VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

UNIVERSITY OF INFROMATION TECHNOLOGY

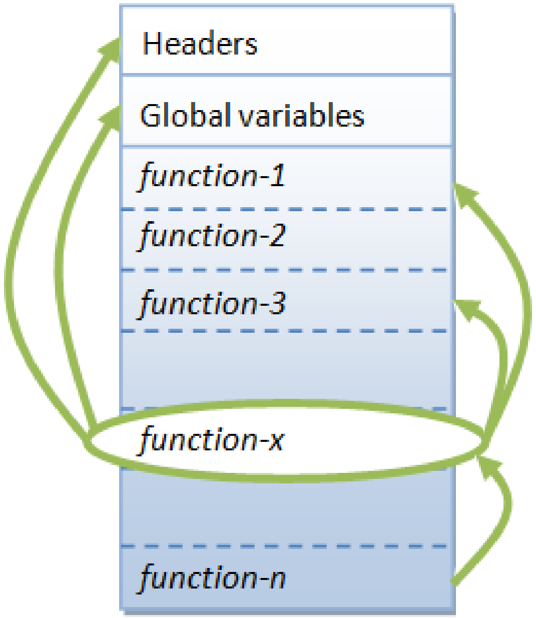
Chapter 3

JAVA OOP

Lecturer: MSc. Kiet Van Nguyen
Faculty of Information Science and Engineering
University of Information Technology, VNU-HCM

Recall

- Traditional procedural-oriented programming languages (such as C, Fortran, Cobol and Pascal)



A function (in C) is not well-encapsulated

Recall

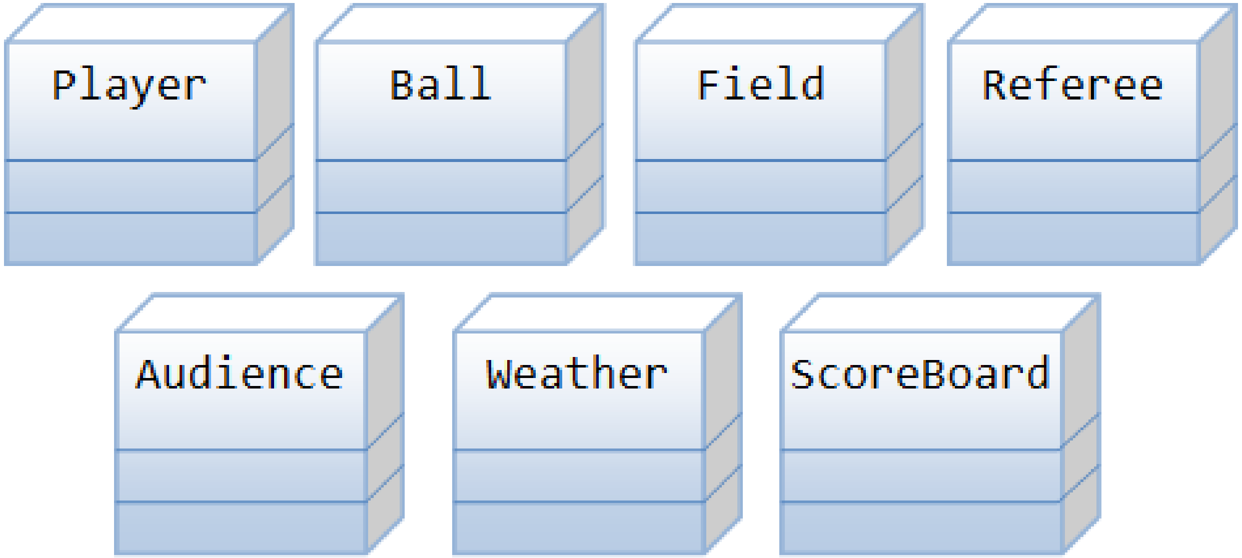
- Traditional procedural-oriented programming languages (such as C, Fortran, Cobol and Pascal)
  - ✓ Difficult to copy a function from one program and reuse in another program
  - ✓ Not suitable of high-level abstraction for solving real life problems.
  - ✓ Separate the data structures (variables) and algorithms (functions).

Why OOP?

- Object-oriented programming (OOP) languages are designed to overcome these problems.
- Well-encapsulated.
- *Higher level of abstraction* for solving real-life problems.

Why OOP?

- **For example,**



Classes (Entities) in a Computer Soccer Game

Why OOP?

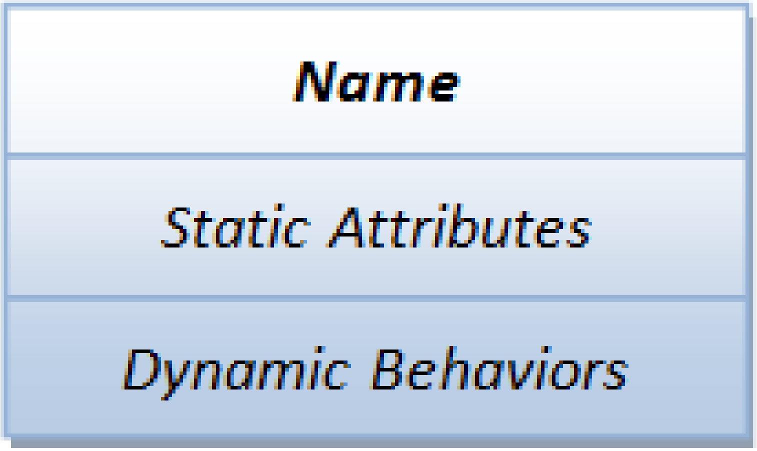
- **Benefits of OOP**
  - ✓ *Ease in software design*
  - ✓ *Ease in software maintenance*
  - ✓ *Reusable software*

OOP in Java

Class & Instances

- **A class** is a definition of objects of the same kind.
- **An instance** is a realization of a particular item of a class.
- For example, you can define a class called "**Student**" and create three instances of the class "Student" for "**Peter**", "**Paul**" and "**Pauline**".
- Note: The term "*object*" usually refers to *instance*.

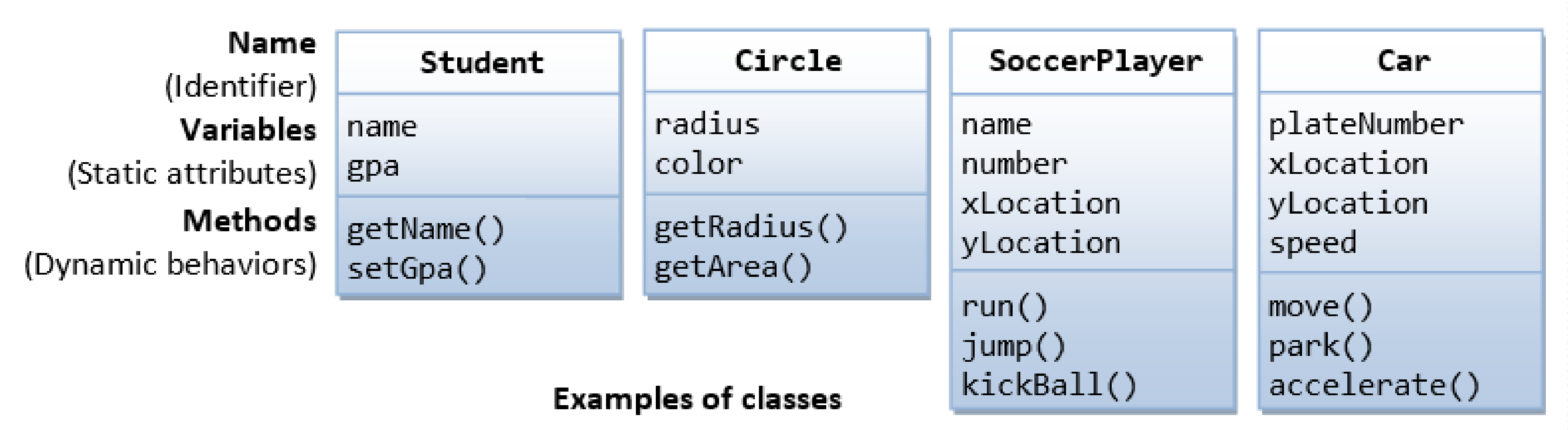
Class & Instances (Cont.)



A class is a 3-compartment box

Class & Instances (Cont.)

- **For example of classes**



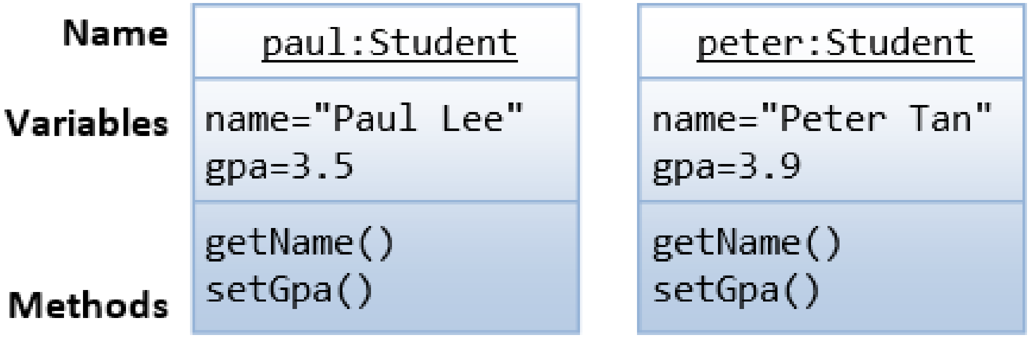
Name (Identifier)	**Student**	**Circle**	**SoccerPlayer**	**Car**
Variables (Static attributes)	name gpa	radius color	name number xLocation yLocation	plateNumber xLocation yLocation speed

Methods
(Dynamic behaviors)
	getName()
	setGpa()
		getRadius()
		getArea()
			run()
			jump()
			kickBall()
				move()
				park()
				accelerate()

Examples of classes

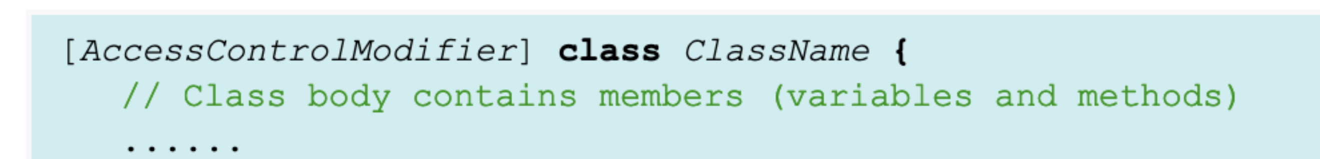
Class & Instances (Cont.)

