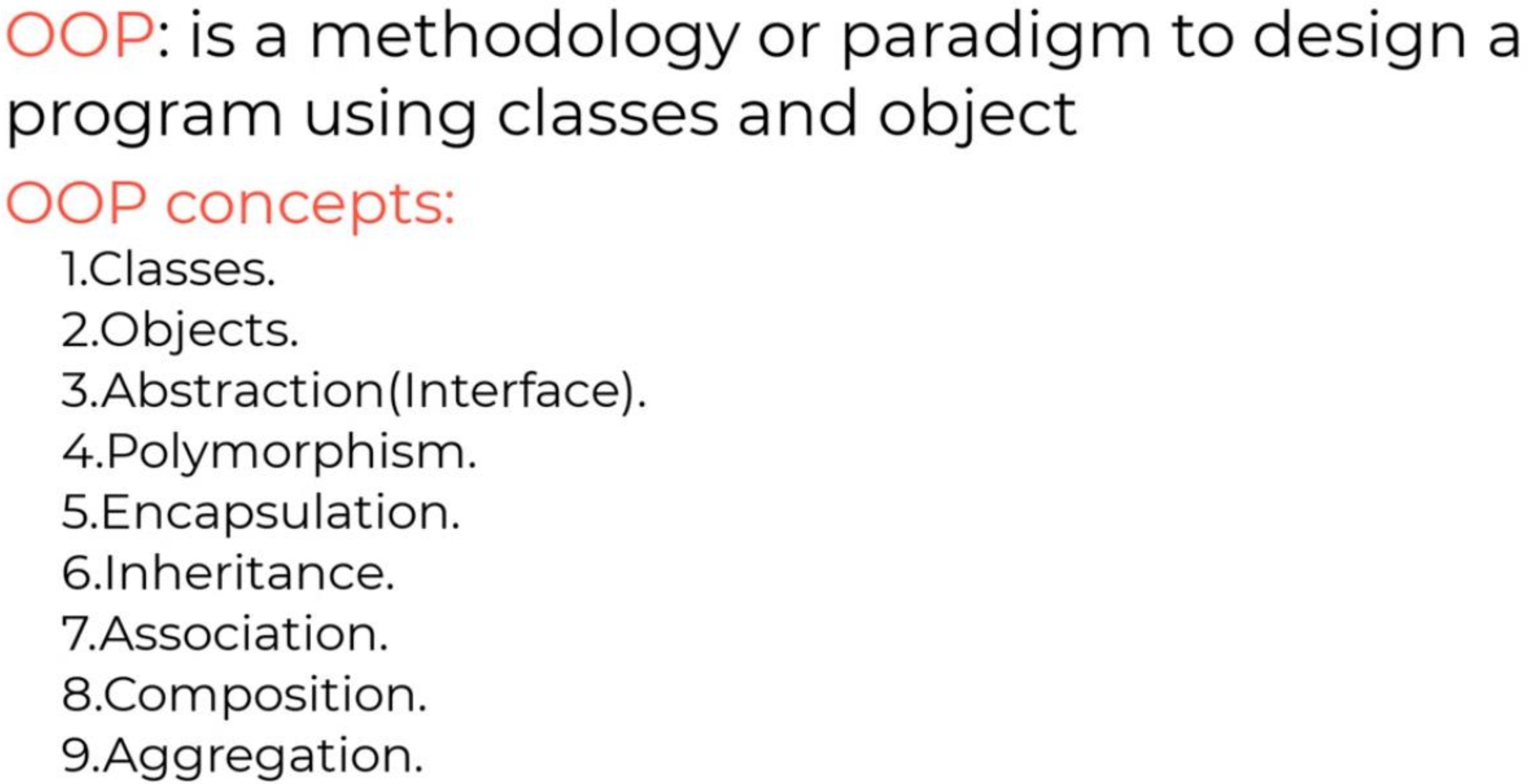
PROGRAMMING 2
CHAPTER 9

Gaza University
Musbah j. Mousa

OBJECT-ORIENTED PROGRAMMING



OOP: is a methodology or paradigm to design a program using classes and object

OOP concepts:

1.Classes.
2.Objects.
3.Abstraction(Interface).
4.Polymorphism.
5.Encapsulation.
6.Inheritance.
7.Association.
8.Composition.
9.Aggregation.

OBJECT-ORIENTED PROGRAMMING

Advantages:

Reusability

Readability

Performance

Space

IS OOP PROGRAMMING LANGUAGE?

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

Object-oriented programming (OOP) is a fundamental programming paradigm used by nearly every developer at some point in their career.

Object-oriented programming (OOP) is a fundamental programming Methodolgy used by nearly every developer at some point in their career.

