



COMPUTER SCIENCE DEPARTMENT FACULTY OF ENGINEERING AND TECHNOLOGY  
**OBJECT-ORIENTED PROGRAMMING**

**COMP2311**

Instructor :Murad Njoum  
Office : Masri322

Chapter 9 Objects and Classes - Revesion

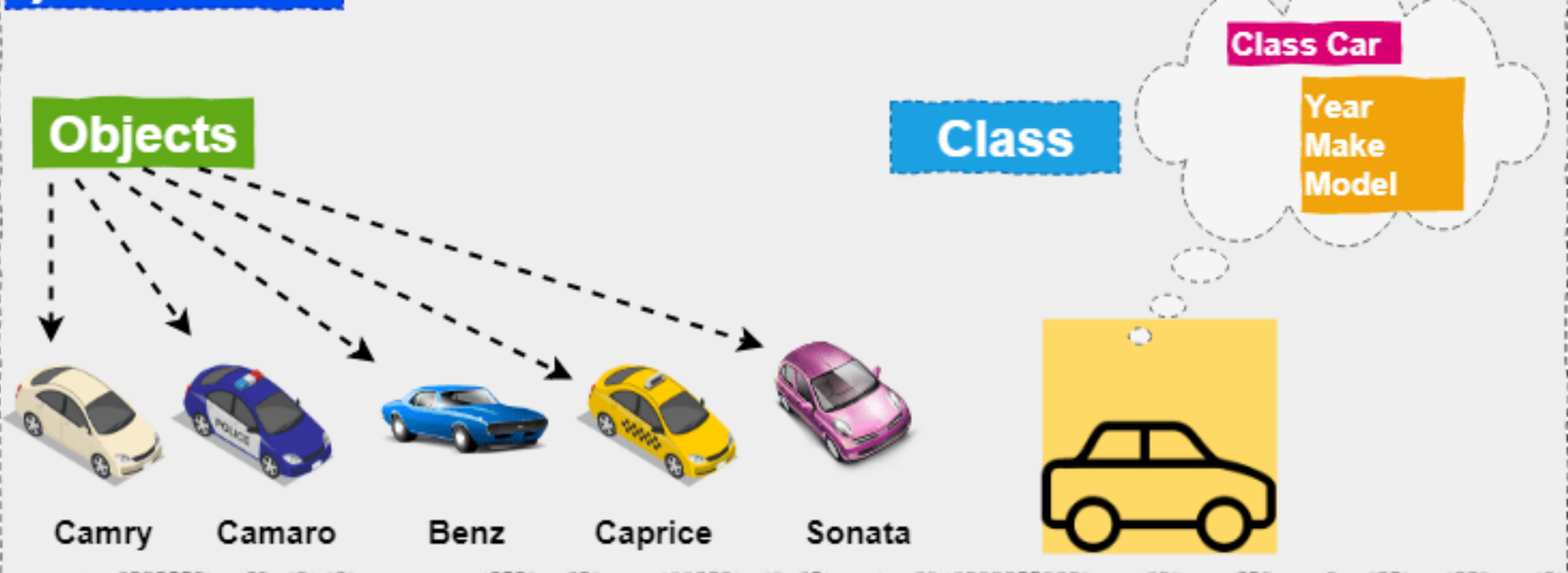
liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