- **For example of instances**



Two instances - paul and peter - of the class Student

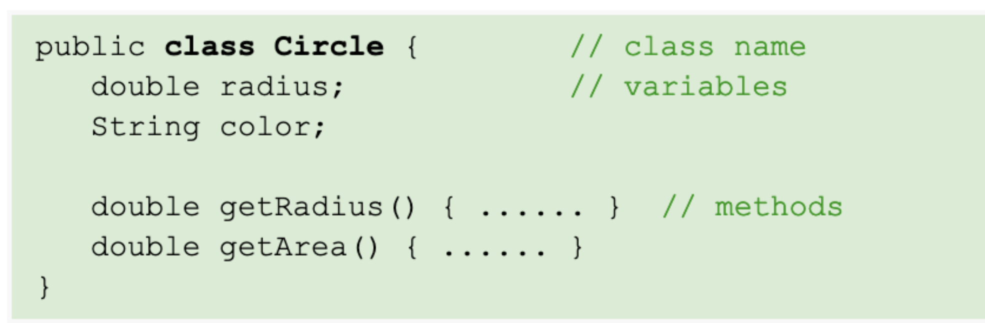
Class Definition



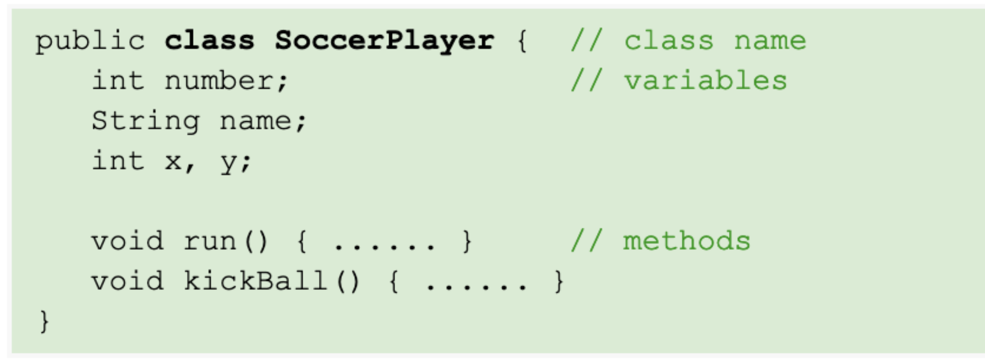
```
[AccessControlModifier] class ClassName {
   // Class body contains members (variables and methods)
   ......
}
```

Class Naming Convention:

- ✓ A class name shall be a noun or a noun phrase made up of several words.
- ✓ All the words shall be initial-capitalized (camel-case).



```
public class Circle {          // class name
   double radius;              // variables
   String color;

   double getRadius() { ...... }  // methods
   double getArea() { ...... }
}
```



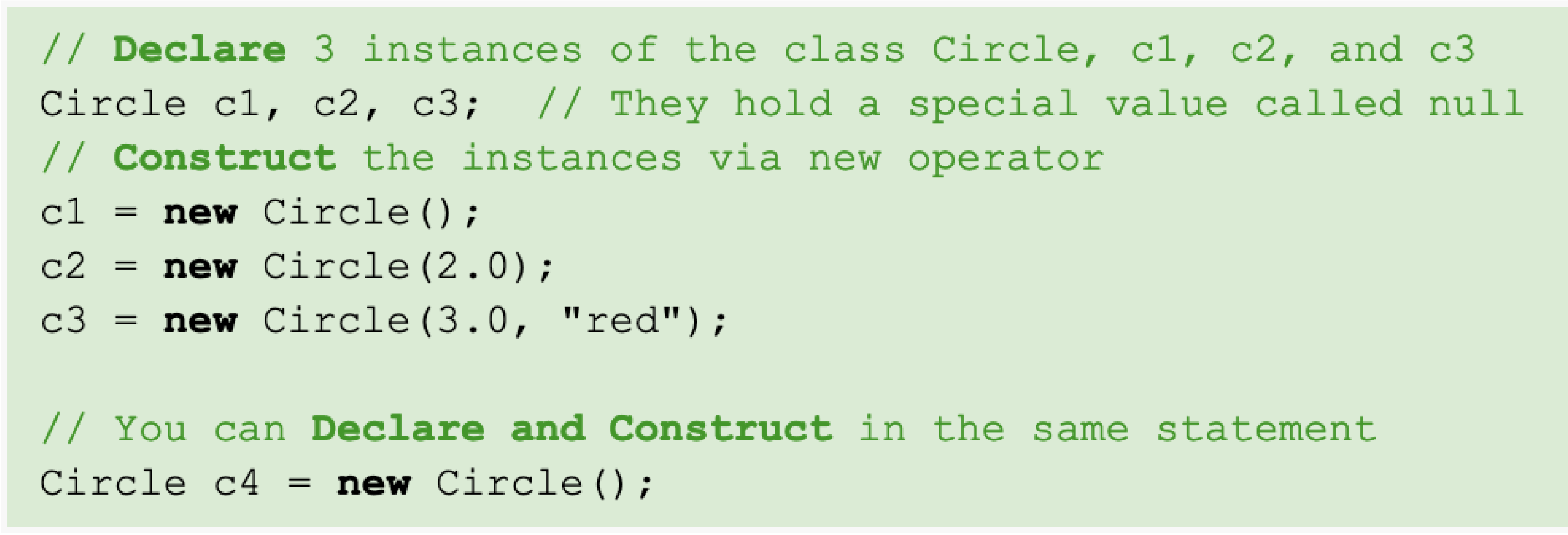
```
public class SoccerPlayer {   // class name
   int number;                 // variables
   String name;
   int x, y;

   void run() { ...... }       // methods
   void kickBall() { ...... }
}
```

Creating Instances of a Class

- To create *an instance of a class*, you have to:
  - ✓ **Declare** an instance identifier (instance name) of a particular class.
  - ✓ **Construct** the instance (i.e., allocate storage for the instance and initialize the instance) using the "new" operator.

Creating Instances of a Class (Cont.)

- For example,



```
// Declare 3 instances of the class Circle, c1, c2, and c3
Circle c1, c2, c3;  // They hold a special value called null
// Construct the instances via new operator
c1 = new Circle();
c2 = new Circle(2.0);
c3 = new Circle(3.0, "red");

// You can Declare and Construct in the same statement
Circle c4 = new Circle();
```

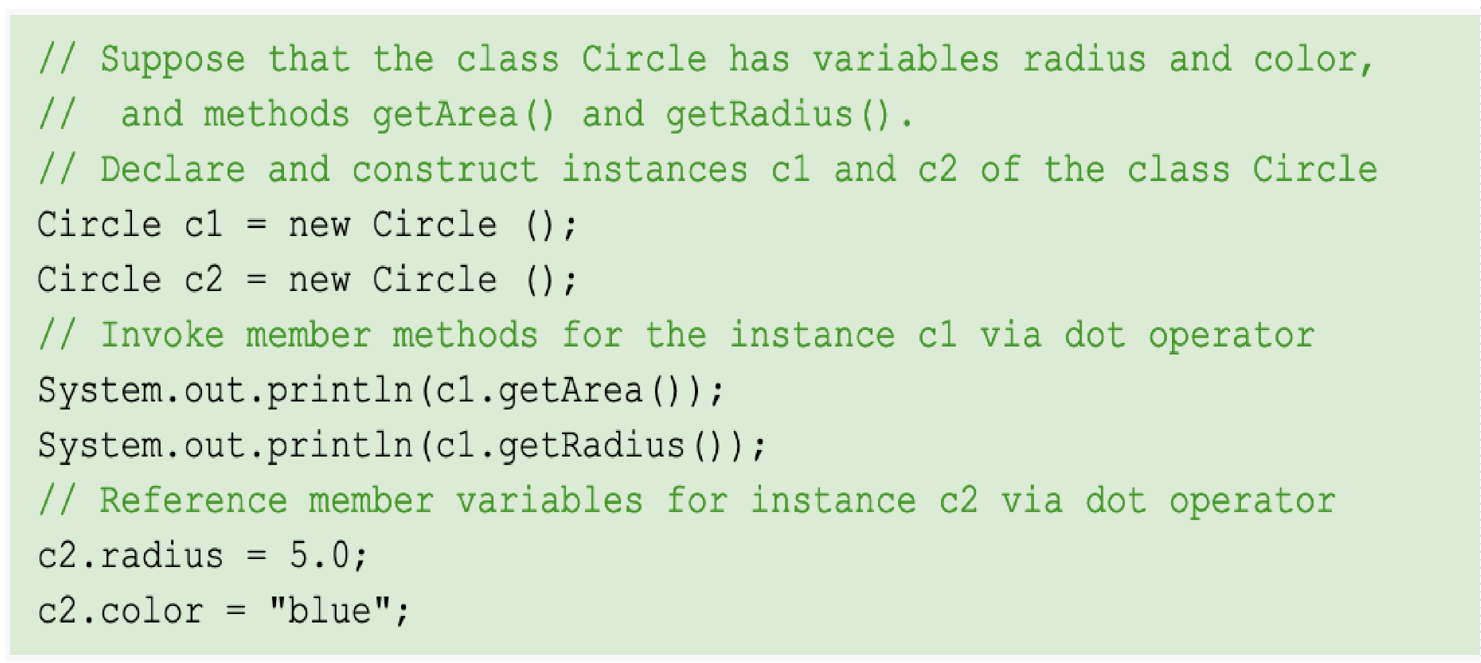
Note: When an instance is declared but not constructed, it holds a special value called null.

Dot (.) Operator

- The *variables* and *methods* belonging to a class are formally called *member variables* and *member methods*. To reference a member variable or method, you must:
  - ✓ First identify the instance you are interested in, and then,
  - ✓ Use the *dot operator* (.) to reference the desired member variable or method.

Dot (.) Operator (Cont.)

- For example,



```
// Suppose that the class Circle has variables radius and color,
//  and methods getArea() and getRadius().
// Declare and construct instances c1 and c2 of the class Circle
Circle c1 = new Circle ();
Circle c2 = new Circle ();
// Invoke member methods for the instance c1 via dot operator
System.out.println(c1.getArea());
System.out.println(c1.getRadius());
// Reference member variables for instance c2 via dot operator
c2.radius = 5.0;
c2.color = "blue";
```

Member Variables

- A *member variable* has a *name* (or *identifier*) and a *type*; and holds a *value* of that particular type.
- **Variable Naming Convention:** A variable name shall be a noun or a noun phrase made up of several words. The first word is in lowercase and the rest of the words are initial-capitalized (**camel-case**), e.g., fontSize, roomNumber, xMax, yMin and xTopLeft.
- The formal syntax for variable definition:

  [*AccessControlModifier*] *type variableName* [= *initialValue*]**;**

Member Variables (Cont.)

- For example,
  - ✓ private double radius;
  - ✓ public int length = 1, width = 1;

Member Methods

- A method:
    1. receives arguments from the caller,
    2. performs the operations defined in the method body, and
    3. returns a piece of result (or void) to the caller.
- The syntax for method declaration as follows:

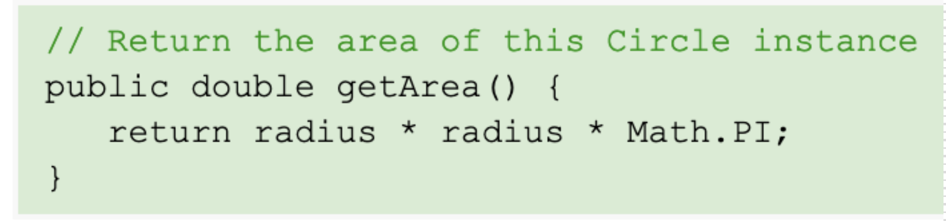
[*AccessControlModifier*] *returnType methodName* **(**[*parameterList*]**) {**

// method body or implementation ......

}

Member Methods (Cont.)