OOP LANGUAGES

Java

C++

C#

Python

PHP

Visual Basic.NET

JavaScript

Ruby

PROGRAMMING PARADIGMS

imperative in which the programmer instructs the machine how to change its state,

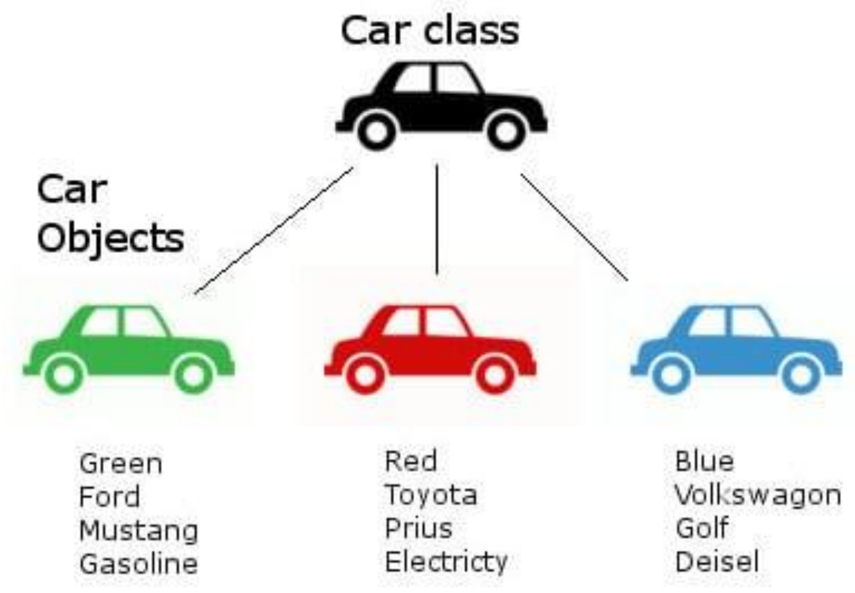
- procedural which groups instructions into procedures,
- object-oriented which groups instructions with the part of the state they operate on,

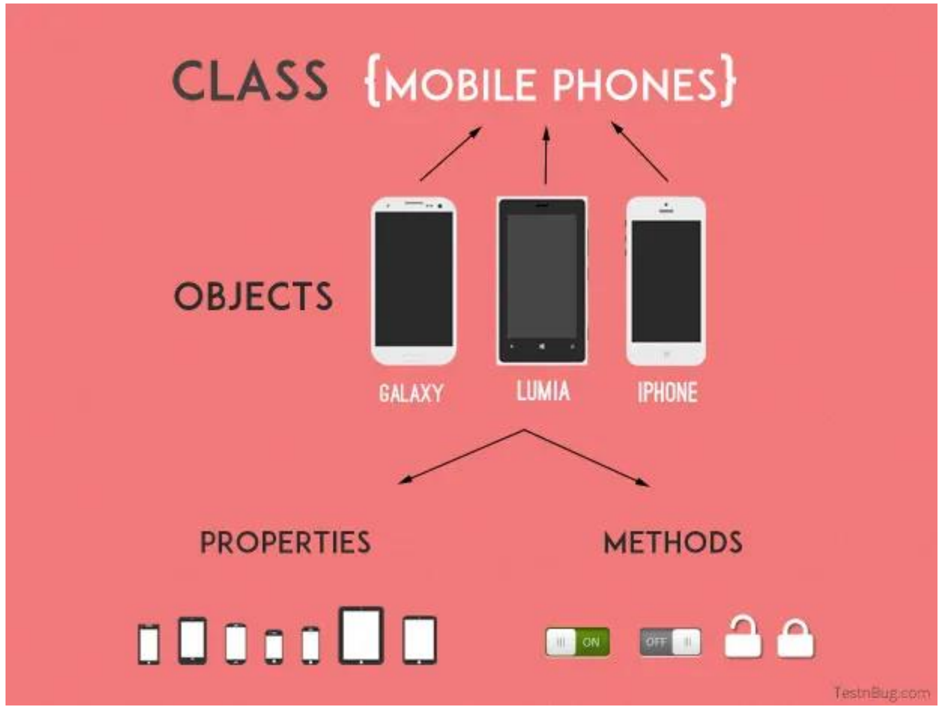
declarative in which the programmer merely declares properties of the desired result, but not how to compute it

- functional in which the desired result is declared as the value of a series of function applications,
- logic in which the desired result is declared as the answer to a question about a system of facts and rules,
- mathematical in which the desired result is declared as the solution of an optimization problem
- reactive in which the desired result is declared with data streams and the propagation of change

OOP

Every thing is object !





OOP

E-Store System : What are objects?

Dental Clinic: What are objects?