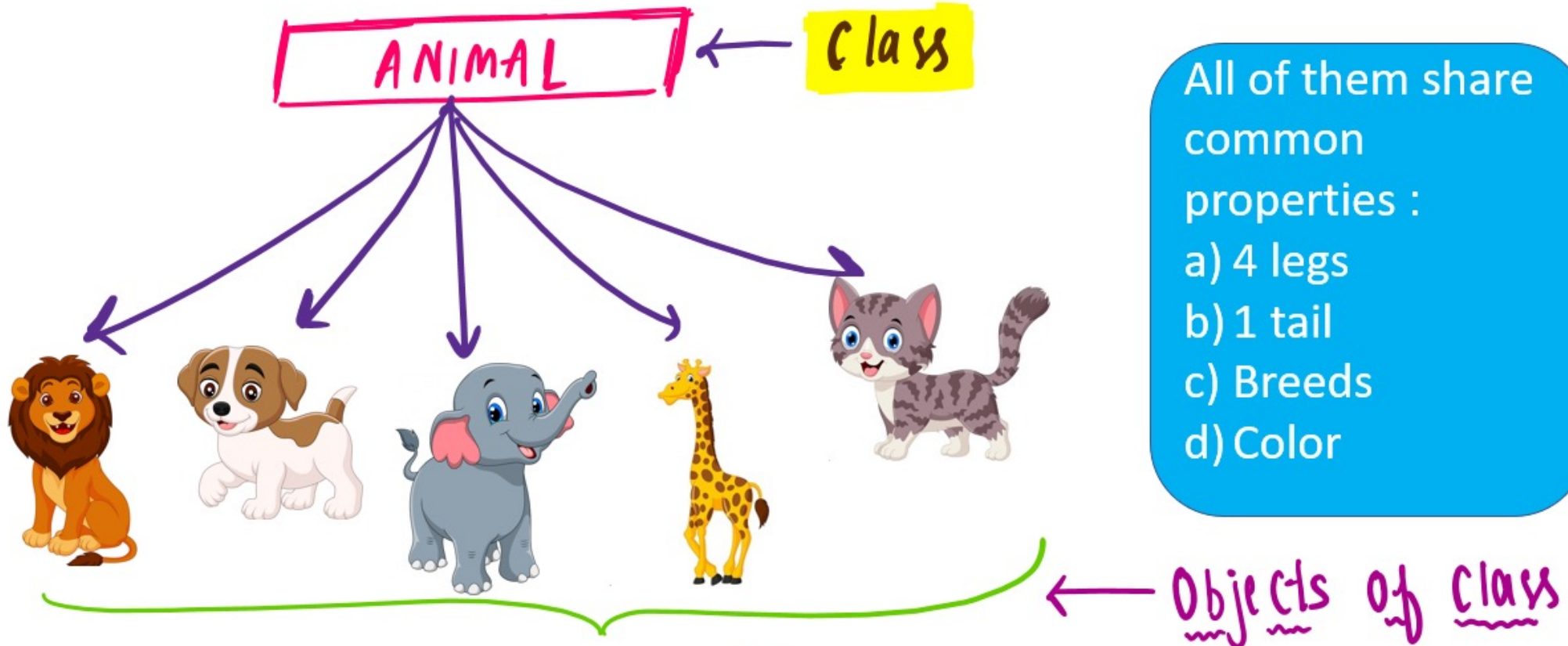


# OO Programming Concepts

Object-oriented programming (OOP) involves programming using objects. **An *object* represents an entity in the real world that can be distinctly identified.** For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects. **An object has a unique identity,** state, and behaviors. The **state** of an object consists of a set of **data fields** (also known as *properties*) with their current values. The **behavior** of an object is defined by a set of methods.

