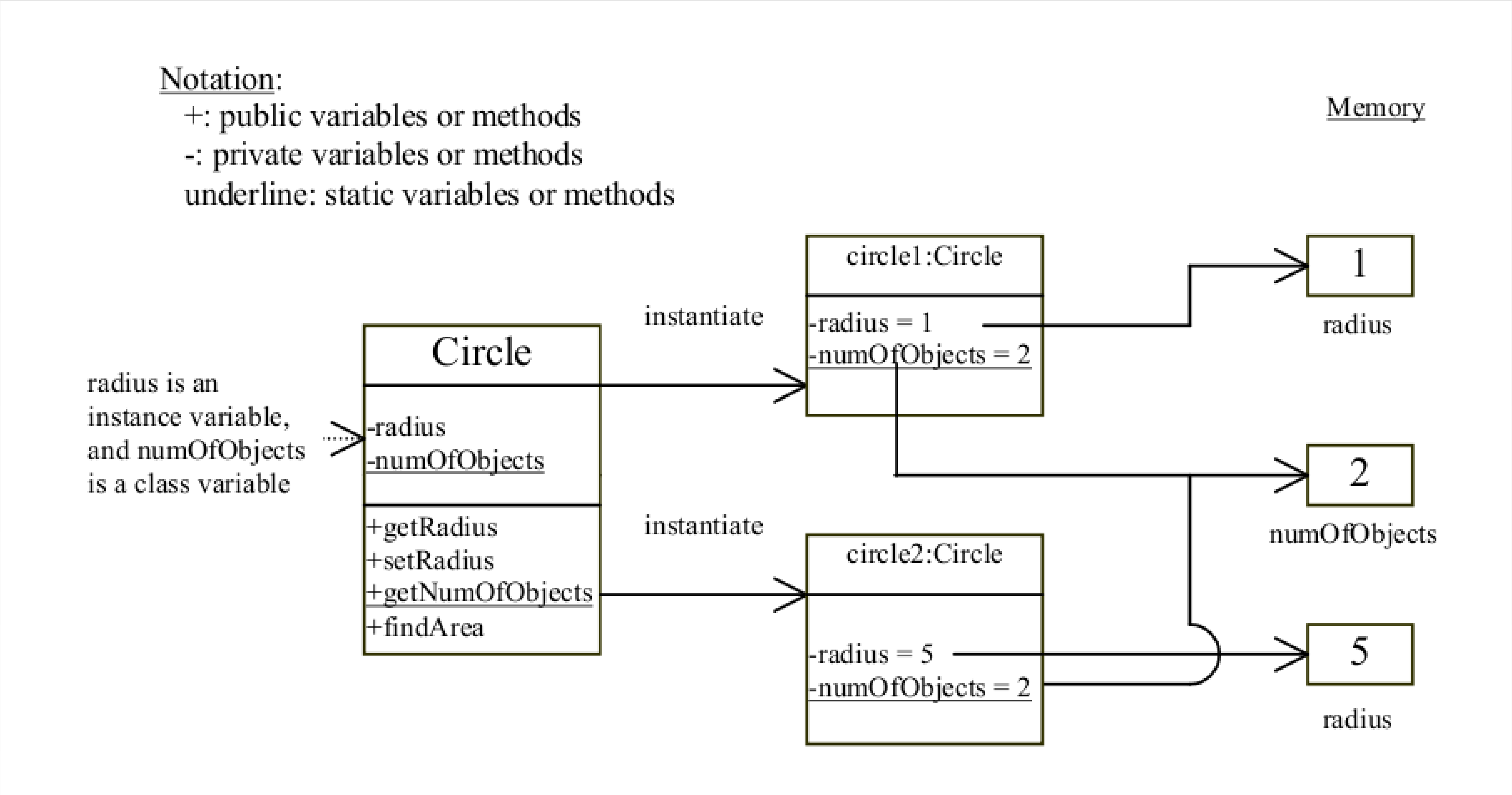
Instance Variables and Methods

- Instance variables are belong to a specific instance
- Instance methods are invoked by an instance of the class

Class Variables, Constants, and Methods

- Class variables are shared by all the instances of the class
- Class methods are not tied to a specific object
- Class constants are final variables shared by all the instances of the class
- To declare class variables, constants, and methods, use the static modifier

Sample Class & Instance Members



Notation:

+: public variables or methods

-: private variables or methods

underline: static variables or methods

Memory

radius is an instance variable, and numOfObjects is a class variable

Circle

-radius

-numOfObjects

+getRadius

+setRadius

+getNumOfObjects

+findArea

instantiate

circle1:Circle

-radius = 1

-numOfObjects = 2

instantiate

circle2:Circle

-radius = 5

-numOfObjects = 2

1

radius

2

numOfObjects

5

radius

TestClassAndInstanceVariables

```
class Circle4 {
     // instance variable
     private double radius;
    // class variable
    private static int numCircles = 0;
    // default constructor
    public Circle4() {
        radius = 1.0;
        numCircles++;
    }
```

TestClassAndInstanceVariables

```
// construct a circle with a specified radius
public Circle4(double r) {
        radius = r;
        numCircles++;
}
public static int getNumCircles() {
         return numCircles;
}
public double getRadius() {
         return radius;
}
```

TestClassAndInstanceVariables

```
public class TestClassAndInstanceVariables {
public static void main(String[] args) {
Circle4 c1 = new Circle4();
System.out.println("c1:");
printCircle(c1);
Circle4 c2 = new Circle4(5);
System.out.println("c2:");
printCircle(c2);
}
```

TestClassAndInstanceVariables

```
public static void printCircle(Circle4 c) {
System.out.println("radius(" + c.getRadius()
+ ") while number of circles is " +
Circle4.getNumCircles());
}
```

- What is the output?