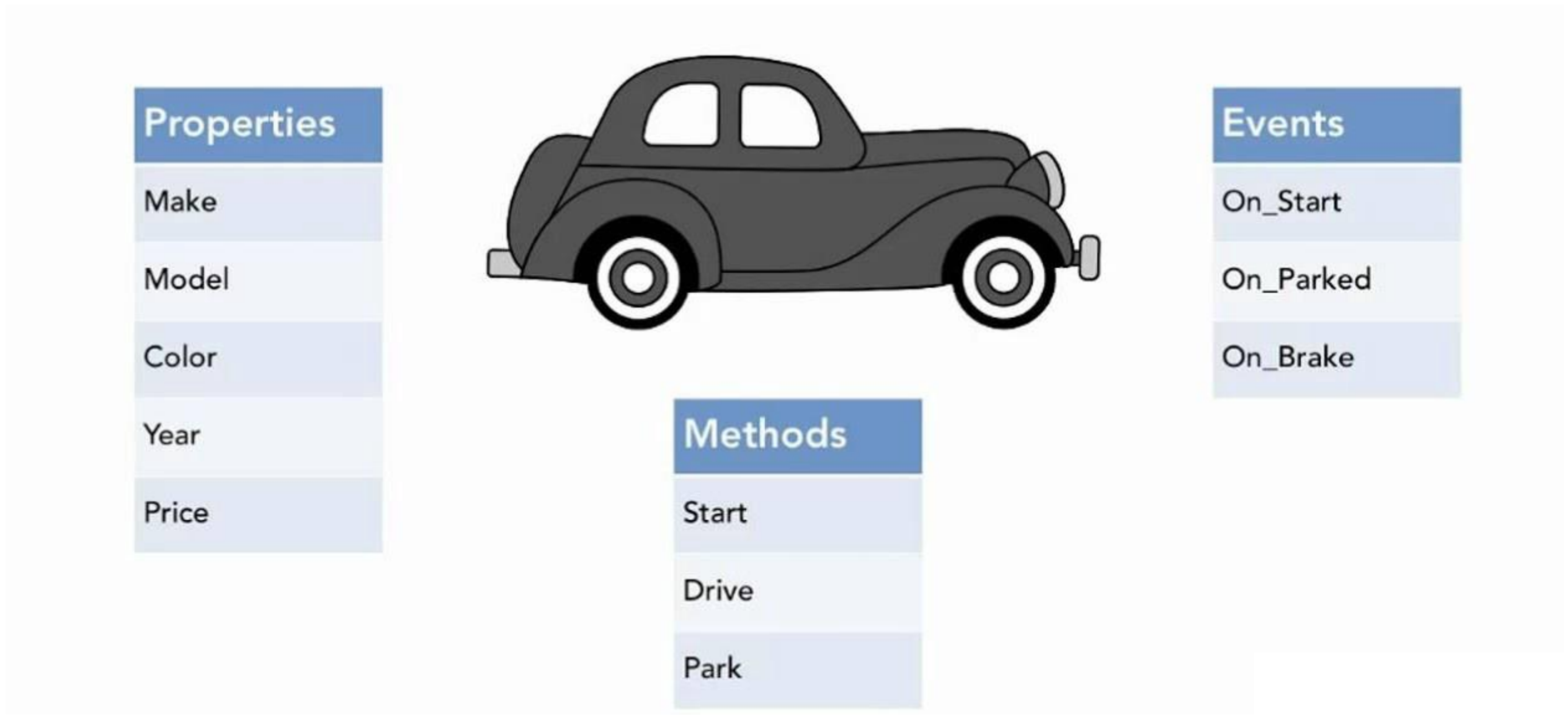
OOP

Object

Data - Properties - Attributes

Operations() - Methods() - Functions() - Behaviors()