- For example,



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

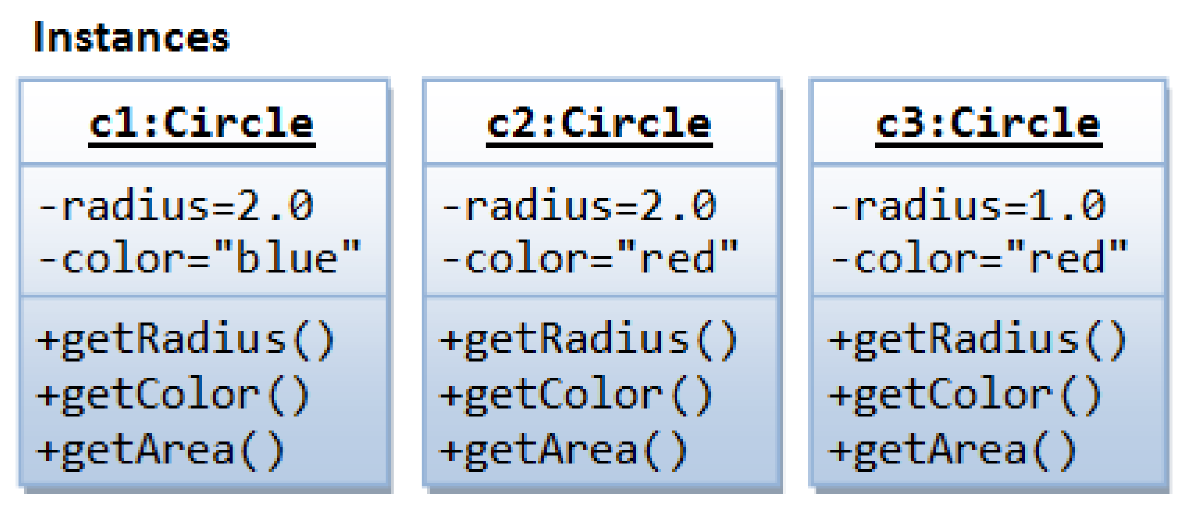
- **Method Naming Convention:**
  - ✓ A method name shall be a verb, or a verb phrase made up of several words.
  - ✓ The first word is in lowercase and the rest of the words are initial-capitalized (camel-case).
  - ✓ For example, getArea(), setRadius(), getParameterValues(), hasNext().

Variable name vs. Method name vs. Class name?

Variable name vs. Method name vs. Class name?

- A variable name is a noun, denoting an attribute;
- A method name is a verb, denoting an action.
- A class name shall be a noun or a noun phrase made up of several words, denoting an object.

An OOP Example



Instances

c1:Circle

-radius=2.0
-color="blue"

+getRadius()
+getColor()
+getArea()

c2:Circle

-radius=2.0
-color="red"

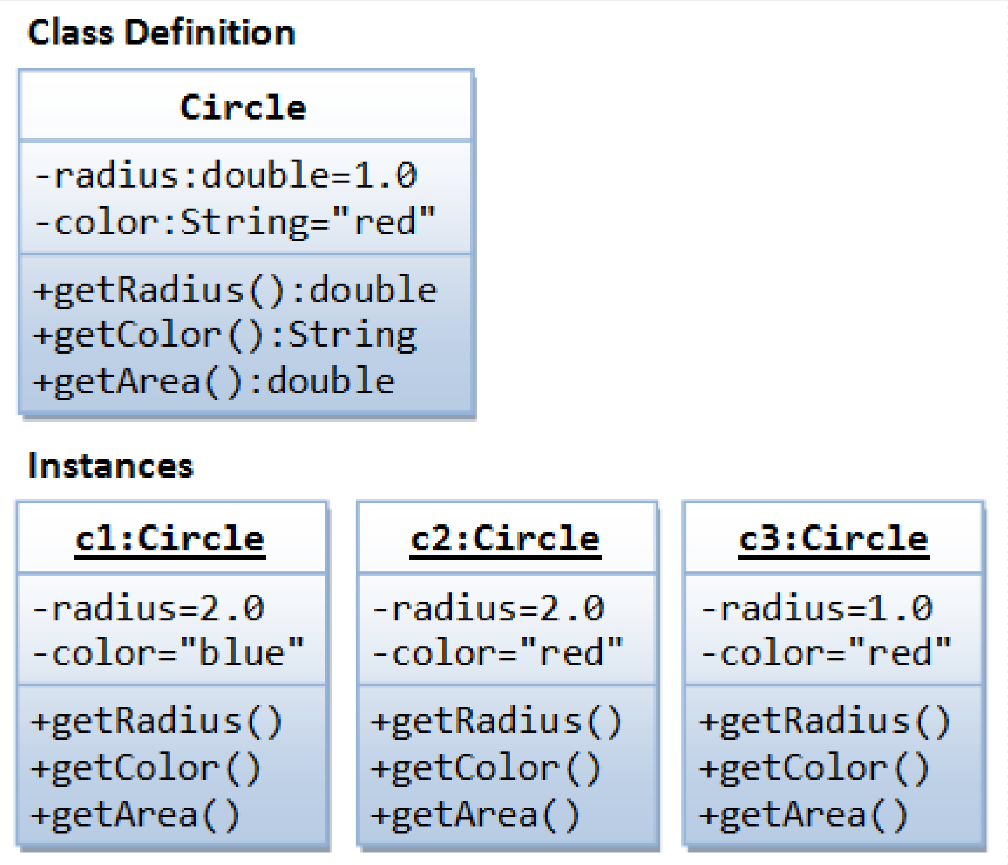
+getRadius()
+getColor()
+getArea()

c3:Circle

-radius=1.0
-color="red"

+getRadius()
+getColor()
+getArea()

An OOP Example (Cont.)



Class Definition

Circle

-radius:double=1.0
-color:String="red"

+getRadius():double
+getColor():String
+getArea():double

Instances

c1:Circle

-radius=2.0
-color="blue"

+getRadius()
+getColor()
+getArea()

c2:Circle

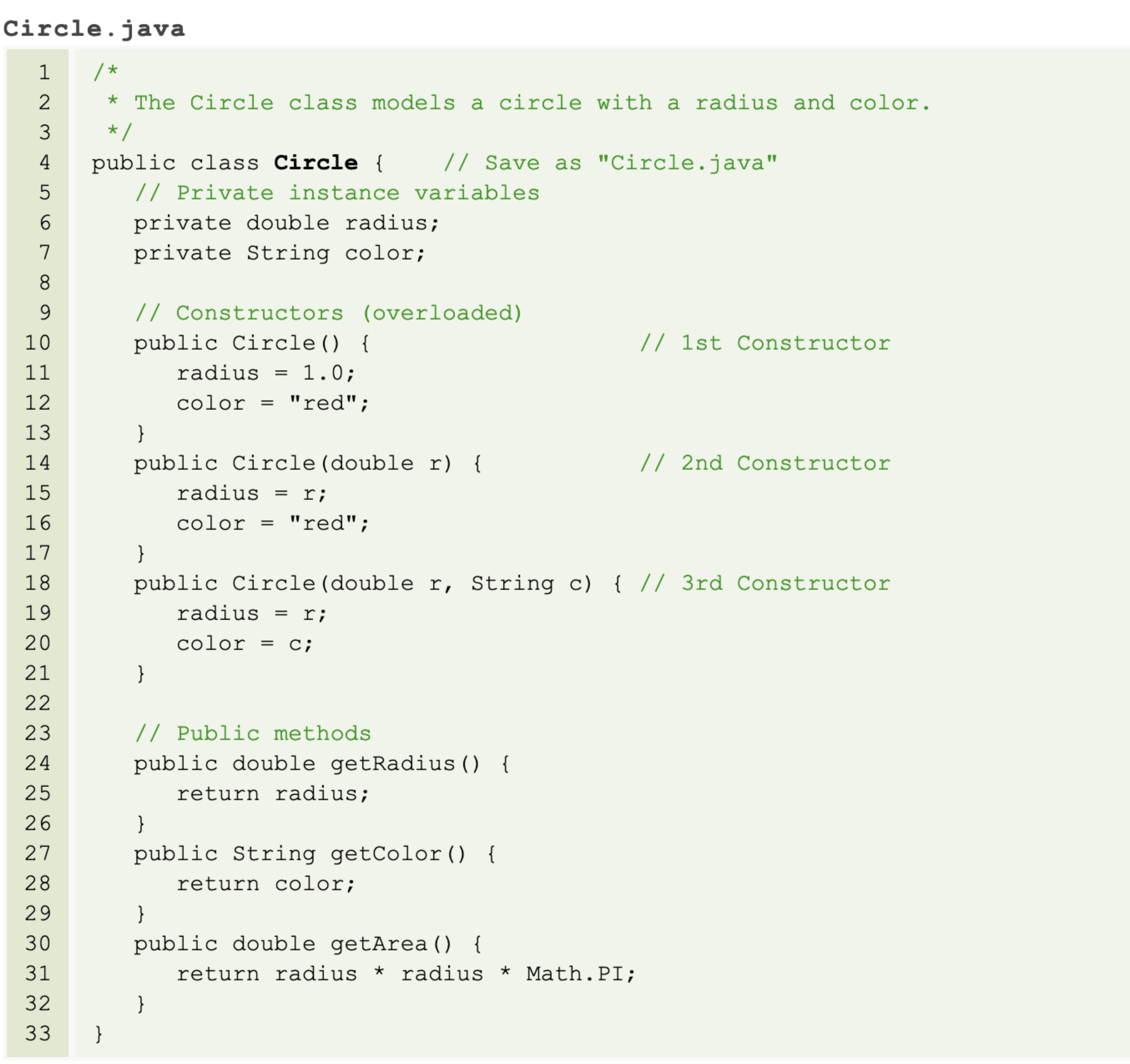
-radius=2.0
-color="red"

+getRadius()
+getColor()
+getArea()

c3:Circle

-radius=1.0
-color="red"

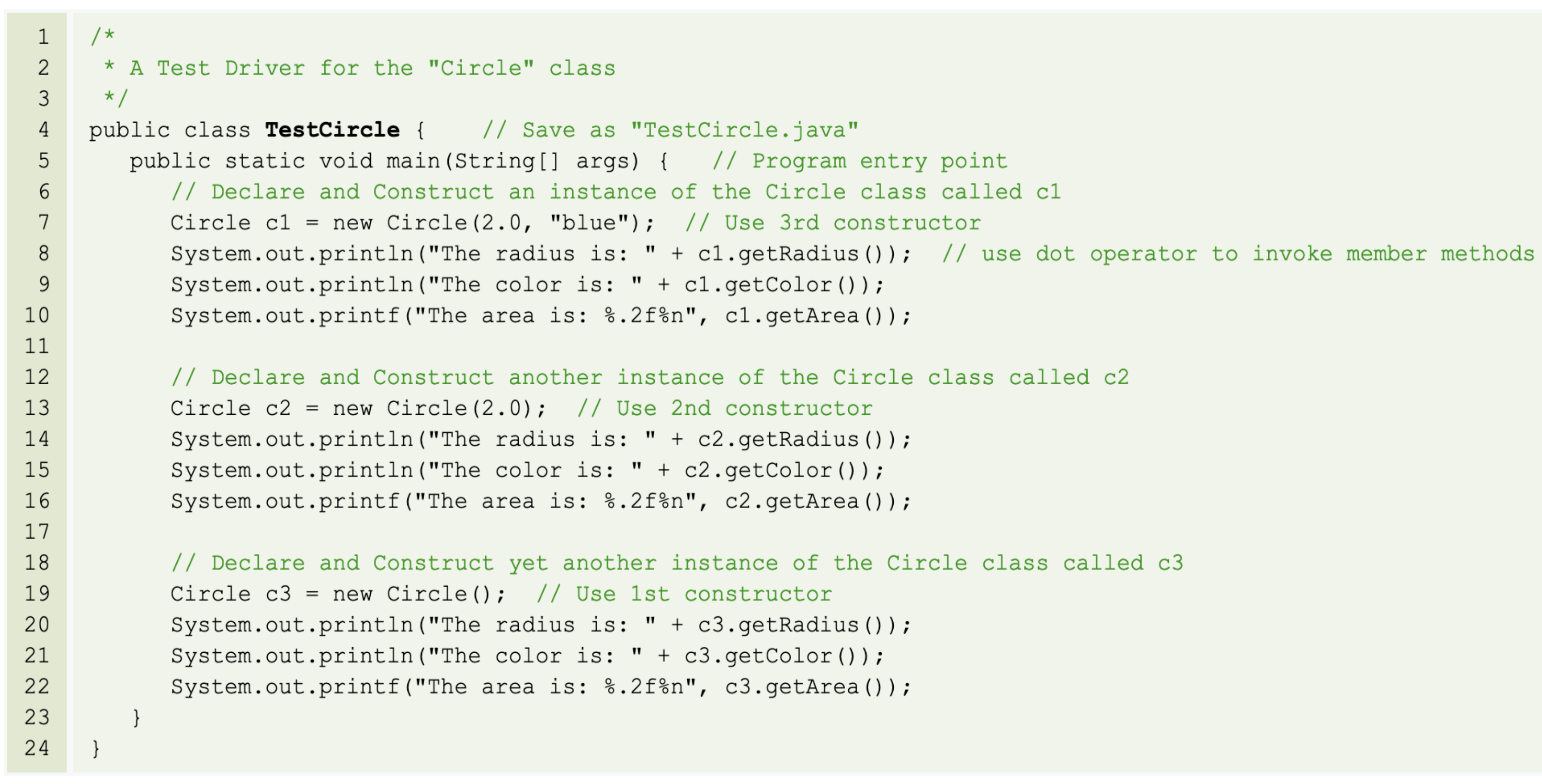
+getRadius()
+getColor()
+getArea()



