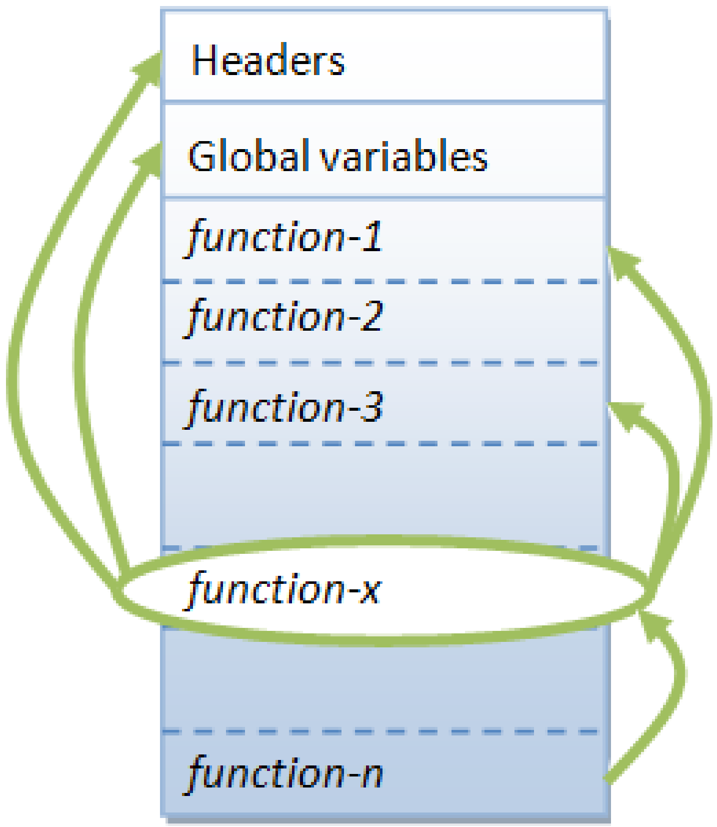
Java Programming
OOP Basics

Traditional Procedural languages

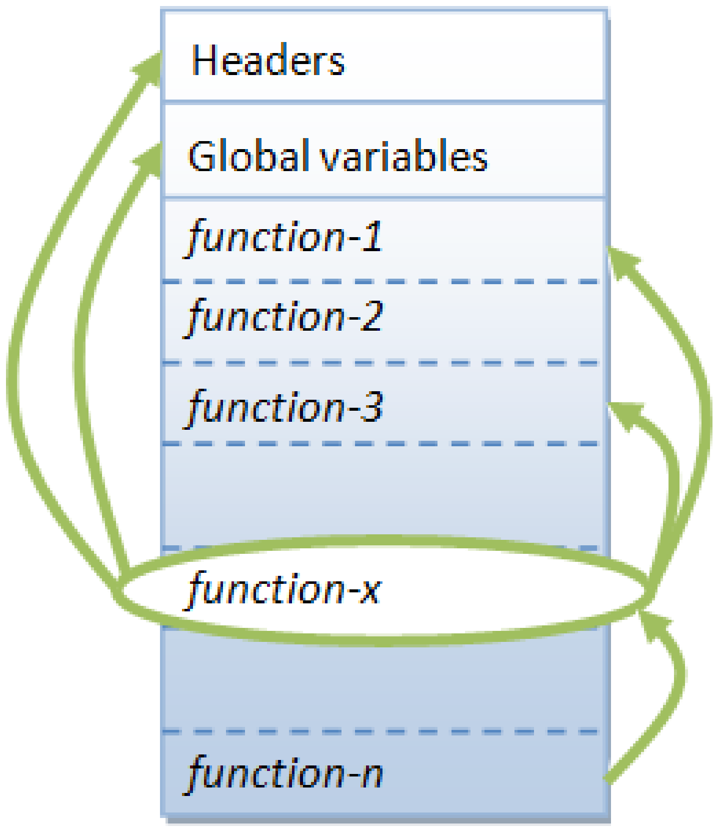
- Functions are less reusable. It is very difficult to copy a function from one program and reuse in another program.



A function (in C) is not well-encapsulated

Traditional Procedural languages

- Hard to abstract real problems such as a Customer Relationship Management (CRM) system.