OOP

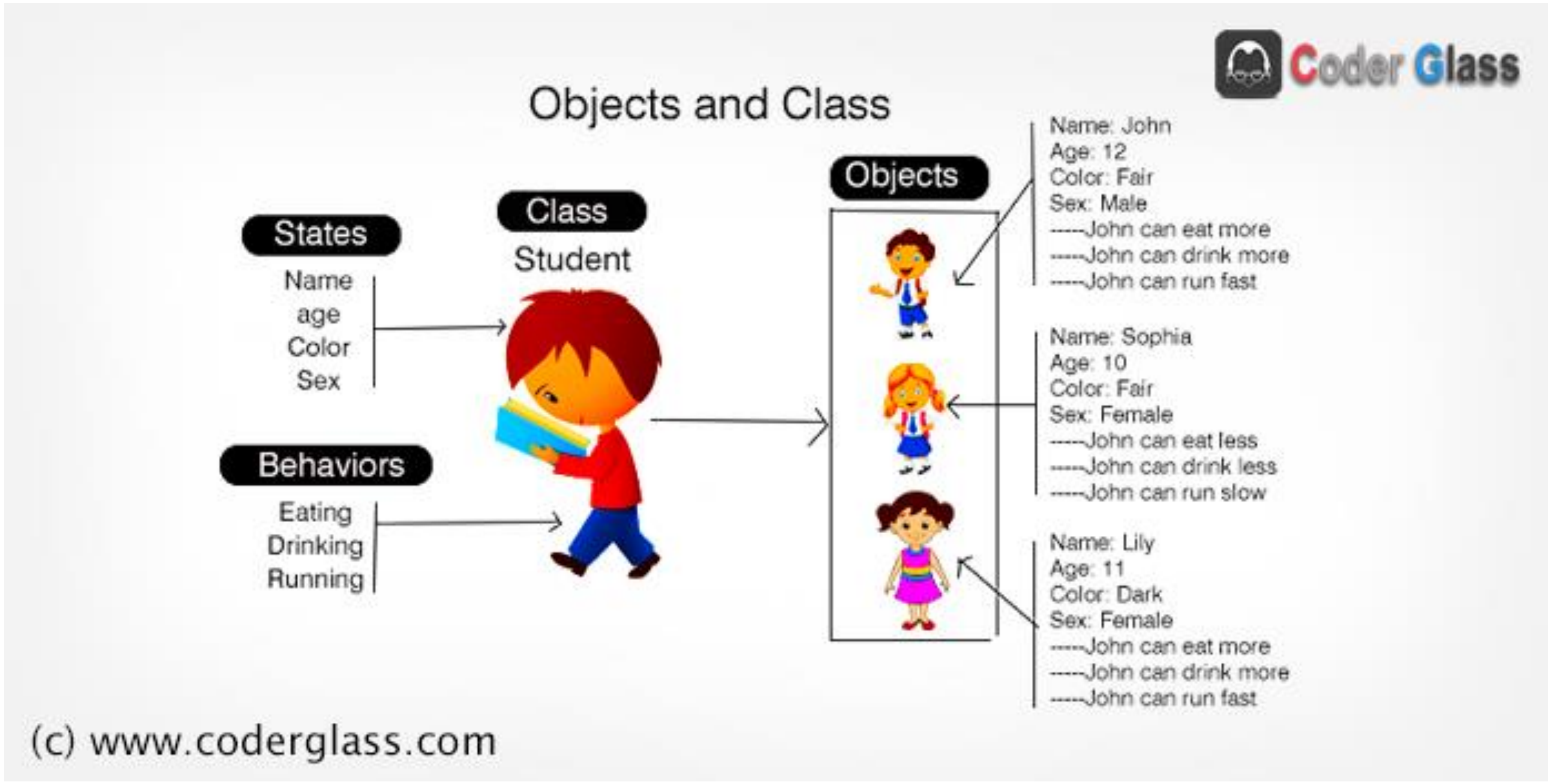


Properties
Make
Model
Color
Year
Price

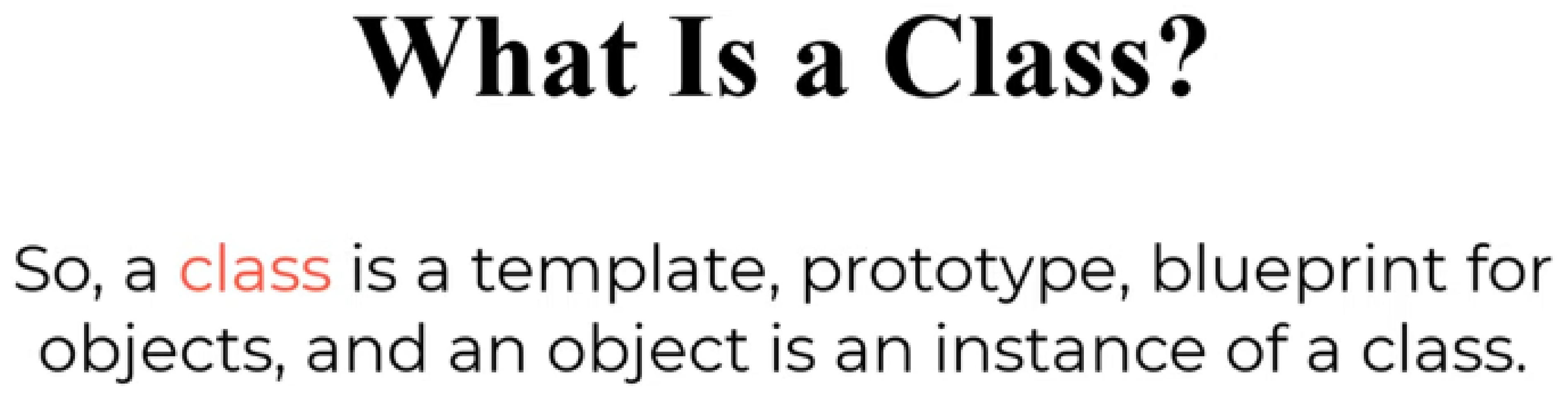
Methods
Start
Drive
Park

Events
On_Start
On_Parked
On_Brake

OOP
Student as Object



OOP



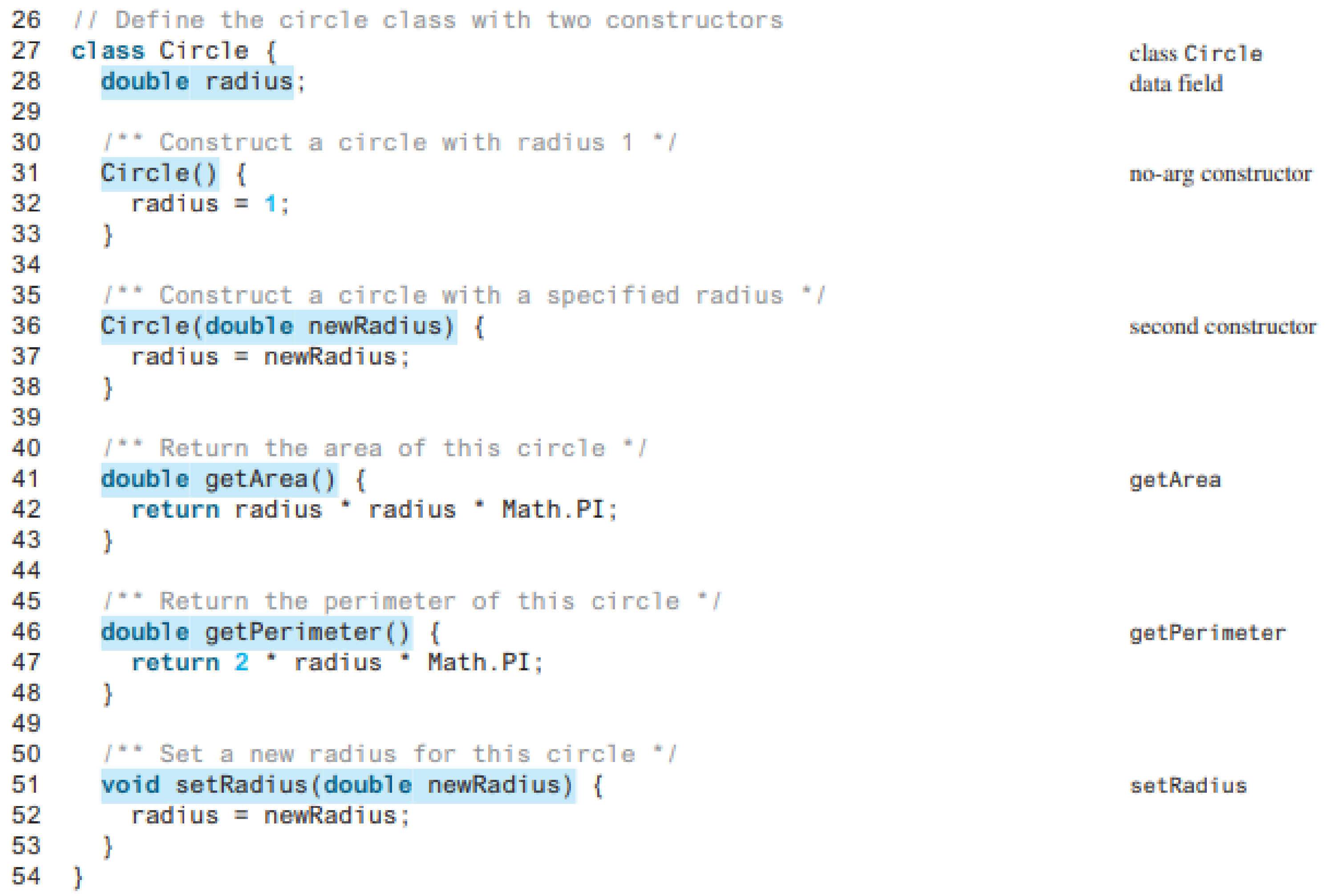
What Is a Class?