Circle.java

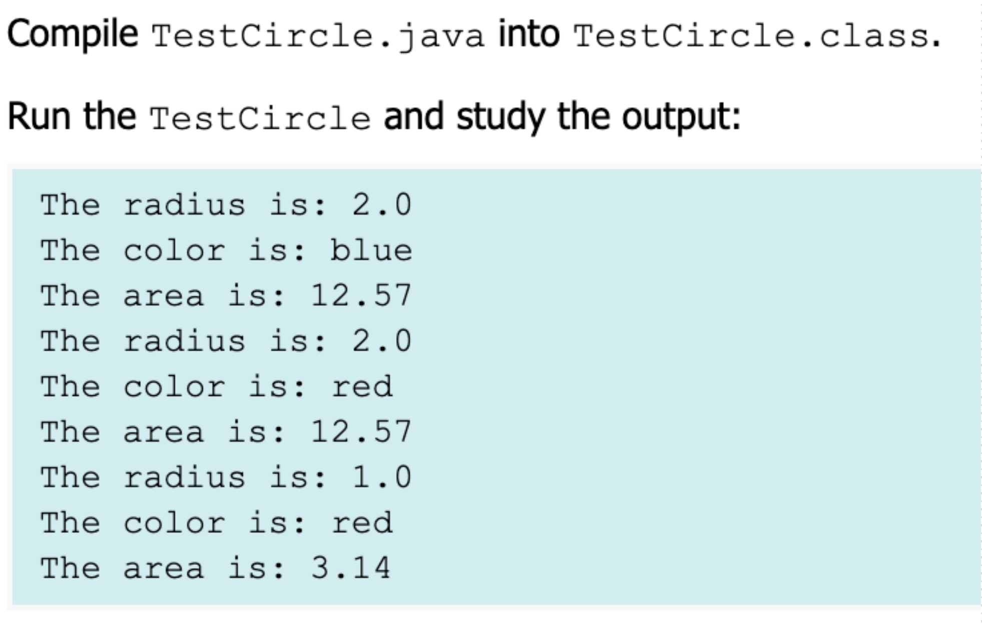
```
/*
 * The Circle class models a circle with a radius and color.
 */
public class Circle {    // Save as "Circle.java"
   // Private instance variables
   private double radius;
   private String color;

   // Constructors (overloaded)
   public Circle() {                  // 1st Constructor
      radius = 1.0;
      color = "red";
   }
   public Circle(double r) {          // 2nd Constructor
      radius = r;
      color = "red";
   }
   public Circle(double r, String c) { // 3rd Constructor
      radius = r;
      color = c;
   }

   // Public methods
   public double getRadius() {
      return radius;
   }
   public String getColor() {
      return color;
   }
   public double getArea() {
      return radius * radius * Math.PI;
   }
}
```

An OOP Example (Cont.)



```
/*
 * A Test Driver for the "Circle" class
 */
public class TestCircle {    // Save as "TestCircle.java"
   public static void main(String[] args) {   // Program entry point
      // Declare and Construct an instance of the Circle class called c1
      Circle c1 = new Circle(2.0, "blue");  // Use 3rd constructor
      System.out.println("The radius is: " + c1.getRadius());  // use dot operator to invoke member methods
      System.out.println("The color is: " + c1.getColor());
      System.out.printf("The area is: %.2f%n", c1.getArea());

      // Declare and Construct another instance of the Circle class called c2
      Circle c2 = new Circle(2.0);  // Use 2nd constructor
      System.out.println("The radius is: " + c2.getRadius());
      System.out.println("The color is: " + c2.getColor());
      System.out.printf("The area is: %.2f%n", c2.getArea());

      // Declare and Construct yet another instance of the Circle class called c3
      Circle c3 = new Circle();  // Use 1st constructor
      System.out.println("The radius is: " + c3.getRadius());
      System.out.println("The color is: " + c3.getColor());
      System.out.printf("The area is: %.2f%n", c3.getArea());
   }
}
```

An OOP Example (Cont.)



Compile `TestCircle.java` into `TestCircle.class`.

Run the `TestCircle` and study the output:

```
The radius is: 2.0
The color is: blue
The area is: 12.57
The radius is: 2.0
The color is: red
The area is: 12.57
The radius is: 1.0
The color is: red
The area is: 3.14
```

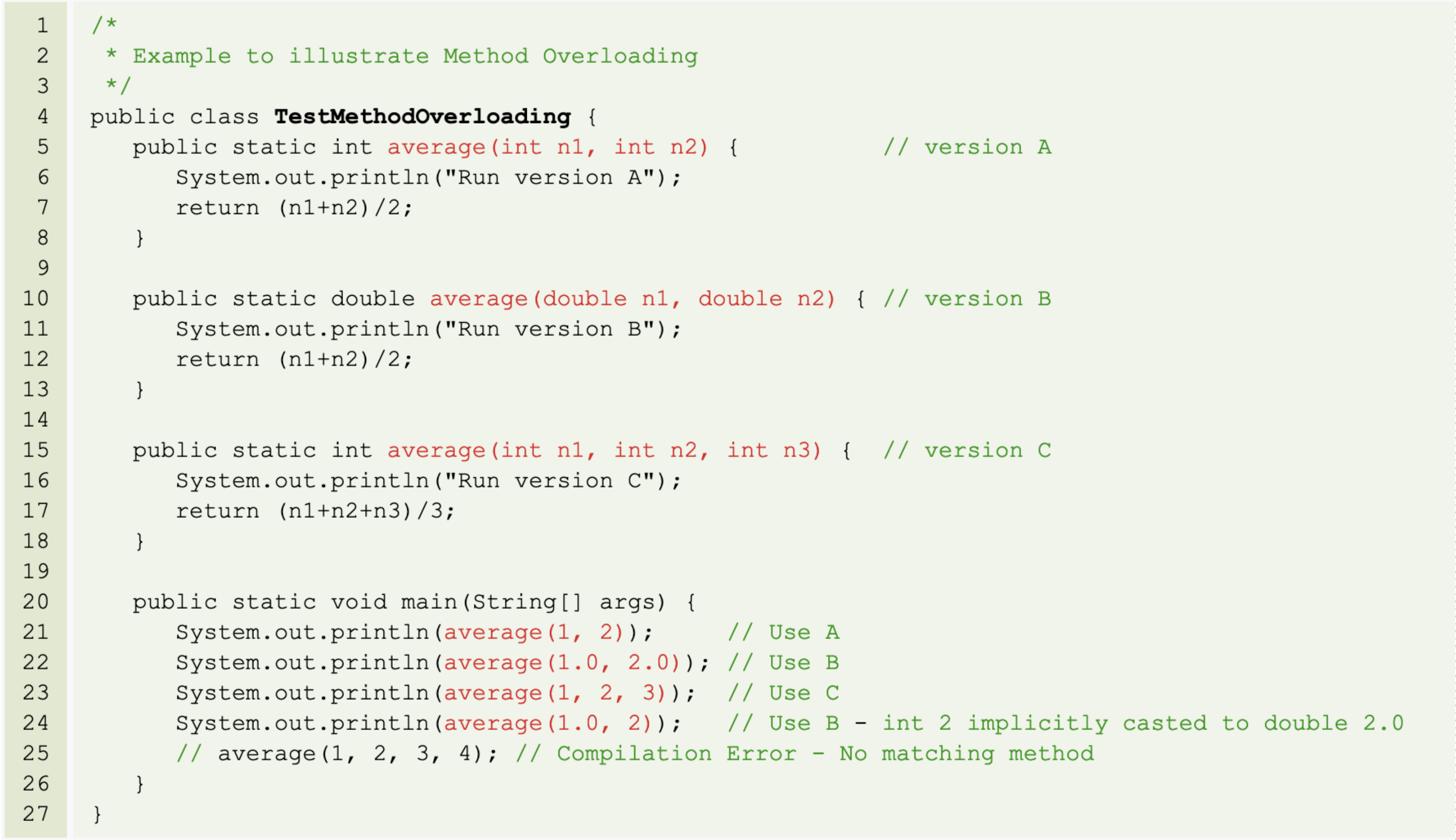
Constructor

- For example,
  - ✓ Circle c1 = new Circle();
  - ✓ Circle c2 = new Circle(2.0);
  - ✓ Circle c3 = new Circle(3.0, "red");
- **Note:**
  - ✓ The name of the constructor method is the same as the class name.
  - ✓ Constructor has no return type.
  - ✓ Constructor can only be invoked via the "new" operator.
  - ✓ Constructors are not inherited (next day).
- **Default Constructor**: A constructor with no parameter is called the *default constructor*.

Method Overloading

- Method overloading means that the *same method name* can have *different implementations* (versions). However, the different implementations must be distinguishable by their parameter list (either the number of parameters, or the type of parameters, or their order).