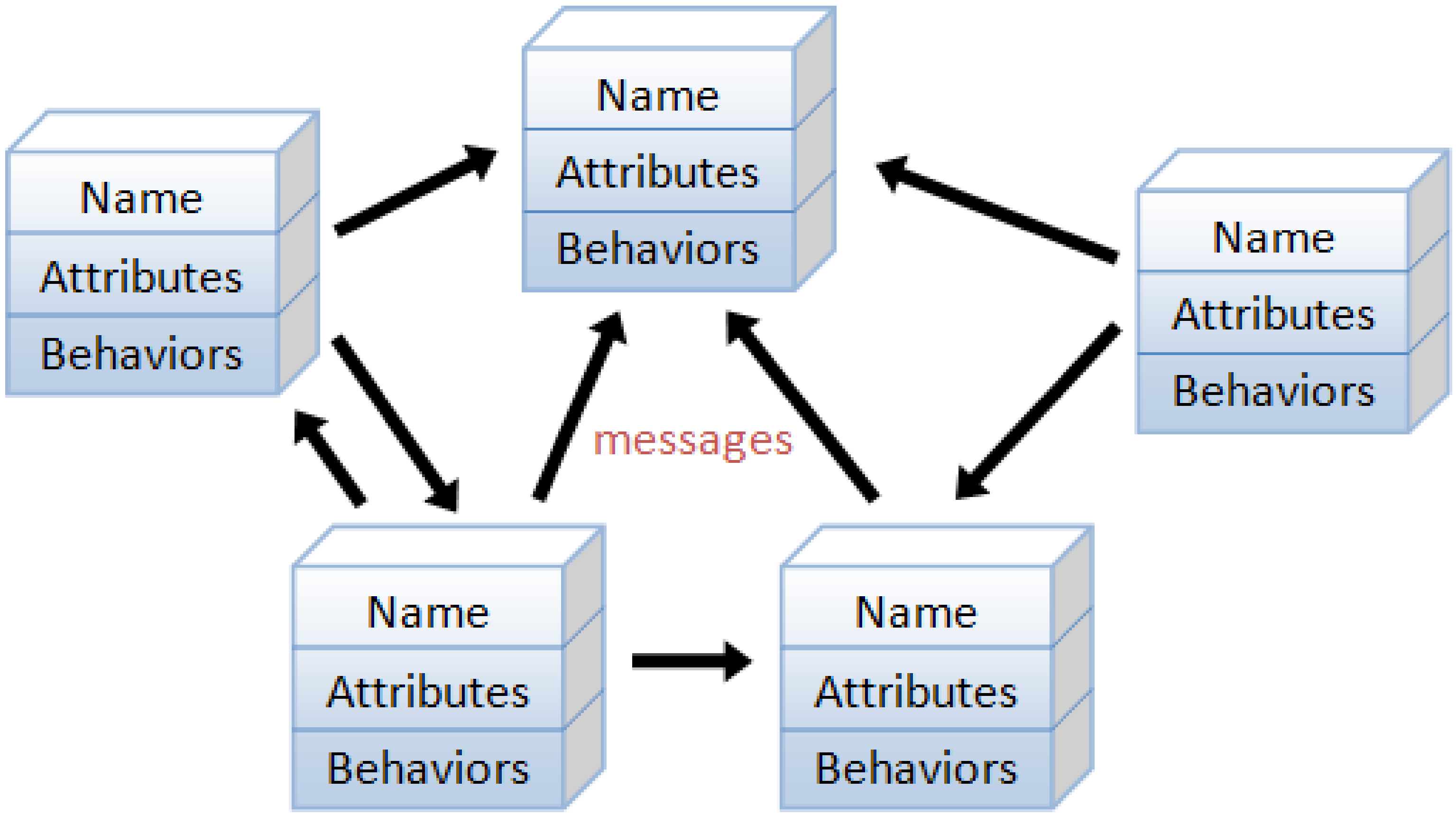


A function (in C) is not well-encapsulated

Object-Oriented Programming

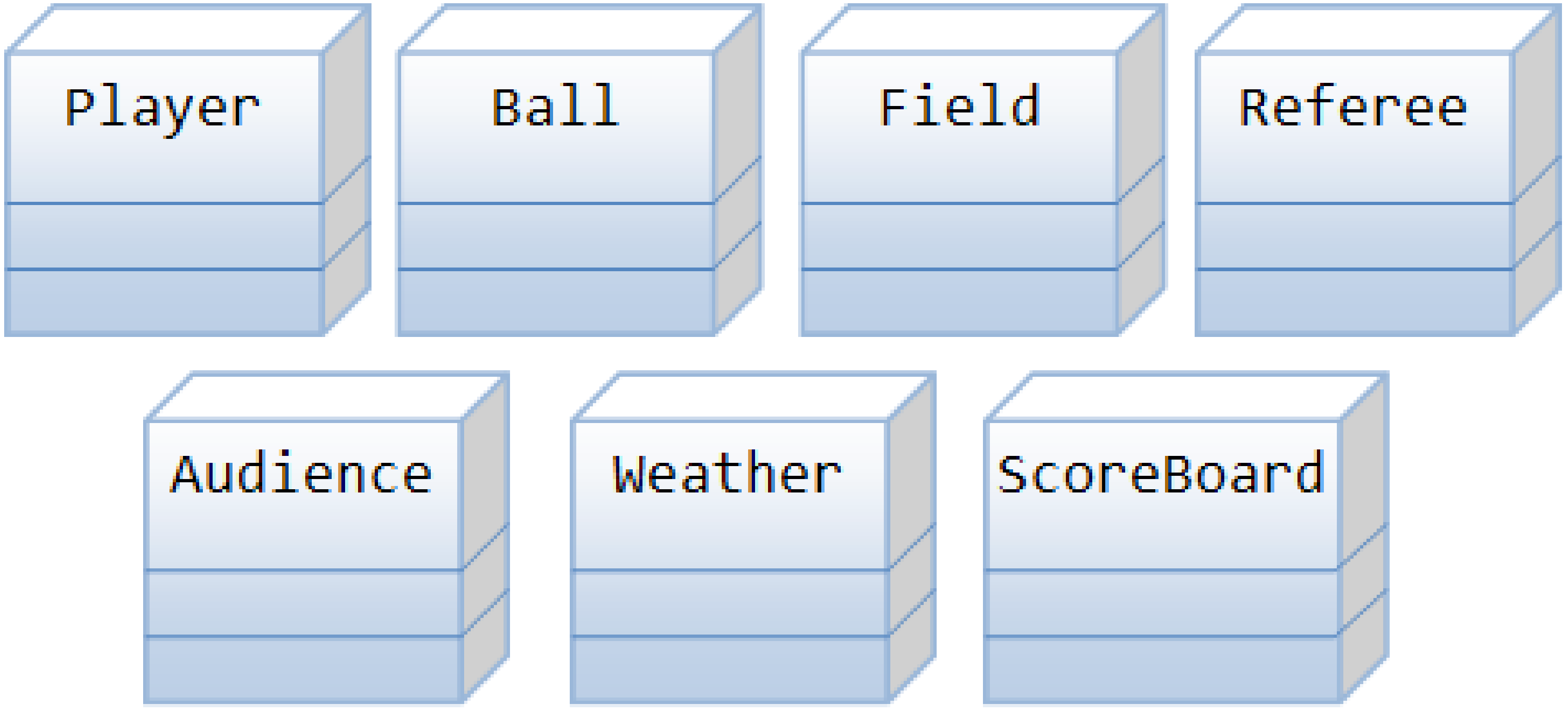
- The basic unit of OOP is a class.
- It is easier to reuse these classes.
- To represent and abstract entities of the problem space to solve the problem.
- OOP languages permit higher level of abstraction for solving real-life problems