So, a class is a template, prototype, blueprint for objects, and an object is an instance of a class.

SYNTAX OF CLASS

```
class <ClassName>
{
attributes/variables;
Constructors();
methods();
}
```

SYNTAX OF CLASS



```
// Define the circle class with two constructors
class Circle {
  double radius;

  /** Construct a circle with radius 1 */
  Circle() {
    radius = 1;
  }

  /** Construct a circle with a specified radius */
  Circle(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;
  }
}
```

class Circle
data field
no-arg constructor
second constructor
getArea
getPerimeter
setRadius

INSTANCE

- Instance is an Object of a class which is an entity with its own attribute values and methods.

- **<u>Creating an Instance</u>**

```
ClassName  refVariable;
refVariable = new Constructor();
  or
ClassName  refVariable = new  Constructor();
```

CONSTRUCTOR

- A constructor initializes the instance variables of an object.
- It is called immediately after the object is created but before the new operator completes.
  1) it is syntactically similar to a method:
  2) it has the same name as the name of its class
  3) it is written without return type; the default return type of a class.
- constructor is the same class When the class has no constructor, the default constructor automatically initializes all its instance variables with zero.

CONSTRUCTOR

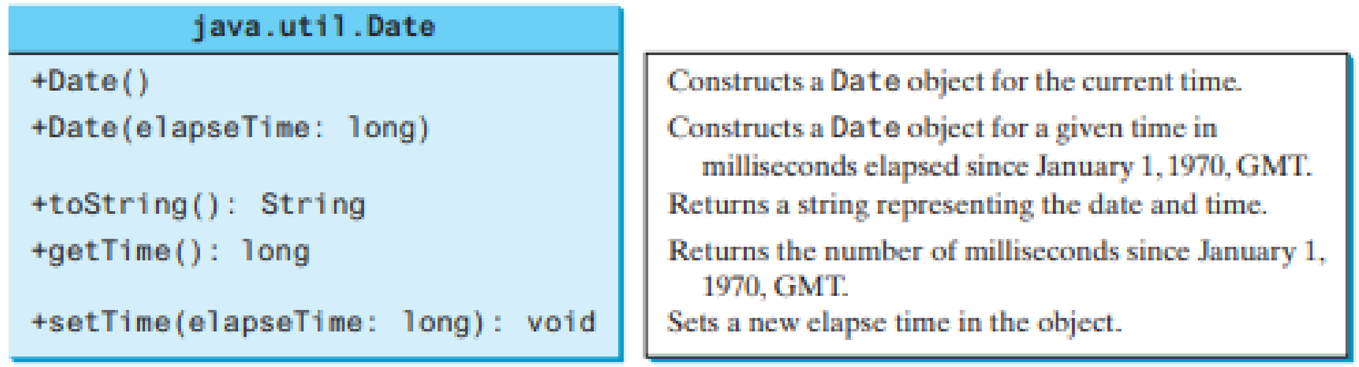
-
  ```
  class Box {
  double width;
  double height;
  double depth;
  Box() {
  System.out.println("Constructing Box");
  width = 10; height = 10; depth = 10;
  }
  double volume() {
  return width * height * depth;
  }
  }
  ```

PARAMETERIZED CONSTRUCTOR

-
  ```
  class Box {
  double width;
  double height;
  double depth;
  Box(double w, double h, double d) {
  width = w; height = h; depth = d;
  }
  double volume()
  { return width * height * depth;
  }
  }
  ```

