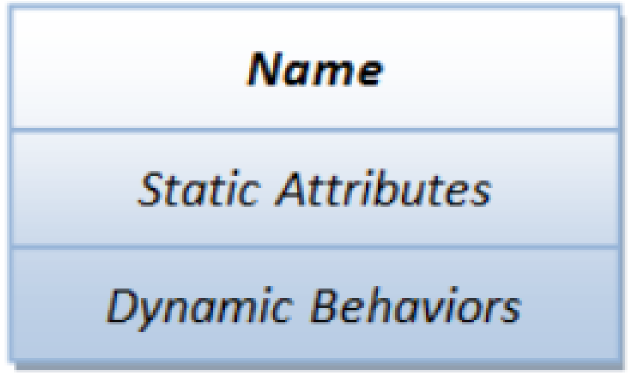
OOP

RECAP da settembre ad oggi

CLASSE



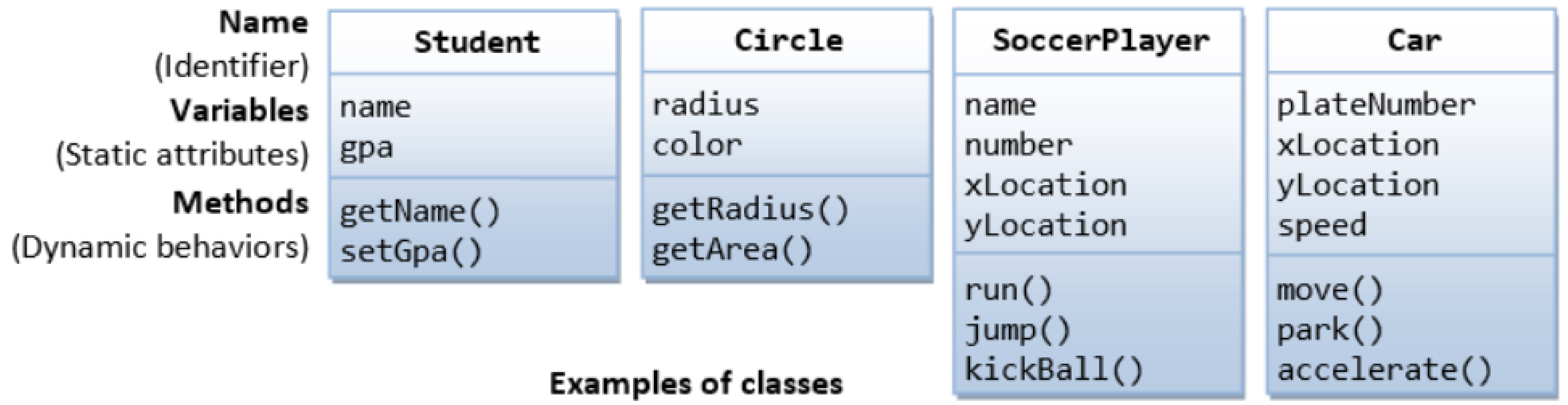
A class is a 3-compartment box

A class can be visualized as a three-compartment box, as illustrated:

1. *Name* (or identity): identifies the class.
2. *Variables* (or attribute, state, field): contains the *static attributes* of the class.
3. *Methods* (or behaviors, function, operation): contains the *dynamic behaviors* of the class.

In other words, a class encapsulates the static attributes (data) and dynamic behaviors (operations that operate on the data) in a box.