Object-Oriented Programming



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Example



Classes (Entities) in a Computer Soccer Game

OOP in JAVA
Class & Instances

Class & Instances

- A class is a definition of objects of the same kind.
- It is sort of a blueprint or a template that defines the properties of the object and it’s functions
- Describes the static attributes and dynamic behaviors common to all objects of the same kind

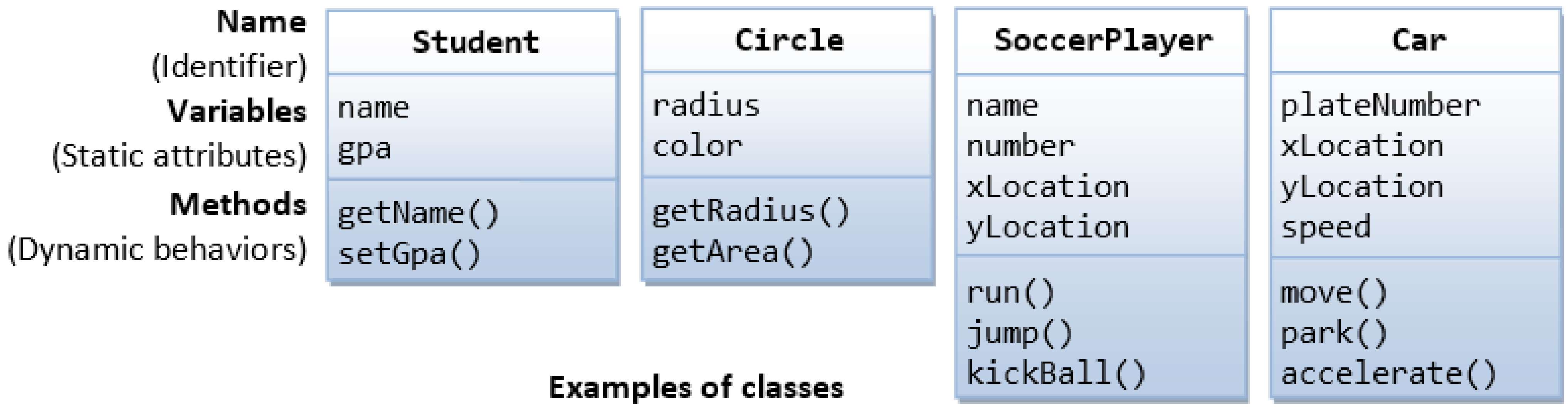
Class & Instances

- An instance is a realization of a particular item of a class
- All the instances of a class have similar properties, as described in the class definition.
- For example, you can define a class called "Student" and create three instances of the class "Student" for "Peter", "Paul" and "Pauline".

Class & Instances

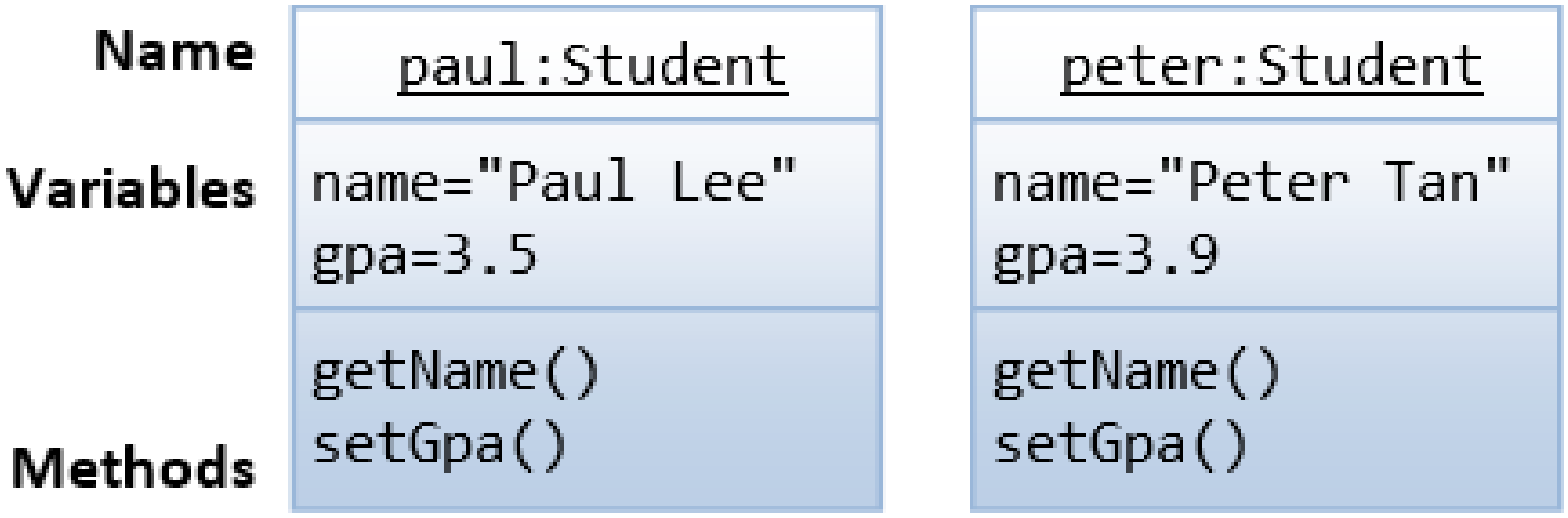
- A Class is a 3-Compartment Box Encapsulating Data and Operations:
- Name (or identity): identifies the class.
- Variables (or attribute, state, field): contains the static attributes of the class.
- Methods (or behaviors, function, operation): contains the dynamic behaviors of the class.