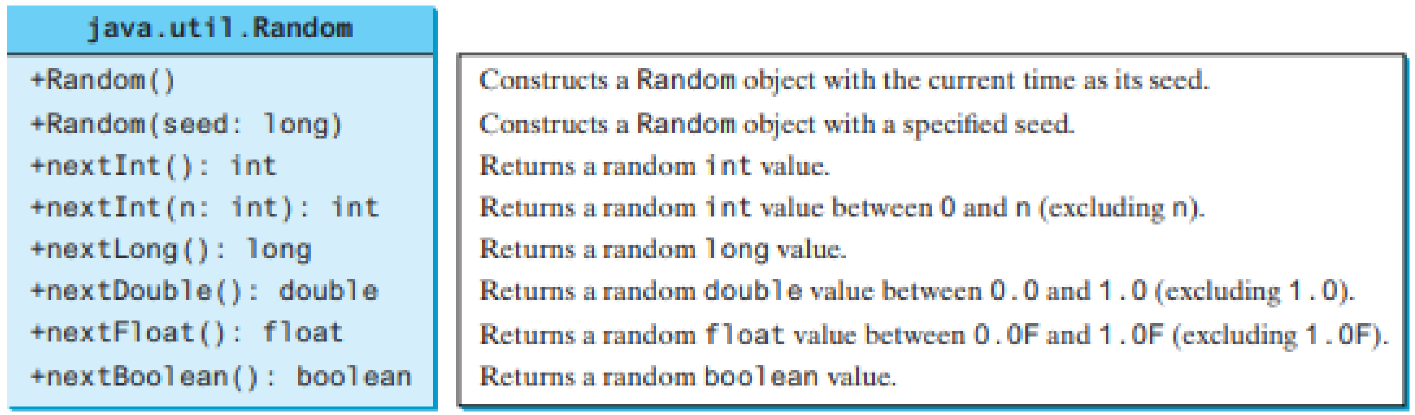
CLASSES OF JAVA LIBRARY

- Date Class
- Random Class



java.util.Date	
+Date()	Constructs a Date object for the current time.
+Date(elapseTime: long)	Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.
+toString(): String	Returns a string representing the date and time.
+getTime(): long	Returns the number of milliseconds since January 1, 1970, GMT.
+setTime(elapseTime: long): void	Sets a new elapse time in the object.

A Date object represents a specific date and time.



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 (excluding n).
+nextLong(): long	Returns a random long value.
+nextDouble(): double	Returns a random double value between 0.0 and 1.0 (excluding 1.0).
+nextFloat(): float	Returns a random float value between 0.0F and 1.0F (excluding 1.0F).
+nextBoolean(): boolean	Returns a random boolean value.

11 A Random object can be used to generate random values.

STATIC

- **if a field is declared *static*, then exactly a single copy of that field is created and shared among all instances of that class.**

```
public class Car {
    private String name;
    private String engine;

    public static int numberOfCars;

    public Car(String name, String engine) {
        this.name = name;
        this.engine = engine;
        numberOfCars++;
    }
}
```

DATA FIELD ENCAPSULATION

- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.
- To achieve encapsulation in Java −
  - Declare the variables of a class as private.
  - Provide public setter and getter methods to modify and view the variables values.

DATA FIELD ENCAPSULATION

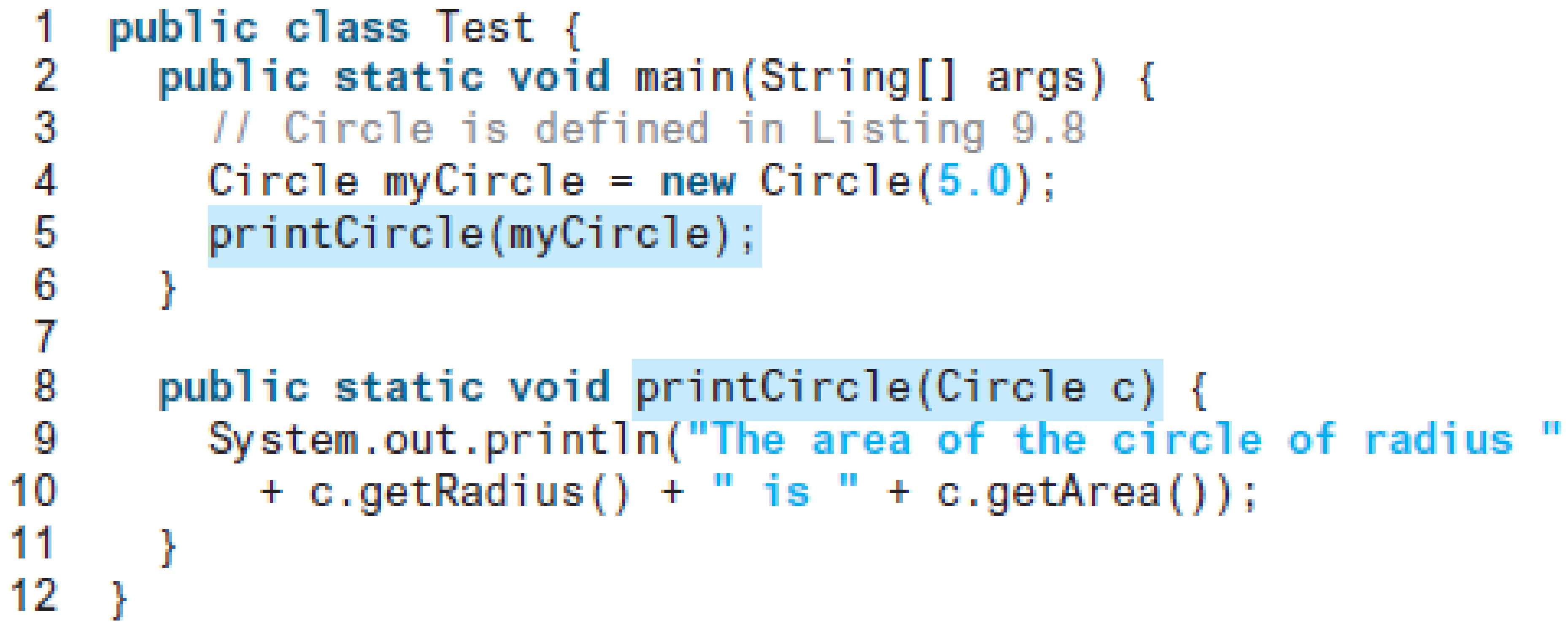
```
public class Student {
  private String name;
  private String id;
  private int age;

  public int getAge() {
    return age;
  }

  public String getName() {
    return name;
  }
public String getId() {
    return id;
  }

  public void setAge( int newAge) {
    age = newAge;
  }

  public void setName(String newName) {
    name = newName;
  }
  public void setId( String newId) {
    id = newId;
  }
}
```