ISTANZA

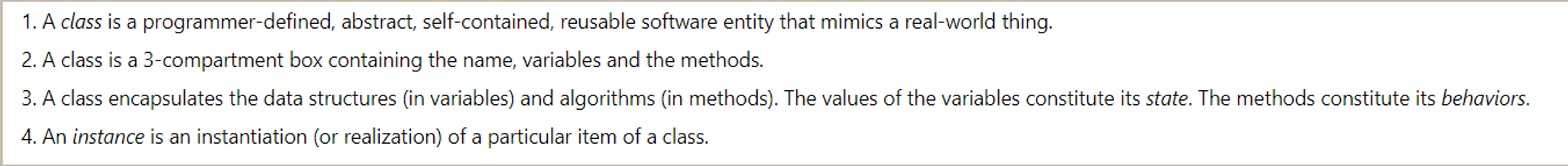


	Student	Circle	SoccerPlayer	Car
Name (Identifier)	Student	Circle	SoccerPlayer	Car
Variables (Static attributes)	name gpa	radius color	name number xLocation yLocation	plateNumber xLocation yLocation speed

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

Examples of classes

Summary



1. A *class* is a programmer-defined, abstract, self-contained, reusable software entity that mimics a real-world thing.
2. A class is a 3-compartment box containing the name, variables and the methods.
3. A class encapsulates the data structures (in variables) and algorithms (in methods). The values of the variables constitute its *state*. The methods constitute its *behaviors*.
4. An *instance* is an instantiation (or realization) of a particular item of a class.

Class Definition in Java

The syntax for class definition in Java is:

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

Examples

```
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:

1. Declare an instance identifier (instance name) of a particular class.
2. Construct the instance (i.e., allocate storage for the instance and initialize the instance) using the "new" operator.

For examples, suppose that we have a class called Circle, we can create instances of Circle as follows:

```
// 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();
```

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:

1. First identify the instance you are interested in, and then,
2. Use the dot operator (.) to reference the desired member variable or method.
3.

For example, suppose that we have a class called Circle, with two member variables (radius and color) and two member methods (getRadius() and getArea()).
We have created three instances of the class Circle, namely, c1, c2 and c3.
To invoke the method getArea(), you must first identity the instance of interest, say c2, then use the dot operator, in the form of c2.getArea().

Dot (.) Operator

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";
```

Dot (.) Operator

Calling getArea() without identifying the instance is meaningless, as the radius is unknown (there could be many instances of Circle - each maintaining its own radius). Furthermore, c1.getArea() and c2.getArea() are likely to produce different results.

In general, suppose there is a class called AClass with a member variable called aVariable and a member method called aMethod(). An instance called anInstance is constructed for AClass. You use anInstance.aVariable and anInstance.aMethod().

Member Variables

A member variable has a name (or identifier) and a type; and holds a value of that particular type (as descried in the earlier chapter).

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 in Java is:

[AccessControlModifier] type variableName [= initialValue];

[AccessControlModifier] type variableName-1 [= initialValue-1] [, type variableName-2 [= initialValue-2]] ... ;

For example,

```
private double radius;
public int length = 1, width = 1;
```

Member Methods

A method (as described in the earlier chapter):

receives arguments from the caller,
performs the operations defined in the method body, and
returns a piece of result (or void) to the caller.

The syntax for method declaration in Java is as follows:

```
[AccessControlModifier] returnType methodName
([parameterList]) {
   // method body or implementation
   ......
}
```