Example - Classes



Examples of classes

Example - Instances

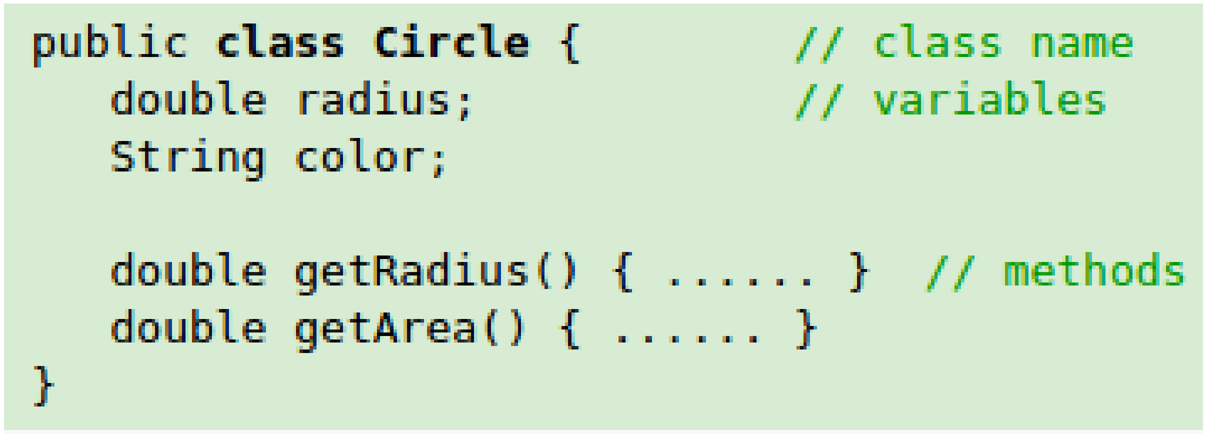


Two instances - paul and peter - of the class Student

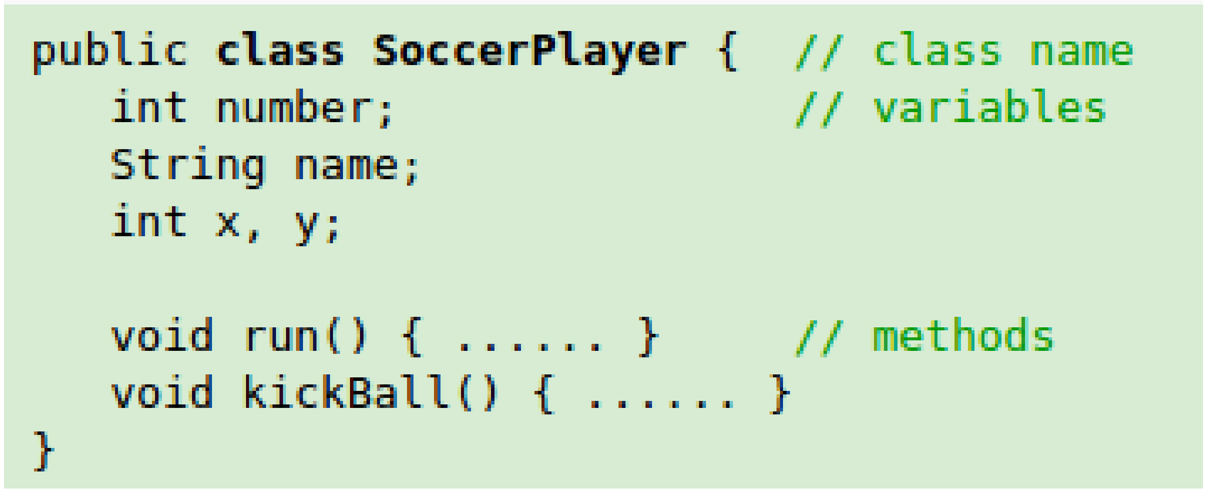
Abstraction & Encapsulation

- Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user.
- Encapsulation is the process of combining data and functions into a single unit called class.