VISIBILITY MODIFIERS

- ***Visibility modifiers can be used to specify the visibility of a class and its members.***
  - ***Public***: classes, methods, and data fields to denote they can be accessed from any other classes.
  - ***Protected:*** makes methods and data fields accessible only from within its own class and any class inherit from it.
  - ***Private:*** makes methods and data fields accessible only from within its own class.

PASSING OBJECTS TO METHODS

- *Passing an object to a method is to pass the reference of the object.*



```
public class Test {
  public static void main(String[] args) {
    // Circle is defined in Listing 9.8
    Circle myCircle = new Circle(5.0);
    printCircle(myCircle);
  }

  public static void printCircle(Circle c) {
    System.out.println("The area of the circle of radius "
      + c.getRadius() + " is " + c.getArea());
  }
}
```

ARRAY OF OBJECTS

- Circle[] circleArray = **new** Circle[**10**];

To initialize **circleArray**, you can use a **for** loop as follows:

```
for (int i = 0; i < circleArray.length; i++) {
circleArray[i] = new Circle();
{
```

IMMUTABLE OBJECTS AND CLASSES

- For a class to be immutable, it must meet the following requirements:
  - All data fields must be private.
  - There can’t be any mutator methods for data fields.
  - No accessor methods can return a reference to a data field that is mutable.

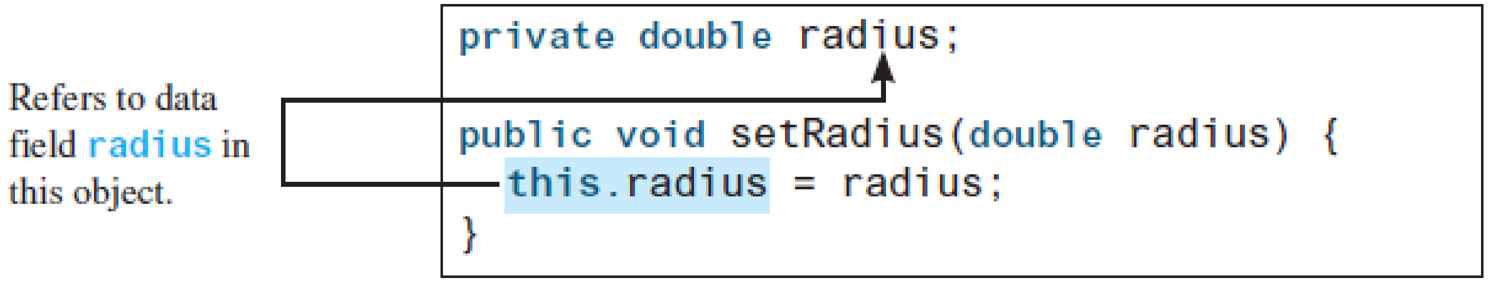
IMMUTABLE OBJECTS AND CLASSES

```
public class Student {
private int id;
private String name;
private java.util.Date dateCreated;
public Student(int ssn, String newName) {
id = ssn;
name = newName;
dateCreated = new java.util.Date();
}
```

```
public int getId() {
return id;
{
public String getName() {
return name;
{
public java.util.Date getDateCreated() {
return dateCreated;
}
}
```