Variable name vs. Method name vs. Class name

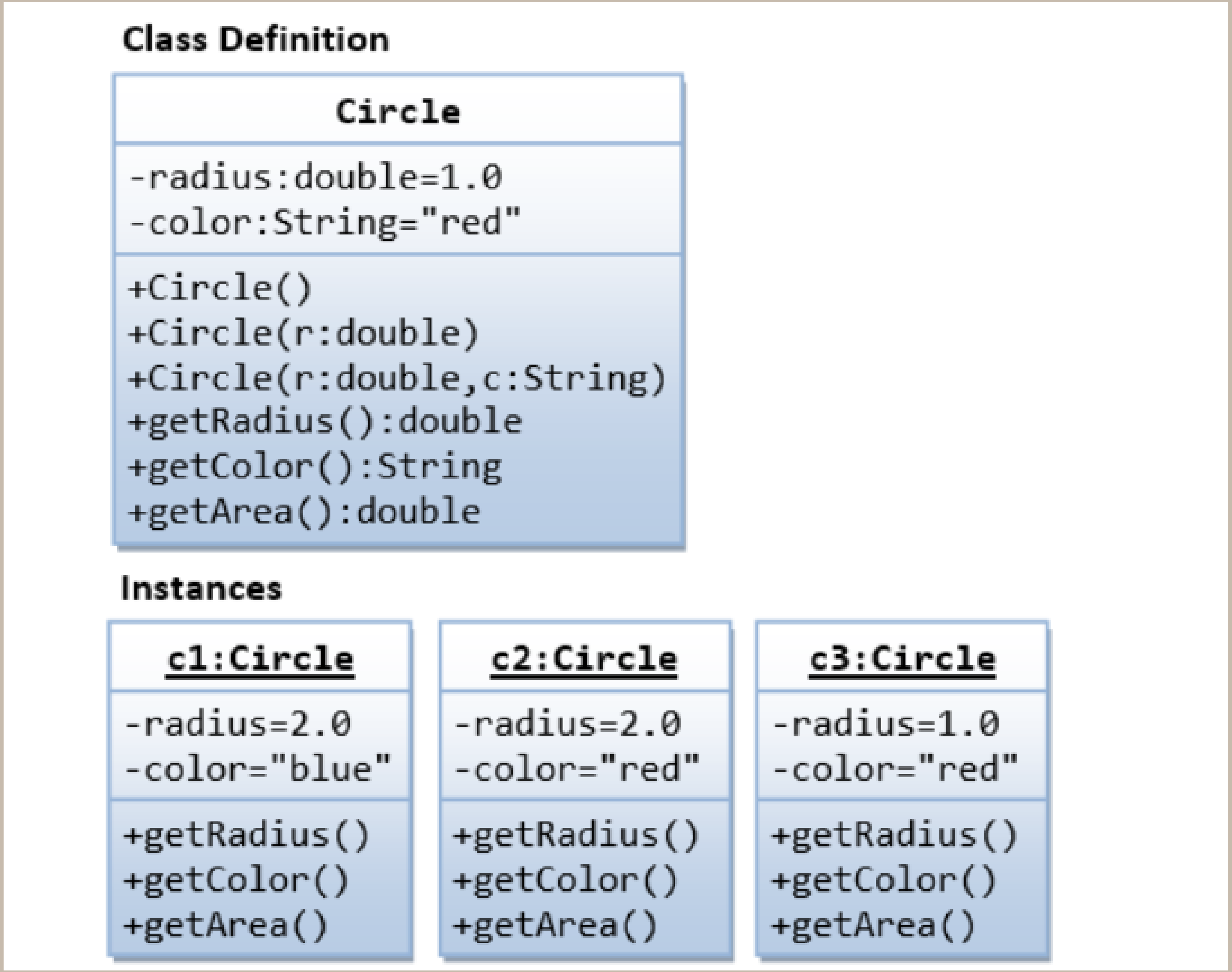
A variable name is a noun, denoting an attribute; while a method name is a verb, denoting an action.

They have the same naming convention (the first word in lowercase and the rest are initial-capitalized). Nevertheless, you can easily distinguish them from the context.

Methods take arguments in parentheses (possibly zero arguments with empty parentheses), but variables do not. In this writing, methods are denoted with a pair of parentheses, e.g., println(), getArea() for clarity.

On the other hand, class name is a noun beginning with uppercase.