Method Overloading (Cont.)



```java
/*
 * Example to illustrate Method Overloading
 */
public class TestMethodOverloading {
   public static int average(int n1, int n2) {          // version A
      System.out.println("Run version A");
      return (n1+n2)/2;
   }

   public static double average(double n1, double n2) { // version B
      System.out.println("Run version B");
      return (n1+n2)/2;
   }

   public static int average(int n1, int n2, int n3) {  // version C
      System.out.println("Run version C");
      return (n1+n2+n3)/3;
   }

   public static void main(String[] args) {
      System.out.println(average(1, 2));     // Use A
      System.out.println(average(1.0, 2.0)); // Use B
      System.out.println(average(1, 2, 3));  // Use C
      System.out.println(average(1.0, 2));   // Use B - int 2 implicitly casted to double 2.0
      // average(1, 2, 3, 4); // Compilation Error - No matching method
   }
}
```

Overloading Circle Class' Constructor

- Circle()
- Circle(double r)
- Circle(double r, String c)

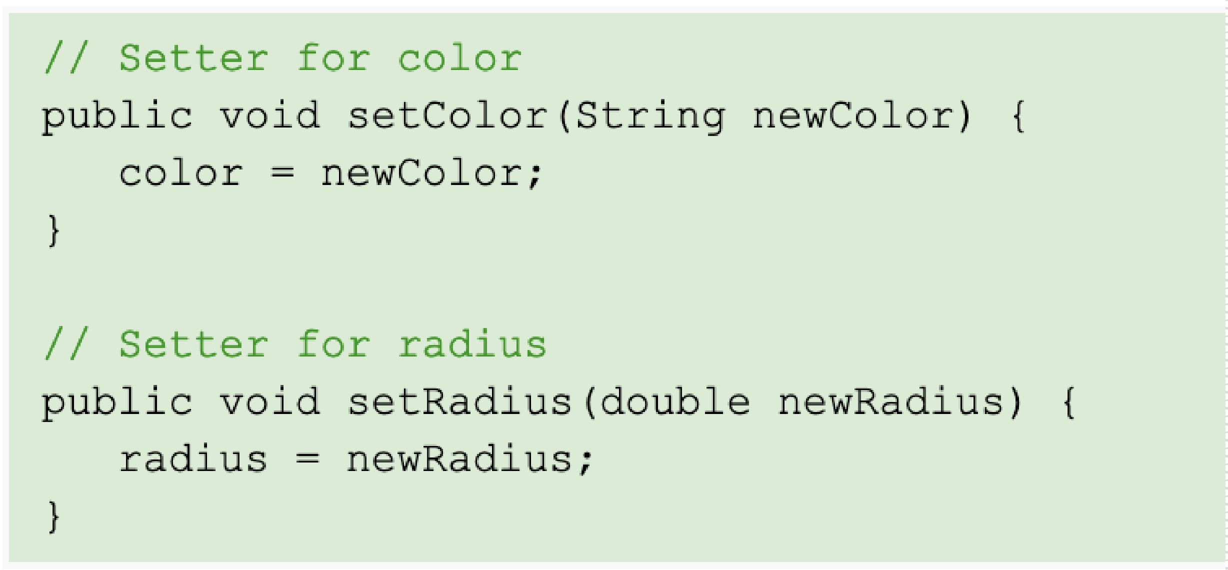
Access Control Modifiers : public vs private

- ***public***: The class/variable/method is accessible and available to *all* the ***other objects*** in the system.
- ***private***: The class/variable/method is accessible and available *within* ***this class only***.

Information Hiding and Encapsulation

- Member variables of a class are typically hidden from the outside word (i.e., the other classes), with private access control modifier.
- Access to the member variables are provided via public assessor methods, e.g., getRadius() and getColor().
- **Hiding?**
- **Encapsulation?**
- **Rule:** Do not make any variable public, unless you have a good reason.

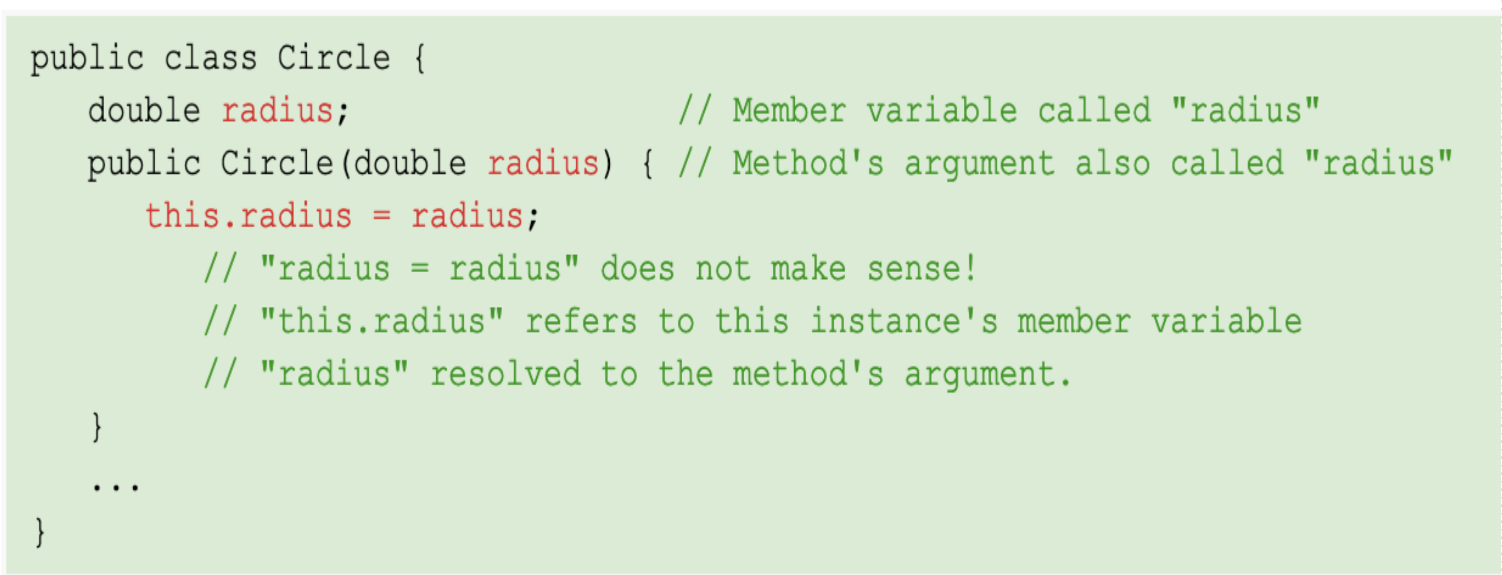
The public Getters and Setters for private Variables



```java
// Setter for color
public void setColor(String newColor) {
   color = newColor;
}

// Setter for radius
public void setRadius(double newRadius) {
   radius = newRadius;
}
```

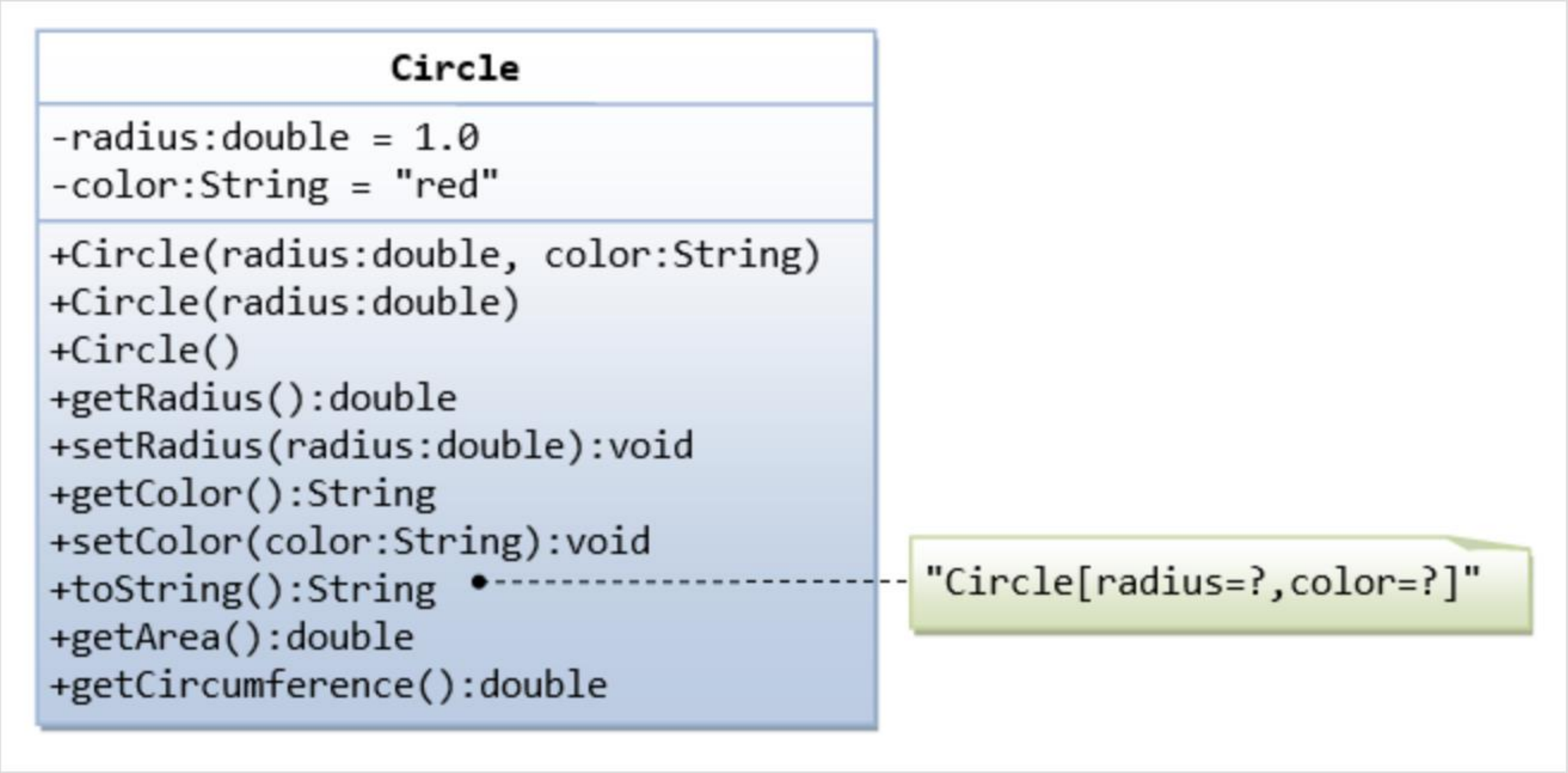
Keyword "this"

- To refer to *this* instance inside a class definition.



```
public class Circle {
   double radius;                     // Member variable called "radius"
   public Circle(double radius) { // Method's argument also called "radius"
      this.radius = radius;
         // "radius = radius" does not make sense!
         // "this.radius" refers to this instance's member variable
         // "radius" resolved to the method's argument.
   }
   ...
}
```