Class Definition in Java



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

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



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

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

Class Naming Convention

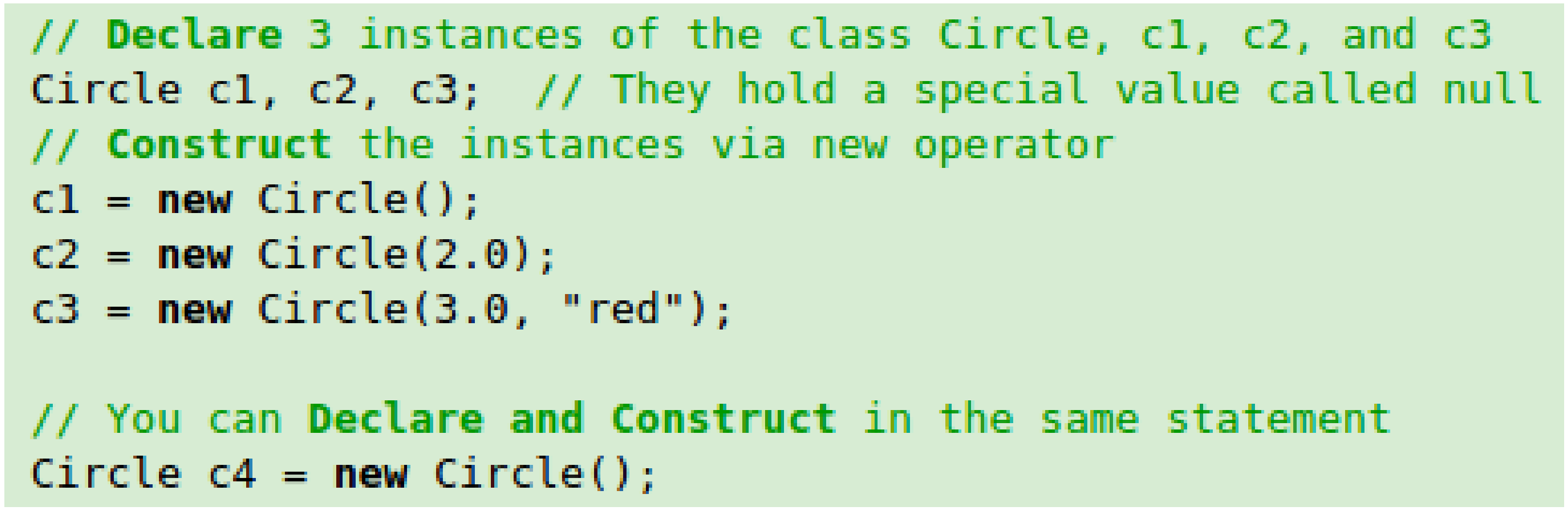
- Class name shall be a noun or a noun phrase made up of several words.
- All the words shall be initial-capitalized (camel-case)
- Use a singular noun for class name

OOP in JAVA
Creating Instances of a Class

Creating Instances of a Class

- 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(Example)

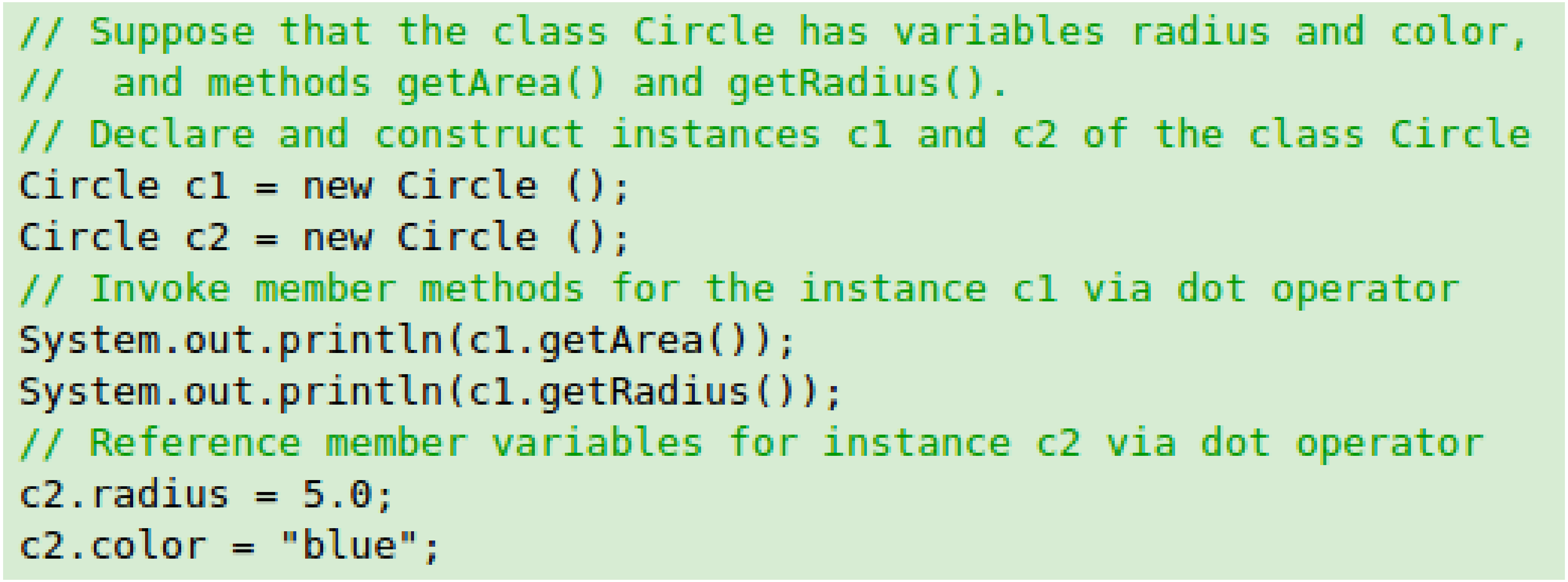


```java
// Declare 3 instances of the class Circle, c1, c2, and c3
Circle c1, c2, c3;  // They hold a special value called null
// Construct the instances via new operator
c1 = new Circle();
c2 = new Circle(2.0);
c3 = new Circle(3.0, "red");

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

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, then,
- Use the dot operator (.) to reference the desired member variable or method.

Dot (.) Operator - Example



```java
// 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

Syntax for variable definition

- [AccessControlModifier] type variableName [= initialValue];

  Example:
- private double radius;
- public int length = 1, width = 1;

Access Modifiers

- Public: The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.
- Private: The private access modifier is accessible only within class.
- Protected: The protected access modifier is accessible within package and outside the package but through inheritance only.
- Default: If you don't use any modifier, it is treated as default. The default modifier is accessible only within package
- Note: A class cannot be private or protected except nested class.