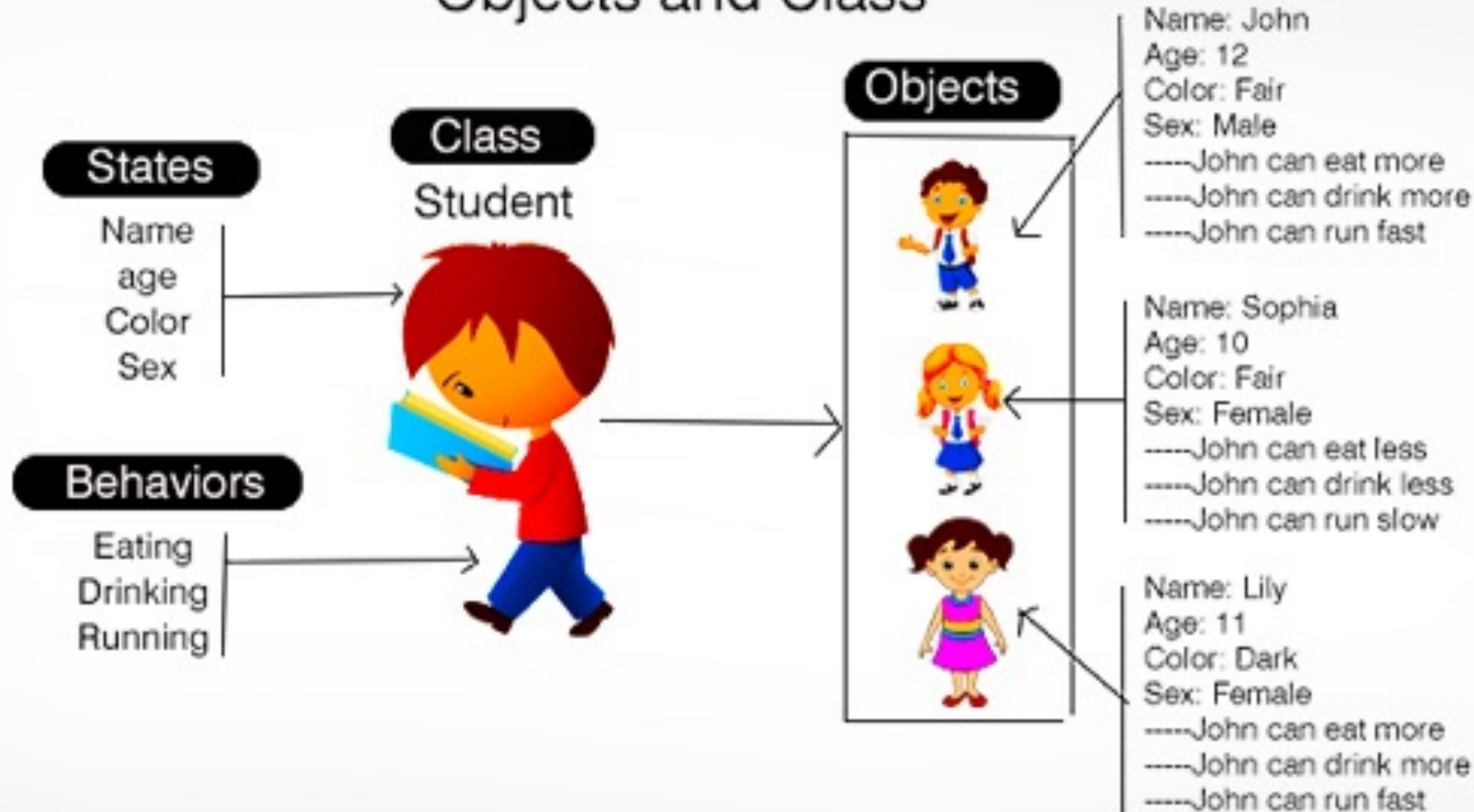








# Objects and Class



# Java Class & Objects

Class

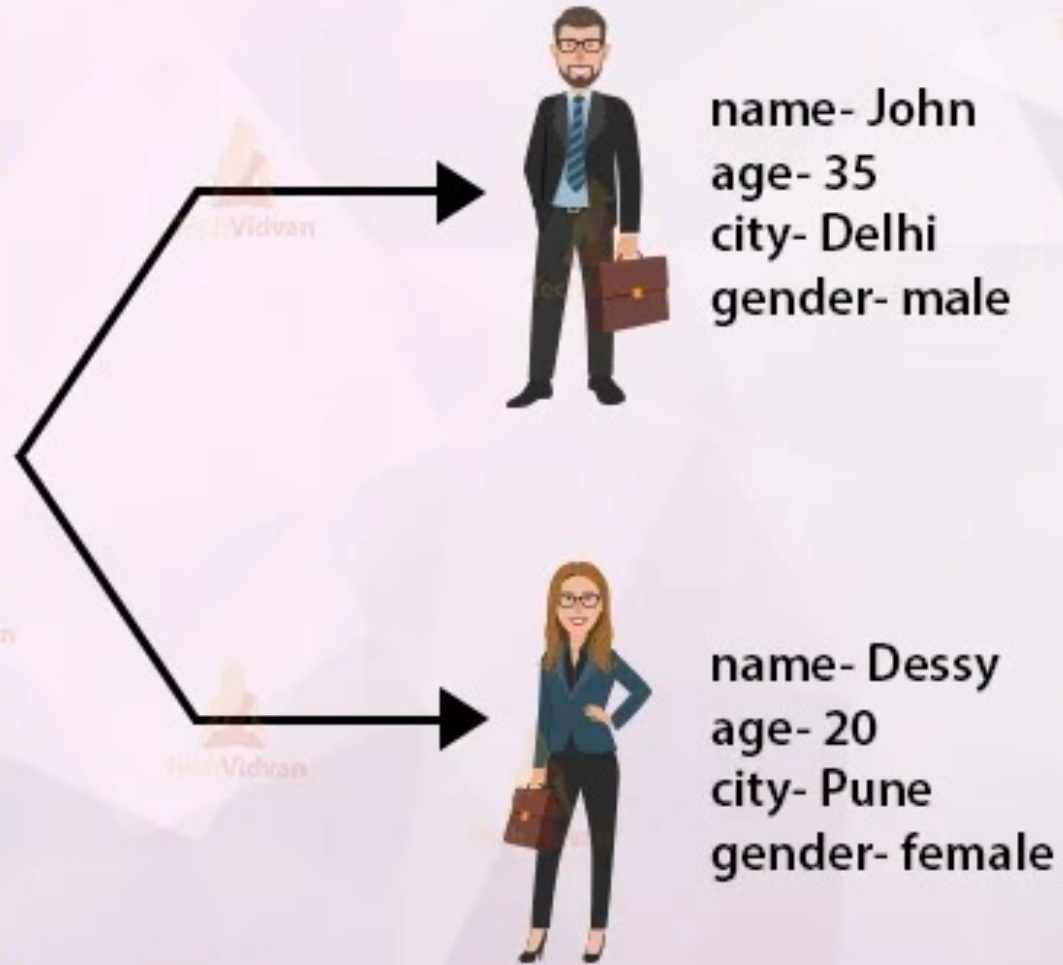
Person

Data  
Members

unique\_id  
name  
age  
city  
gender

Methods

eat()  
study()  
sleep()  
play()



# CLASS VERSUS OBJECT

## CLASS

A template for creating or instantiating objects within a program

Logical entity

Declared with the “class” keyword

A class does not get any memory when it is created

A class is declared once

## OBJECT

An instance of a class

Physical entity

Created using the “new” keyword

Objects get memory when they are created

Multiple objects are created using a class



# Classes

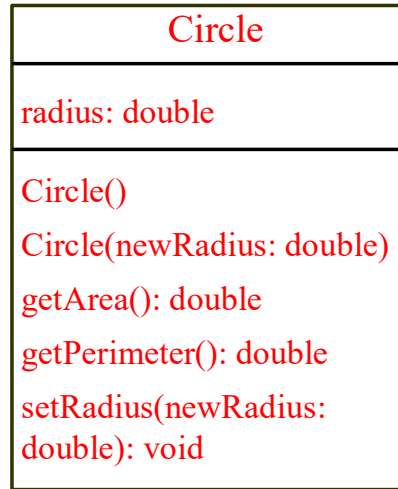
***Classes* are constructs that define objects of the same type. A Java class uses **variables to define data fields and methods to define behaviors**. Additionally, a class provides a special type of methods, **known as constructors**, which are invoked to construct objects from the class.**