Revised Circle Class



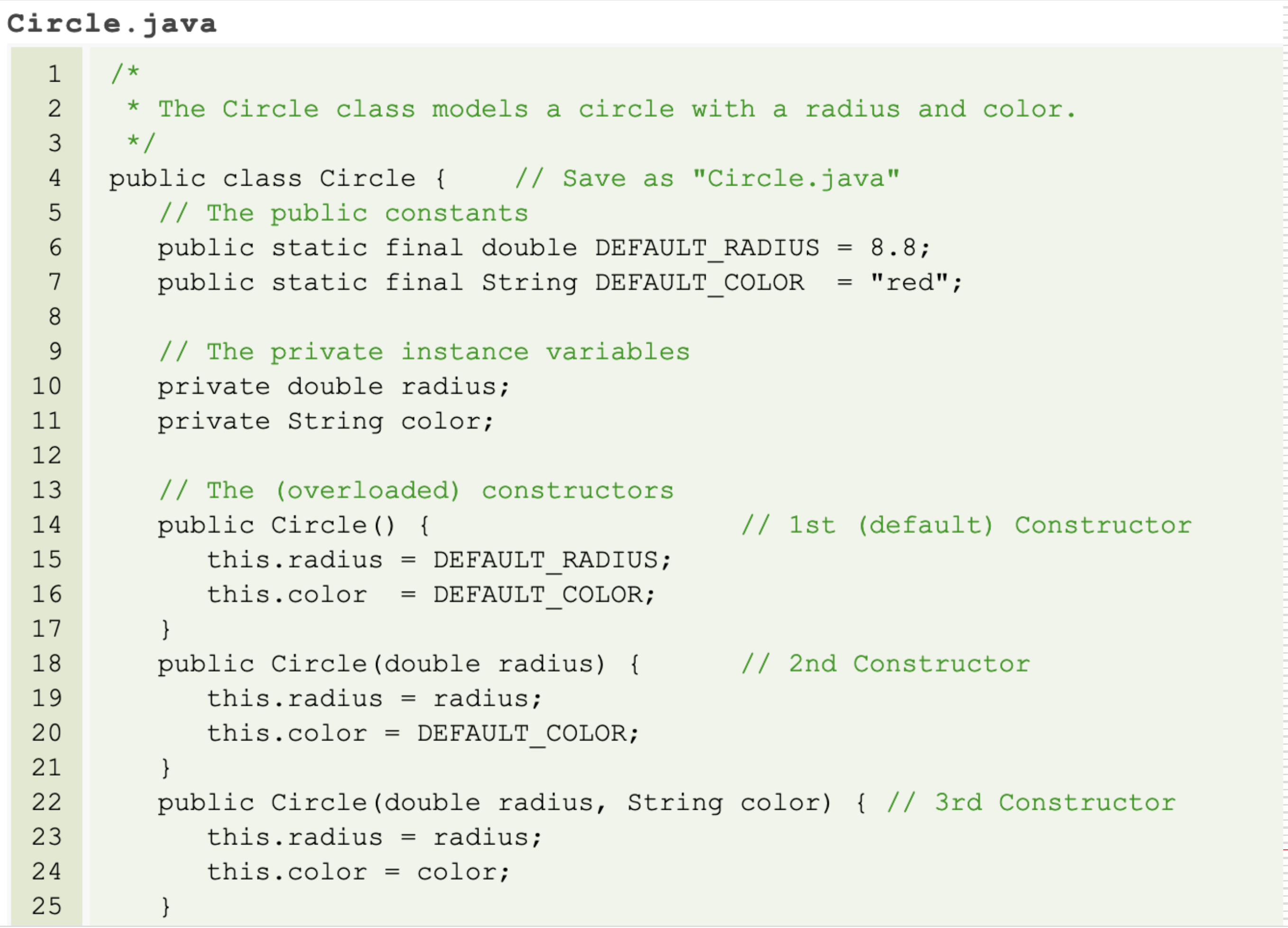
Circle

-radius:double = 1.0
-color:String = "red"

+Circle(radius:double, color:String)
+Circle(radius:double)
+Circle()
+getRadius():double
+setRadius(radius:double):void
+getColor():String
+setColor(color:String):void
+toString():String
+getArea():double
+getCircumference():double

"Circle[radius=?,color=?]"

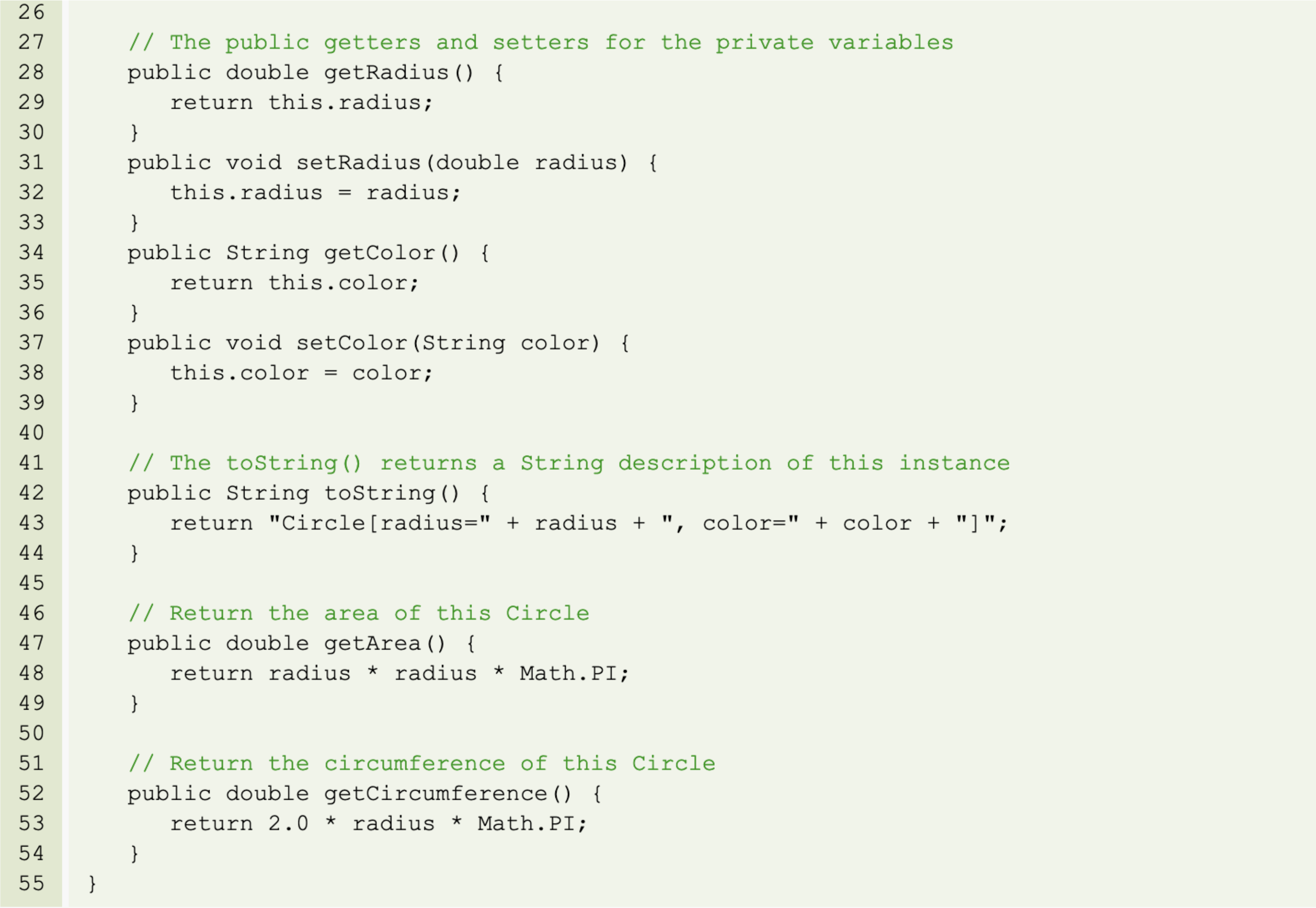
Revised Circle Class (Cont.)



Circle.java

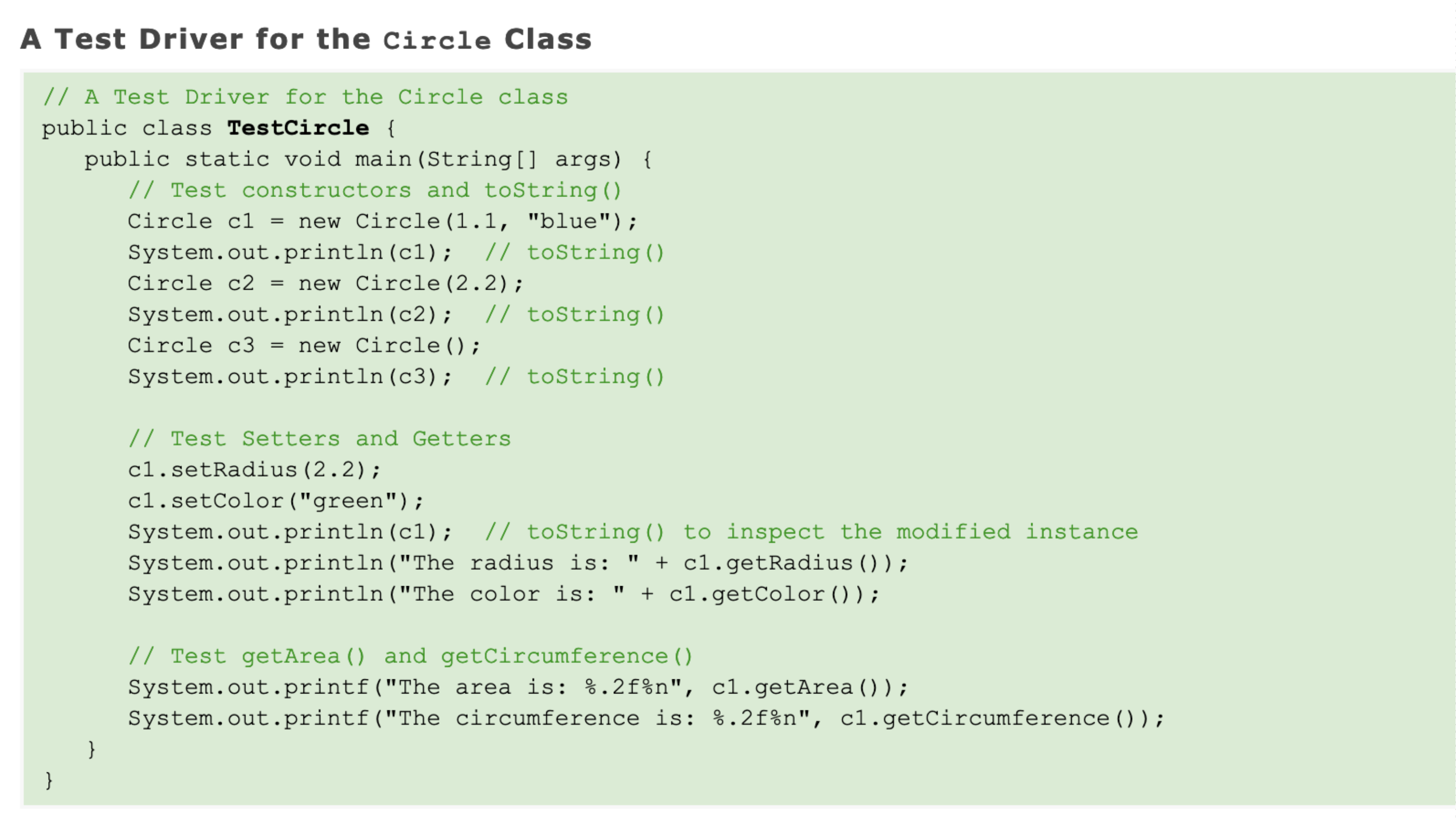
```java
/*
 * The Circle class models a circle with a radius and color.
 */
public class Circle {    // Save as "Circle.java"
   // The public constants
   public static final double DEFAULT_RADIUS = 8.8;
   public static final String DEFAULT_COLOR  = "red";

   // The private instance variables
   private double radius;
   private String color;

   // The (overloaded) constructors
   public Circle() {                   // 1st (default) Constructor
      this.radius = DEFAULT_RADIUS;
      this.color  = DEFAULT_COLOR;
   }
   public Circle(double radius) {      // 2nd Constructor
      this.radius = radius;
      this.color = DEFAULT_COLOR;
   }
   public Circle(double radius, String color) { // 3rd Constructor
      this.radius = radius;
      this.color = color;
   }
```

Revised Circle Class (Cont.)



```java

   // The public getters and setters for the private variables
   public double getRadius() {
      return this.radius;
   }
   public void setRadius(double radius) {
      this.radius = radius;
   }
   public String getColor() {
      return this.color;
   }
   public void setColor(String color) {
      this.color = color;
   }

   // The toString() returns a String description of this instance
   public String toString() {
      return "Circle[radius=" + radius + ", color=" + color + "]";
   }

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

   // Return the circumference of this Circle
   public double getCircumference() {
      return 2.0 * radius * Math.PI;
   }
}
```

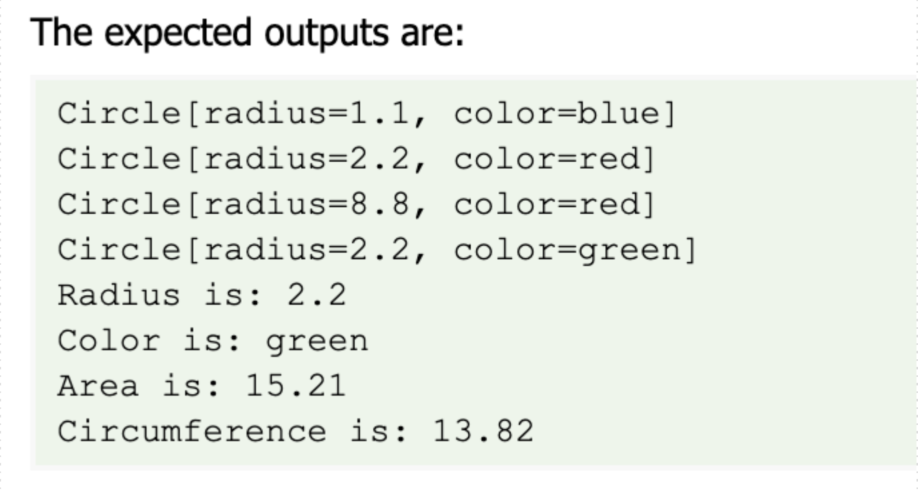
Revised Circle Class (Cont.)