Member Methods

A method:

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

Syntax for method declaration

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



}
```

Example

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

Method Naming Convention

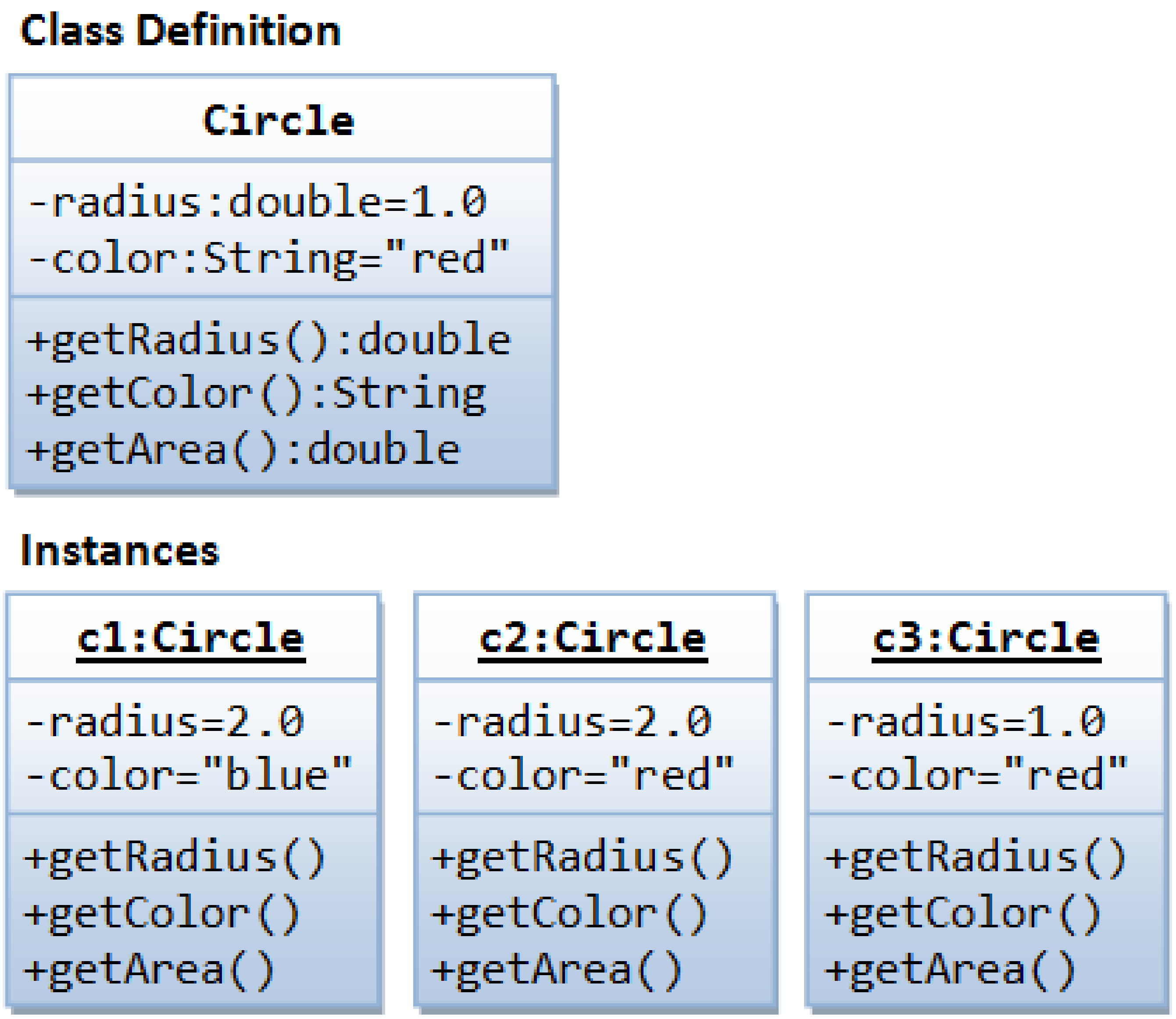
- A method name shall be a verb, or a verb phrase made up of several words
- The first word is in lowercase and the rest of the words are initial-capitalized (camel-case).

- For example, getArea(), setRadius(), getParameterValues(), hasNext()

Variable name vs. Method name vs. Class name

- A variable name is a noun, denoting an attribute
- A method name is a verb, denoting an action
- Methods take arguments in parentheses, but variables do not
- A class name is a noun beginning with uppercase

An OOP Example



Constructors

- A constructor is a special method that has the same method name as the class name.
- 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.

Constructors - Example