An OOP Example



An OOP Example

```
// The Circle class models a circle with a radius and color.

public class Circle {
   // Private instance variables
   private double radius;
   private String color;

   // Constructors (overloaded)
   /** Constructs a Circle instance with default radius and color */
   public Circle() {                    // 1st Constructor (default constructor)
      radius = 1.0;
      color = "red";
   }
   /** Constructs a Circle instance with the given radius and default color*/
   public Circle(double r) {           // 2nd Constructor
      radius = r;
      color = "red";
   }
```

Circle.java

```
/** Constructs a Circle instance with the given radius and color */
public Circle(double r, String c) { // 3rd Constructor
   radius = r;
   color = c;
}
// Public methods
/** Returns the radius */
public double getRadius() {  // getter for radius
   return radius;
}
/** Returns the color */
public String getColor() {   // getter for color
   return color;
}
/** Returns the area of this circle */
public double getArea() {
   return radius * radius * Math.PI;
}}
```

TestCircle.java

```
  //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//The radius is: 2.0
      System.out.println("The color is: " + c1.getColor());       //The color is: blue
      System.out.printf("The area is: %.2f%n", c1.getArea());  //The area is: 12.57
// 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());       //The radius is: 2.0
      System.out.println("The color is: " + c2.getColor());       //The color is: red
      System.out.printf("The area is: %.2f%n", c2.getArea());       //The area is: 12.57
```

TestCircle.java

```
// 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());
    //The radius is: 1.0
    System.out.println("The color is: " + c3.getColor());
    //The color is: red
    System.out.printf("The area is: %.2f%n", c3.getArea());
    //The area is: 3.14
  }
}
```

Constructors

A constructor is a special method that has the same method name as the class name. That is, the constructor of the class Circle is called Circle(). In the above Circle class, we define three overloaded versions of constructor Circle(...). A constructor is used to construct and initialize all the member variables. To construct a new instance of a class, you need to use a special "new" operator followed by a call to one of the constructors. For example,

```
Circle c1 = new Circle();            // use 1st constructor
Circle c2 = new Circle(2.0);          // use 2nd constructor
Circle c3 = new Circle(3.0, "red");  // use 3rd constructor
```

Constructors

A constructor method is different from an ordinary method in the following aspects:

- The name of the constructor method must be the same as the classname. By classname's convention, it begins with an uppercase (instead of lowercase for ordinary methods).
- Constructor has no return type in its method heading. It implicitly returns void. No return statement is allowed inside the constructor's body.
- Constructor can only be invoked via the "new" operator. It can only be used once to initialize the instance constructed. Once an instance is constructed, you cannot call the constructor anymore.
- Constructors are not inherited (to be explained later). Every class shall define its own constructors.

Default Constructor: A constructor with no parameter is called the default constructor. It initializes the member variables to their default values. For example, the Circle() in the above example initialize member variables radius and color to their default values.

Revisit 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).

Example: The method average() has 3 versions, with different parameter lists. The caller can invoke the chosen version by supplying the matching arguments.

```
public class MethodOverloadingTest {
   public static int average(int n1, int n2) {          // version 1
      System.out.println("Run version 1");
      return (n1+n2)/2;
   }
   public static double average(double n1, double n2) { // version 2
      System.out.println("Run version 2");
      return (n1+n2)/2;
   }
   public static int average(int n1, int n2, int n3) {  // version 3
      System.out.println("Run version 3");
      return (n1+n2+n3)/3;   }}
```

Revisit Method Overloading

```
public static void main(String[] args) {
    System.out.println(average(1, 2));
    //Run version 1//1
    System.out.println(average(1.0, 2.0));
    //Run version 2//1.5
    System.out.println(average(1, 2, 3));
    //Run version 3//2
    System.out.println(average(1.0, 2));
    //Run version 2 (int 2 implicitly casted to double 2.0)//1.5
```

Overloading

```
public static void main(String[] args) {
    System.out.println(average(1, 2));
    //Run version 1//1
    System.out.println(average(1.0, 2.0));
    //Run version 2//1.5
    System.out.println(average(1, 2, 3));
    //Run version 3//2
    System.out.println(average(1.0, 2));
    //Run version 2 (int 2 implicitly casted to double 2.0)//1.5
```