A Test Driver for the `Circle` Class

```java
// A Test Driver for the Circle class
public class TestCircle {
   public static void main(String[] args) {
      // Test constructors and toString()
      Circle c1 = new Circle(1.1, "blue");
      System.out.println(c1);  // toString()
      Circle c2 = new Circle(2.2);
      System.out.println(c2);  // toString()
      Circle c3 = new Circle();
      System.out.println(c3);  // toString()

      // Test Setters and Getters
      c1.setRadius(2.2);
      c1.setColor("green");
      System.out.println(c1);  // toString() to inspect the modified instance
      System.out.println("The radius is: " + c1.getRadius());
      System.out.println("The color is: " + c1.getColor());

      // Test getArea() and getCircumference()
      System.out.printf("The area is: %.2f%n", c1.getArea());
      System.out.printf("The circumference is: %.2f%n", c1.getCircumference());
   }
}
```

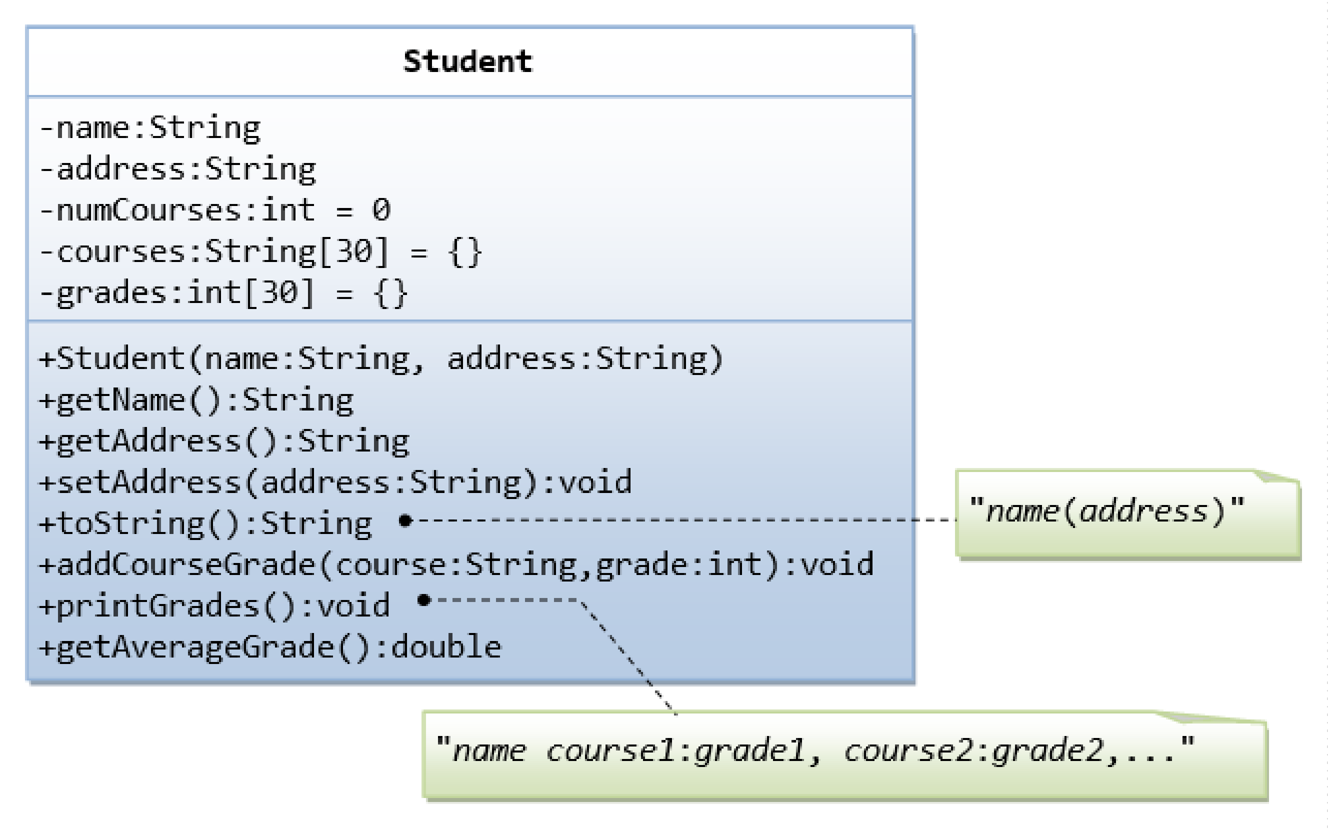
Revised Circle Class (Cont.)



The expected outputs are:

```
Circle[radius=1.1, color=blue]
Circle[radius=2.2, color=red]
Circle[radius=8.8, color=red]
Circle[radius=2.2, color=green]
Radius is: 2.2
Color is: green
Area is: 15.21
Circumference is: 13.82
```

The Student Class

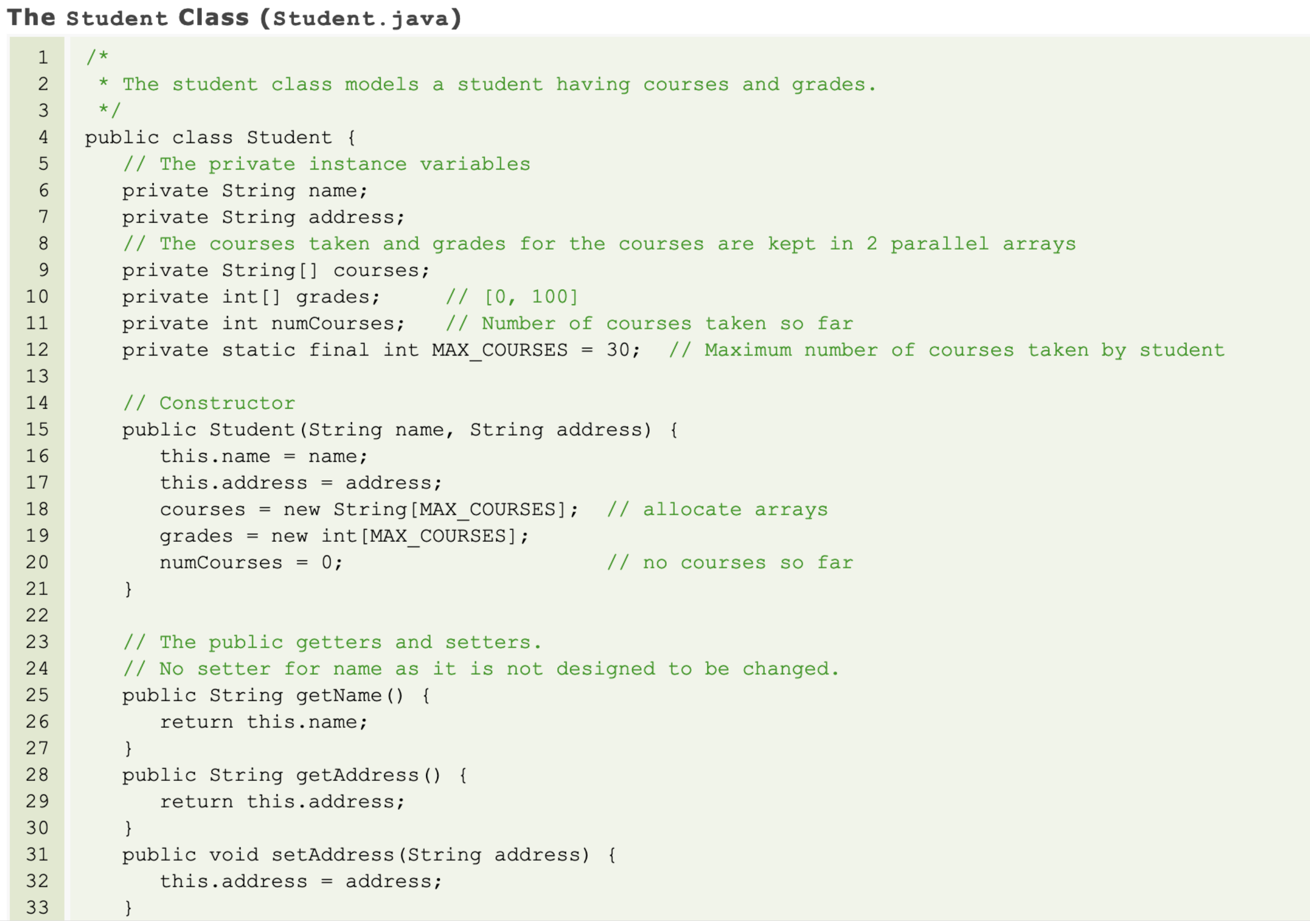


```
Student

-name:String
-address:String
-numCourses:int = 0
-courses:String[30] = {}
-grades:int[30] = {}

+Student(name:String, address:String)
+getName():String
+getAddress():String
+setAddress(address:String):void
+toString():String
+addCourseGrade(course:String,grade:int):void
+printGrades():void
+getAverageGrade():double
```

"name(address)"

"name course1:grade1, course2:grade2,..."