# UML Class Diagram: Unified Modeling Language

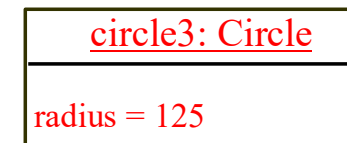
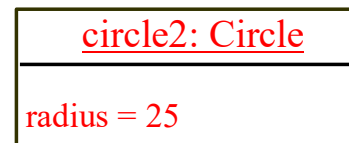
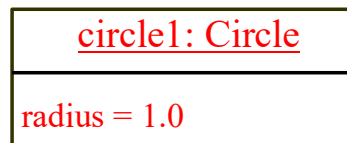
UML Class Diagram



← Class name

← Data fields

← Constructors and methods



← UML notation for objects



# Constructors

```
Circle() {  
}
```

```
Circle(double newRadius) {  
    radius = newRadius;  
}
```

Constructors are a special kind of methods that are invoked to construct objects.



# Constructors, cont.

A constructor with no parameters is referred to as a *no-arg constructor*.

- Constructors must have the **same name as the class itself**.
- Constructors **do not have a return type**—**not even void**.
- Constructors are invoked using the **new operator when an object is created**. Constructors play the role of initializing objects.



# Default Constructor

A class may be defined without constructors. In this case, a no-arg constructor with an empty body is implicitly defined in the class. This constructor, called *a default constructor*, is provided automatically **only if no constructors are explicitly defined in the class.**





# Default Value for a Data Field

The default value of a data field is null for a reference type, 0 for a numeric type, false for a boolean type, and '\u0000' for a char type. However, Java assigns no default value to a local variable inside a method.

```
public class Test {
    public static void main(String[] args) {
        Student student = new Student();
        System.out.println("name? " + student.name);
        System.out.println("age? " + student.age);
        System.out.println("isScienceMajor? " + student.isScienceMajor);
        System.out.println("gender? " + student.gender);
    }
}
```



# Example

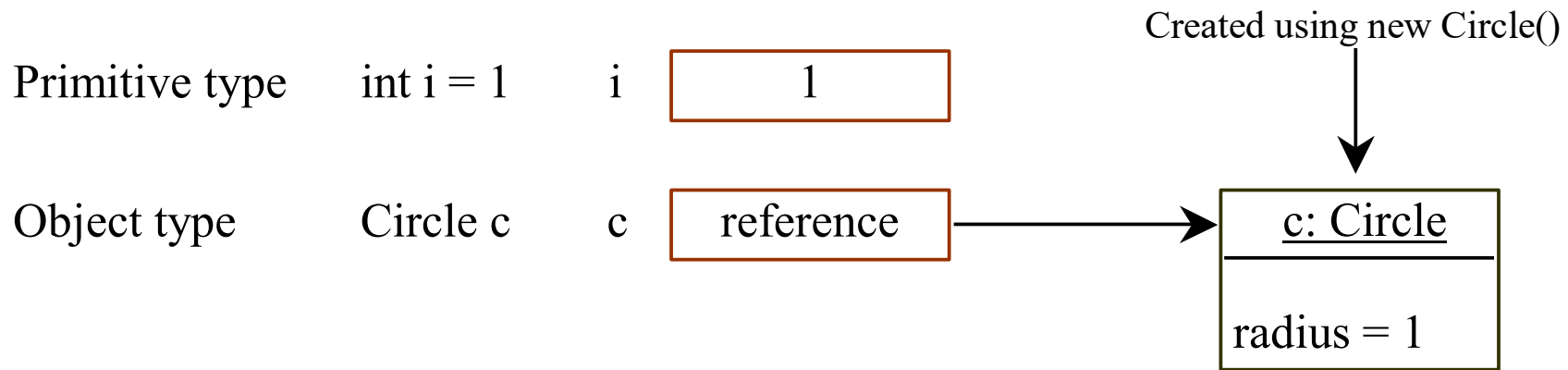
Java assigns no default value to a local variable inside a method.

```
public class Test {  
    public static void main(String[] args) {  
        int x; // x has no default value  
        String y; // y has no default value  
        System.out.println("x is " + x);  
        System.out.println("y is " + y);  
    }  
}
```