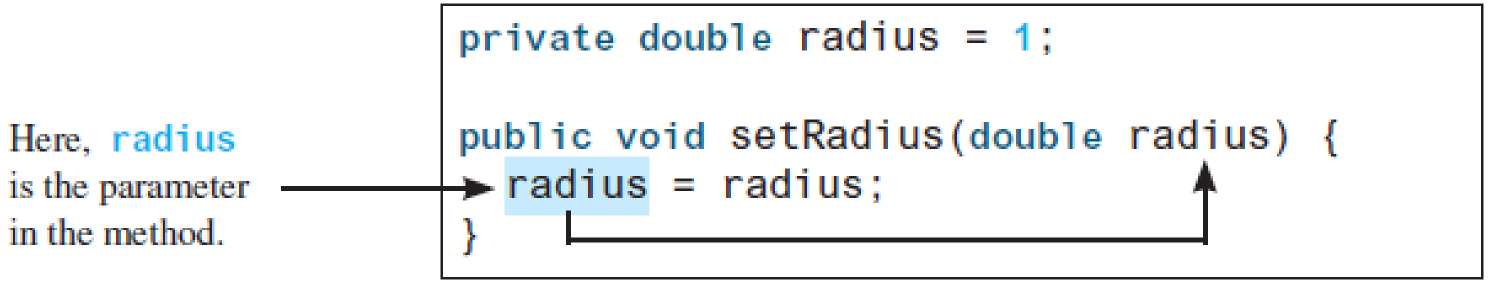
THE **THIS** REFERENCE

The keyword **this** *refers to the object itself. It can also be used inside a constructor to invoke another constructor of the same class.*



```
private double radius;

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

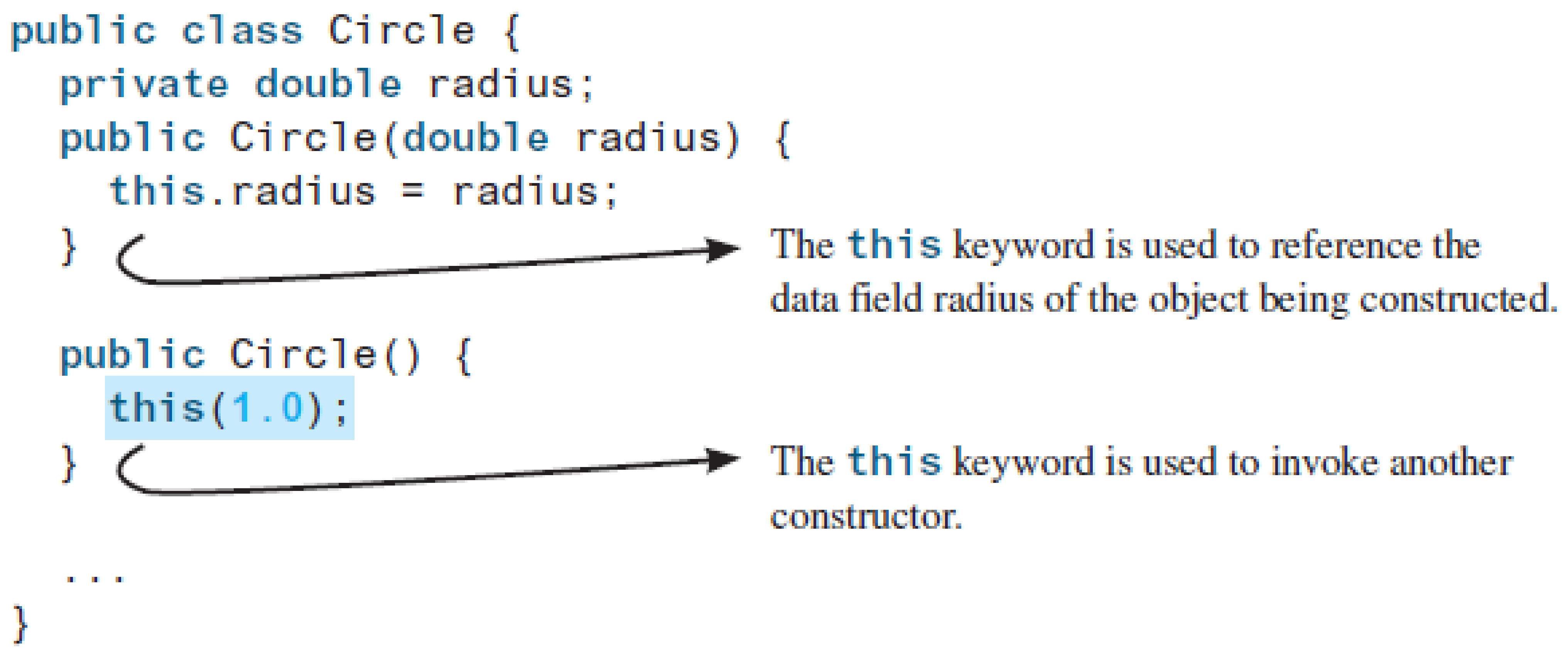
(a) this.radius refers the radius data field in this object.



```
private double radius = 1;

public void setRadius(double radius) {
  radius = radius;
}
```

(b) radius is the parameter defined in the method header.

USING **THIS** TO INVOKE A CONSTRUCTOR



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

  public Circle() {
    this(1.0);
  }

  ...
}
```

The `this` keyword is used to reference the data field radius of the object being constructed.

The `this` keyword is used to invoke another constructor.

LAB TRAINING

1) The Java class called Holiday is started below. An object of class Holiday represents a holiday during the year. This class has three instance variables:

- name, which is a String representing the name of the holiday
- day, which is an int representing the day of the month of the holiday
- month, which is a String representing the month the holiday is in

```
public class Holiday {
private String name;
private int day;
private String month;
// your code goes here
}
```

a) Write a constructor for the class Holiday, which takes a String representing the name, an int representing the day, and a String representing the month as its arguments, and sets the class variables to these values.

b) Write a method inSameMonth, which compares two instances of the class Holiday, and returns the Boolean value true if they have the same month, and false if they do not.

c) Write a method avgDate which takes an array of base type Holiday as its argument, and returns a double that is the average of the day variables in the Holiday instances in the array. You may assume that the array is full (i.e. does not have any null entries).