```
Circle c1 = new Circle();
Circle c2 = new Circle(2.0);
Circle c3 = new Circle(3.0, "red");
```

More on Constructors

- The name of the constructor method is the same as the class name.
- Constructor has no return type. It implicitly returns void.
- Constructor can only be invoked via the "new" operator.
- It can only be used once to initialize the instance constructed.
- Constructors are not inherited

Default Constructor

- A constructor with no parameter is called the default constructor.
- It initializes the member variables to their default value.

Method Overloading

- Method overloading means that the same method name can have different implementations (versions).

- the different implementations must be distinguishable by their parameter list (either the number of parameters, or the type of parameters, or their order).

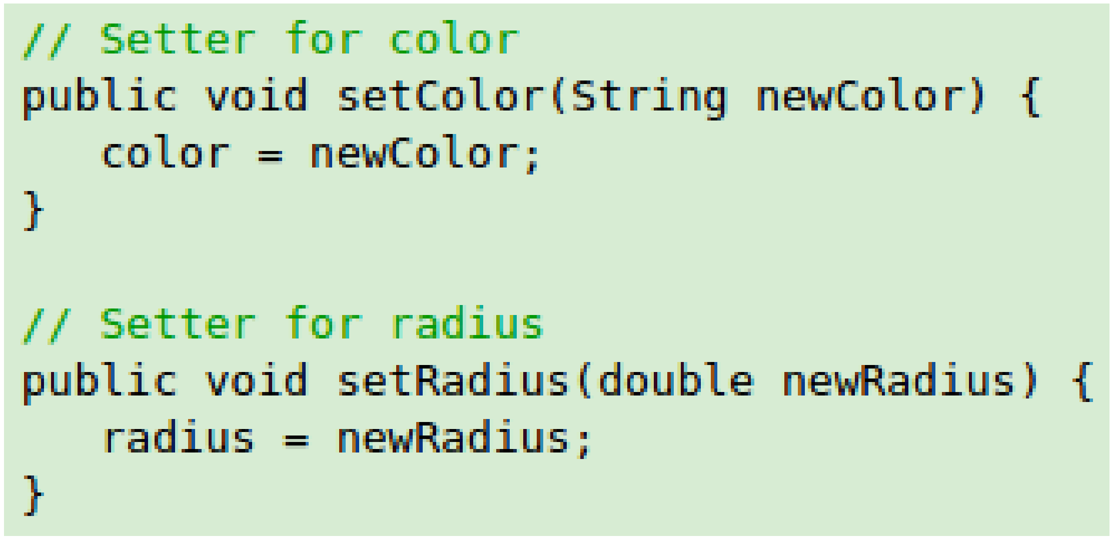
Method Overloading - Example

```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
   }
}
```

Getters and Setters

- Rule of Thumb: Do not make any variable public, unless you have a good reason.
- Public Getters and Setters methods for private Variables.
- To allow other classes to read the value of a private variable say abc, we provide a get method called getAbc().
- To allow other classes to modify the value of a private variable say abc, we provide a set method called setAbc().

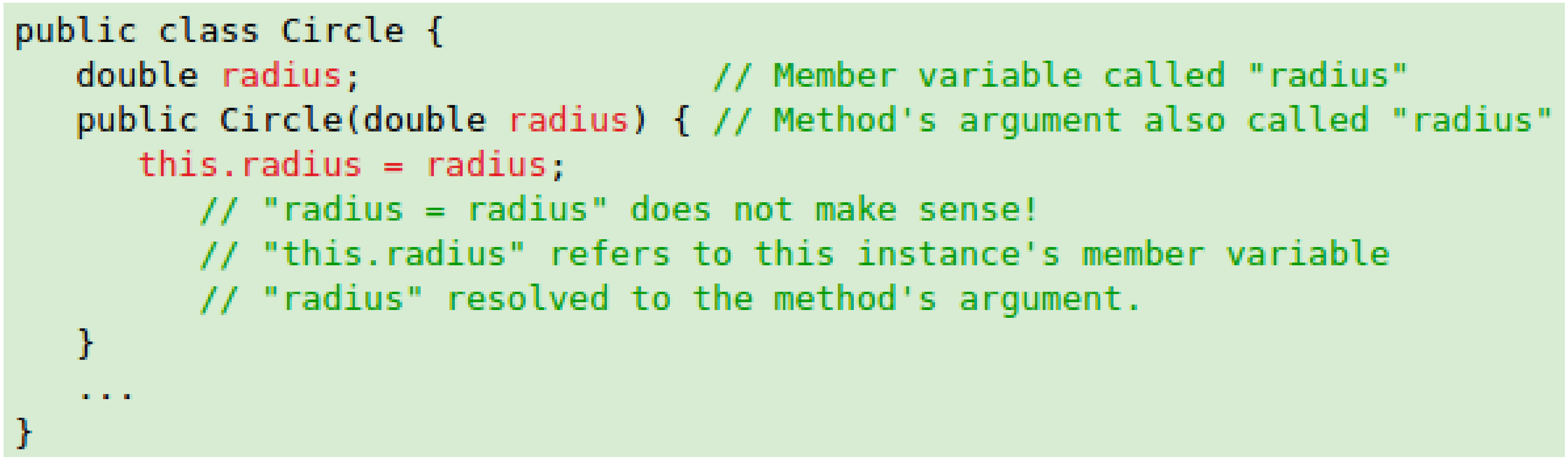
Getters and Setters



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

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

Keyword "this"

- You can use keyword "this" to refer to this instance inside a class definition.
- One of the main usage of keyword this is to resolve ambiguity.