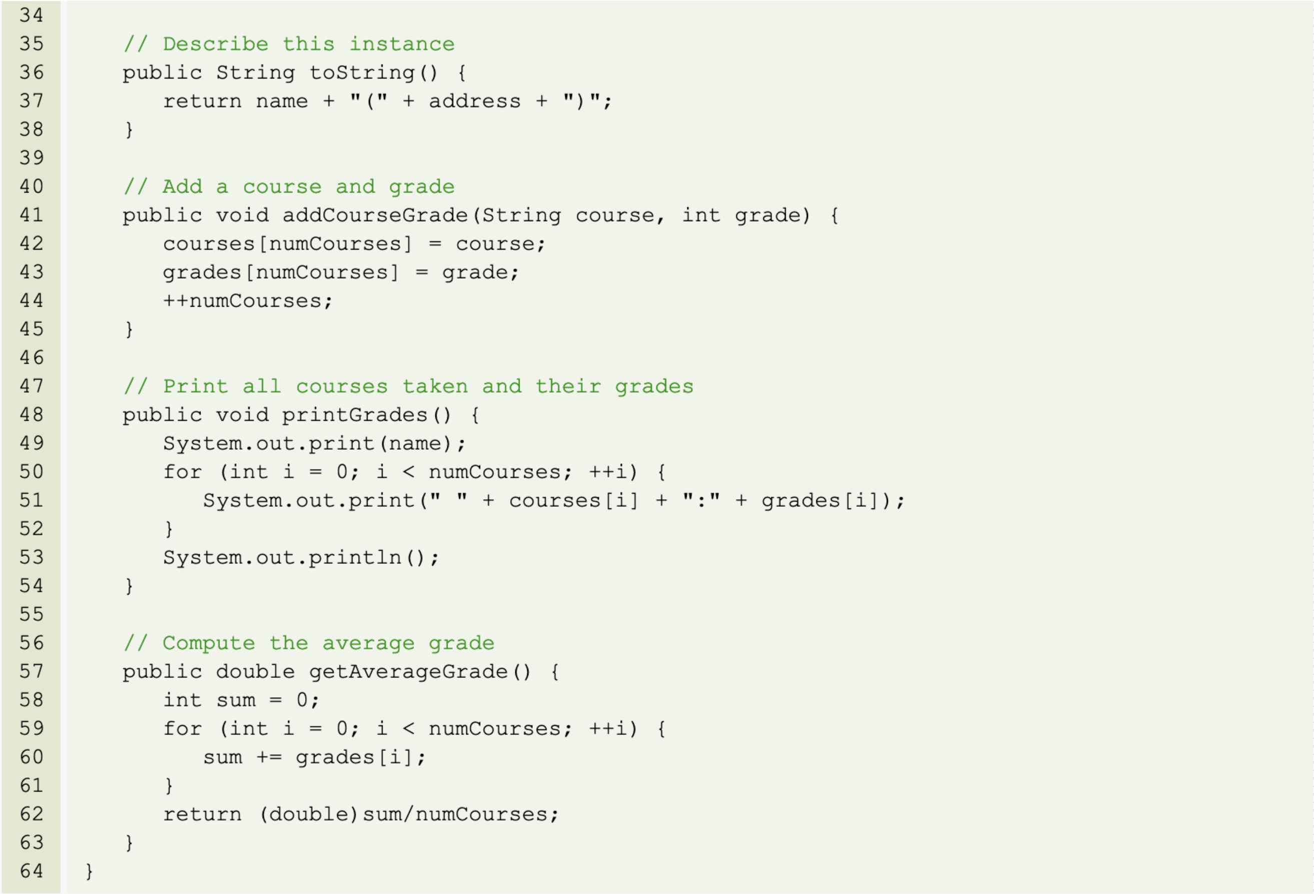
The Student Class (Cont.)



The `Student` Class (`Student.java`)

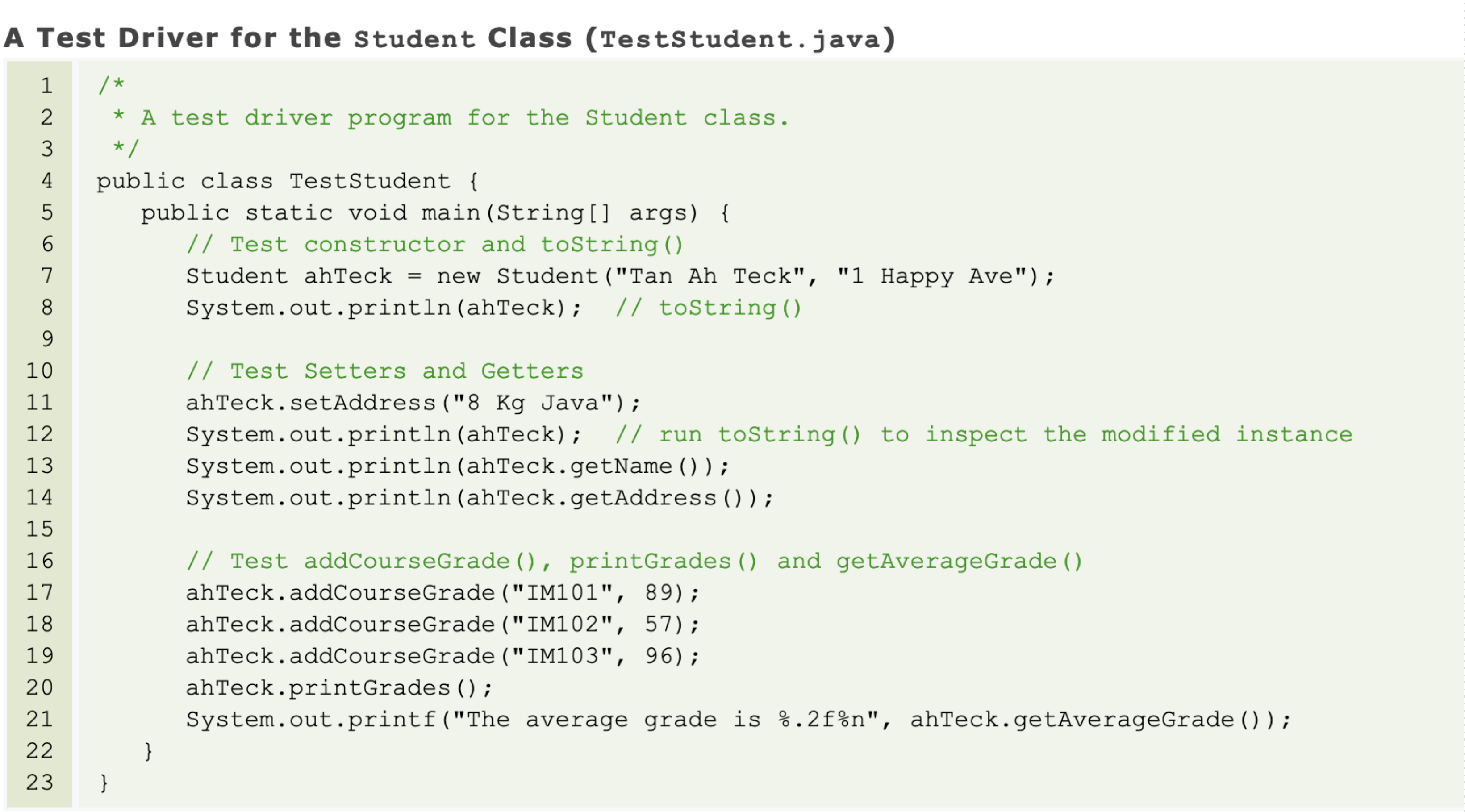
```java
/*
 * The student class models a student having courses and grades.
 */
public class Student {
   // The private instance variables
   private String name;
   private String address;
   // The courses taken and grades for the courses are kept in 2 parallel arrays
   private String[] courses;
   private int[] grades;     // [0, 100]
   private int numCourses;   // Number of courses taken so far
   private static final int MAX_COURSES = 30;  // Maximum number of courses taken by student

   // Constructor
   public Student(String name, String address) {
      this.name = name;
      this.address = address;
      courses = new String[MAX_COURSES];  // allocate arrays
      grades = new int[MAX_COURSES];
      numCourses = 0;                     // no courses so far
   }

   // The public getters and setters.
   // No setter for name as it is not designed to be changed.
   public String getName() {
      return this.name;
   }
   public String getAddress() {
      return this.address;
   }
   public void setAddress(String address) {
      this.address = address;
   }
```

The Student Class (Cont.)



```java

   // Describe this instance
   public String toString() {
      return name + "(" + address + ")";
   }

   // Add a course and grade
   public void addCourseGrade(String course, int grade) {
      courses[numCourses] = course;
      grades[numCourses] = grade;
      ++numCourses;
   }

   // Print all courses taken and their grades
   public void printGrades() {
      System.out.print(name);
      for (int i = 0; i < numCourses; ++i) {
         System.out.print(" " + courses[i] + ":" + grades[i]);
      }
      System.out.println();
   }

   // Compute the average grade
   public double getAverageGrade() {
      int sum = 0;
      for (int i = 0; i < numCourses; ++i) {
         sum += grades[i];
      }
      return (double)sum/numCourses;
   }
}
```

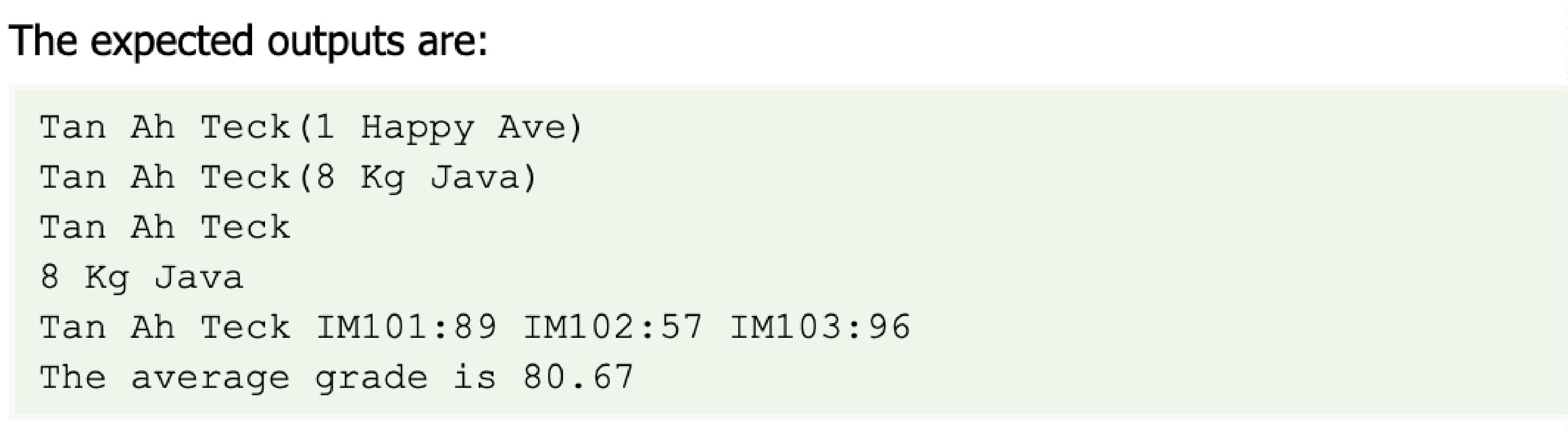
The Student Class (Cont.)



A Test Driver for the Student Class (TestStudent.java)

```java
/*
 * A test driver program for the Student class.
 */
public class TestStudent {
   public static void main(String[] args) {
      // Test constructor and toString()
      Student ahTeck = new Student("Tan Ah Teck", "1 Happy Ave");
      System.out.println(ahTeck);  // toString()

      // Test Setters and Getters
      ahTeck.setAddress("8 Kg Java");
      System.out.println(ahTeck);  // run toString() to inspect the modified instance
      System.out.println(ahTeck.getName());
      System.out.println(ahTeck.getAddress());

      // Test addCourseGrade(), printGrades() and getAverageGrade()
      ahTeck.addCourseGrade("IM101", 89);
      ahTeck.addCourseGrade("IM102", 57);
      ahTeck.addCourseGrade("IM103", 96);
      ahTeck.printGrades();
      System.out.printf("The average grade is %.2f%n", ahTeck.getAverageGrade());
   }
}
```

The Student Class (Cont.)



The expected outputs are:

```
Tan Ah Teck(1 Happy Ave)
Tan Ah Teck(8 Kg Java)
Tan Ah Teck
8 Kg Java
Tan Ah Teck IM101:89 IM102:57 IM103:96
The average grade is 80.67
```

Q&A

Thank you!