Some Java Built-in Classes

- It’s important not to forget that the designers of Java have already provided a large number of reusable classes
- Some classes are meant to be extended to produce new classes
- Some classes can be used directly to create useful objects

Using “+” for String is Inefficient

- It's not efficient to build up a longer string using the + operator
- Example:
  - String str = “Hello”;
  - String msg = str + “ World”;
  - Creating a whole new string that is a copy of str, with the value of “ World” appended onto the end
- Copying the string takes some time

StringBuffer

- The class StringBuffer makes it possible to be efficient about building up a long string from a number of smaller pieces
- To do this, you must make an object belonging to the StringBuffer class.
  - StringBuffer buffer = new StringBuffer();
  - buffer.append(“Hello”);
  - buffer.append(“ World”);
  - System.out.println(buffer.toString());

Classes in Package java.util

A number of useful classes are collected in the package java.util

- For example, this package contains classes for working with collections of objects
- We will study these collection classes later
- Some useful classes
  - java.util.Scanner: to scan stream
  - java.util.Random: to generate random numbers

java.util.Scanner

- A simple text scanner which can parse primitive types and strings using regular expressions.
- A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace
- The resulting tokens may then be converted into values of different types using the various next methods.

Sample Code Using Scanner

```
import java.util.Scanner;
public class ScannerDemo {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
double[] nums = new double[10];
int i = 0;
while (sc.hasNextDouble()) {
    nums[i] = sc.nextDouble();
    i++; }
int numDoubles = i;
System.out.println("num doubles is " +
numDoubles);
for (i = 0; i < numDoubles; i++)
System.out.print(nums[i] + " ");
}
}
```

- What is the output if we type?

  1 3 a 4

java.util.Random

- An object of type Random can generate random integers, as well as random real numbers.
- If randGen is created with the command:
- Random randGen = new Random();
- if N is a positive integer, then randGen.nextInt(N) generates a random integer in the range from 0 to N-1

Sample Code Using Random

```
import java.util.Random;
public class RandomDemo {
    public static void main(String[] args) {
        Random rand = new Random();
         for (int i = 0; i < 6; i++) {
           System.out.print((rand.nextInt(6) + 1) + " ");
        }
    }
}
```

Wrapper Classes

- We have already encountered the classes Double and Integer
  - These classes contain the static methods Double.parseDouble and Integer.parseInteger that are used to convert strings to numerical values
- There is a similar class for each of the other primitive types, Long, Short, Byte, Float, and Boolean
- These classes are called wrapper classes
- They are used for creating objects that represent primitive type values

Primitive Types and Wrapper Classes

- Primitive types are not classes
- Values of primitive type are not objects
- Sometimes it’s useful to treat a primitive value as if it were an object
- You can’t do that literally, but you can “wrap” the primitive type value in an object belonging to one of the wrapper classes.
  - Double d = new Double(3.14);
  - The value of d contains the same information as the value of type double, but it is an object

Autoboxing

- In Java 5.0, wrapper classes have become easier to use
- Java 5.0 introduced automatic conversion between a primitive type and the corresponding wrapper class
- If you use a value of type int in a context that requires an object of type Integer, the int will automatically be wrapped in an Integer object
  - Integer answer = 42;
  - Integer answer = new Integer(42);

WrapperClass Demo

```
public static void main(String[] args) {
    Double d = 3.14;
    System.out.println(d);
    Character c = '2';
    System.out.println(Character.isDigit(c));
    System.out.println(Integer.MAX_VALUE);
    System.out.println(Float.toString(2.1f));
}
```

Class Object

- Every class in Java (with just one exception) is a subclass of some other class
- If you create a class and don't explicitly make it a subclass of some other class, then it automatically becomes a subclass of the special class named Object
- Object is the one class that is not a subclass of any other class
- Class Object defines several instance methods that are inherited by every other class

Method toString() in Class Object

- The instance method toString() in class Object returns a value of type String that is supposed to be a string representation of the object
- The version of toString that is defined in Object just returns the name of the class that the object belongs to, concatenated with a code number called the hash code of the object
  - this is not very useful
- When you create a class, you can write a new toString() method for it, which will replace the inherited version

Class Rectangle

```
package coe.java.demos.c5;
class Rectangle {
protected double width;
protected double height;
Rectangle() { width = 2; height = 3;}
Rectangle(double w, double h) {width = w; height = h;}
public double getArea() { return width*height;}
public double getCircum() {return 2*width + 2*height;}
}
```

Class Square

```
class Square extends Rectangle {
Square() {width = 2; height = 2;}
Square(double w) {width = w; height = w; }
public String toString() {
StringBuffer buffer = new StringBuffer();
buffer.append("Square: Width = "); buffer.append(width);
buffer.append(" Height = "); buffer.append(height);
buffer.append(" Area = "); buffer.append(getArea());
buffer.append(" Circum = "); buffer.append(getCircum());
return buffer.toString();}}
```

MethodToStringDemo

```
public class MethodToStringDemo {
public static void main(String[] args) {
Rectangle r = new Rectangle(3,4);
System.out.println(r);
Square s = new Square(3);
System.out.println(s);}}
```

- What is the output?

References

- David J. Eck, ”Introduction to Programming Using Java”, Version 5.0, December 2006 http://math.hws.edu/javanotes/