```java
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.
   }
   ...
}
```

Keyword “this” - Example

```
public class Aaa {
   // A private variable named xxx of the type T
   private T xxx;

   // Constructor
   public Aaa(T xxx) {
      this.xxx = xxx;
   }

   // A getter for variable xxx of type T receives no argument and return a value of type T
   public T getXxx() {
      return xxx;  // or "return this.xxx" for clarity
   }

   // A setter for variable xxx of type T receives a parameter of type T and return void
   public void setXxx(T xxx) {
      this.xxx = xxx;
   }
}
```

More on "this"

- this.varName refers to varName of this instance;
- this.methodName(...) invokes methodName(...) of this instance.
- In a constructor, we can use ‘this’ to call another constructor of this class.
- Inside a method, we can use the statement "return this" to return this instance to the caller.

Method toString()

- Every well-designed Java class should have a public method called toString()
- That returns a string description of this instance.
- You can invoke the toString() method explicitly by calling anInstanceName.toString()
- or implicitly via println()
- or String concatenation operator '+'

Method toString() - Example

- For example, include the following toString() method in our Circle class

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

In some other class, you can get a description of a Circle instance via:

```
Circle c1 = new Circle();
System.out.println(c1.toString());   // Explicitly calling toString()
System.out.println(c1);              // Implicit call to c1.toString()
System.out.println("c1 is: " + c1);  // '+' invokes toString() to get a String before concatenation
```

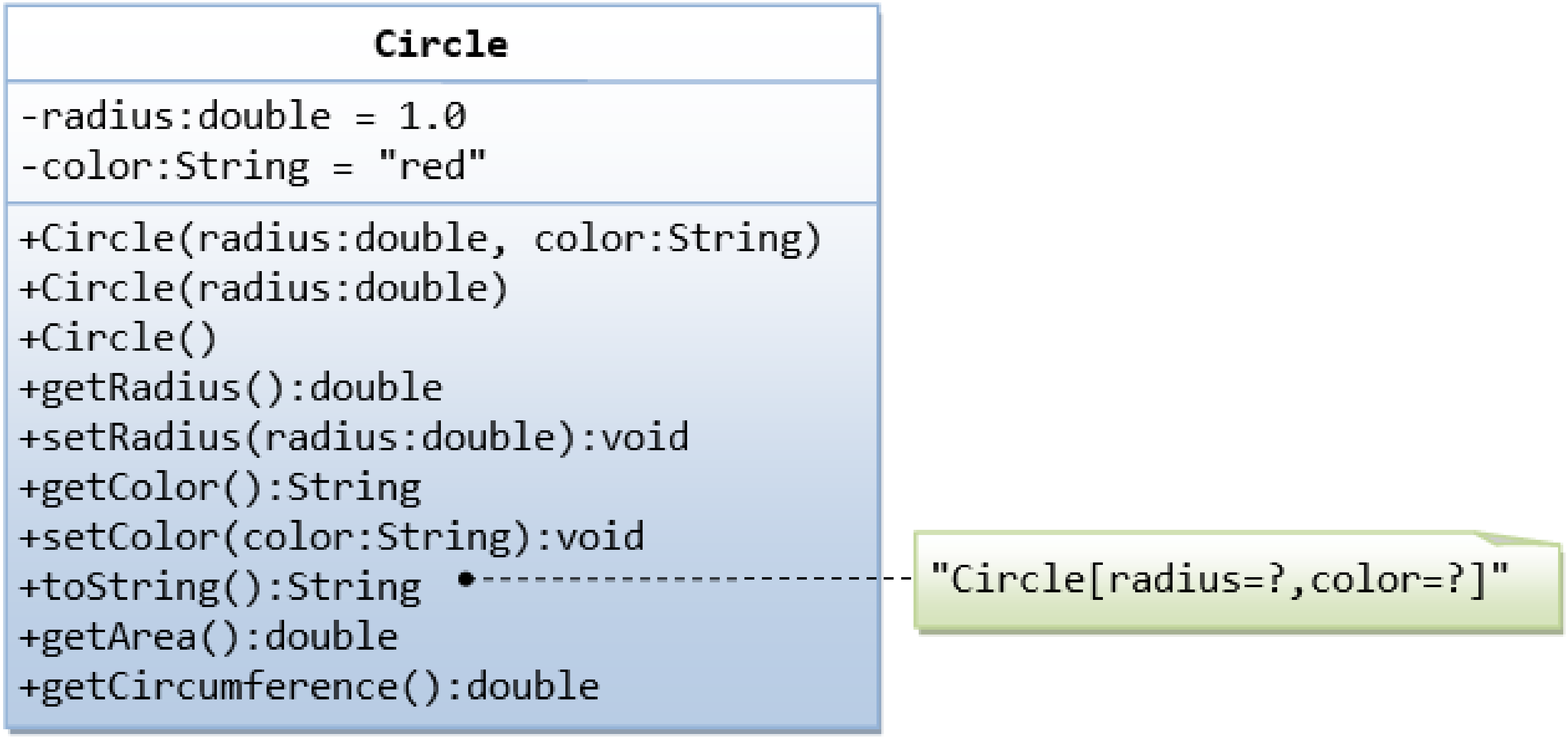
Constants (final)

- Constants are variables defined with the modifier final.
- A final variable can only be assigned once and its value cannot be modified once assigned.
- public final double X_REFERENCE = 1.234;
- private final int MAX_ID = 9999;
  MAX_ID = 10000; // error: cannot assign a value to final variable MAX_ID

Constants (final)

- Constant Naming Convention: A constant name is a noun, or a noun phrase made up of several words. All words are in uppercase separated by underscores '_', for examples, X_REFERENCE, MAX_INTEGER and MIN_VALUE.
- A final primitive variable cannot be re-assigned a new value.
- A final instance cannot be re-assigned a new object.
- A final class cannot be sub-classed (or extended).
- A final method cannot be overridden.

Putting Them Together



Reference

- https://www3.ntu.edu.sg/home/ehchua/programming/java/J3a_OOPBasics.html