Compile error: variable not initialized



# Differences between Variables of Primitive Data Types and Object Types



# Garbage Collection,



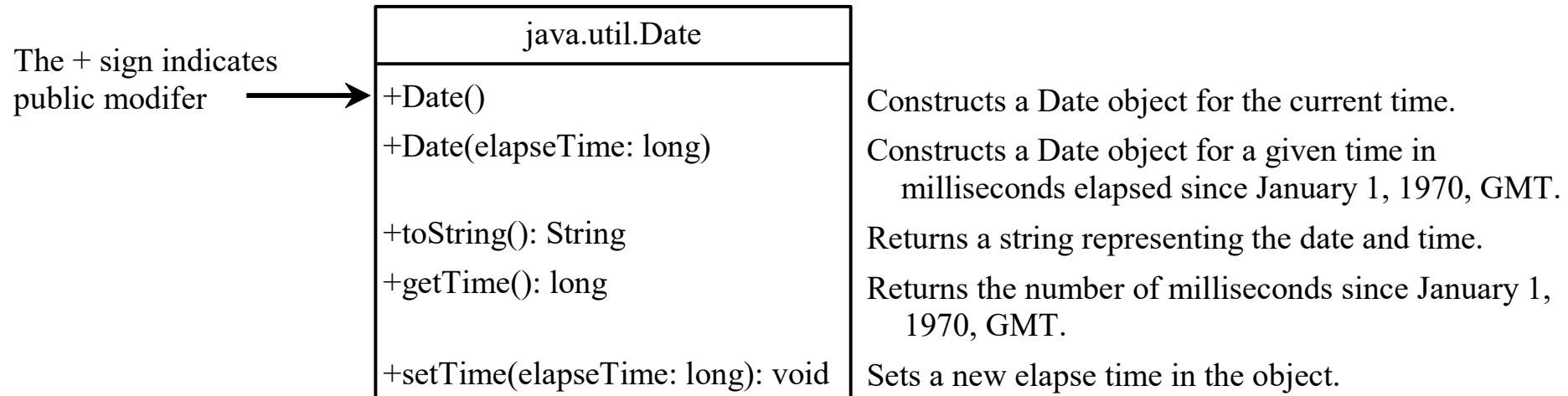
TIP: If you know that an object is **no longer needed**, you can **explicitly assign null to a reference variable** for the object. The **JVM** will automatically collect the space if the object is not referenced by any variable .





# The Date Class

Java provides a system-independent encapsulation of date and time in the java.util.Date class. You can use the Date class to create an instance for the current date and time and use its toString method to return the date and time as a string.



# The Date Class Example

For example, the following code

```
java.util.Date date = new java.util.Date();  
System.out.println(date.toString());
```

displays a string like Sun Mar 09 13:50:19 EST  
2003.



# The Random Class

You have used Math.random() to obtain a random double value between 0.0 and 1.0 (excluding 1.0). A more useful random number generator is provided in the java.util.Random class.

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



# The Random Class Example

If two Random objects have the same seed, they will generate identical sequences of numbers. For example, the following code creates two Random objects with the same seed 3.

```
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
    System.out.print(random1.nextInt(1000) + " ");
Random random2 = new Random(3);
System.out.print("\nFrom random2: ");
for (int i = 0; i < 10; i++)
    System.out.print(random2.nextInt(1000) + " ");
```

From random1: 734 660 210 581 128 202 549 564 459 961

From random2: 734 660 210 581 128 202 549 564 459 961





# Instance Variables, and Methods

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



# Static Variables, Constants, and Methods

Static variables are shared by all the instances of the class.

Static methods are not tied to a specific instance (object).

Static constants are final variables shared by all the instances of the class.

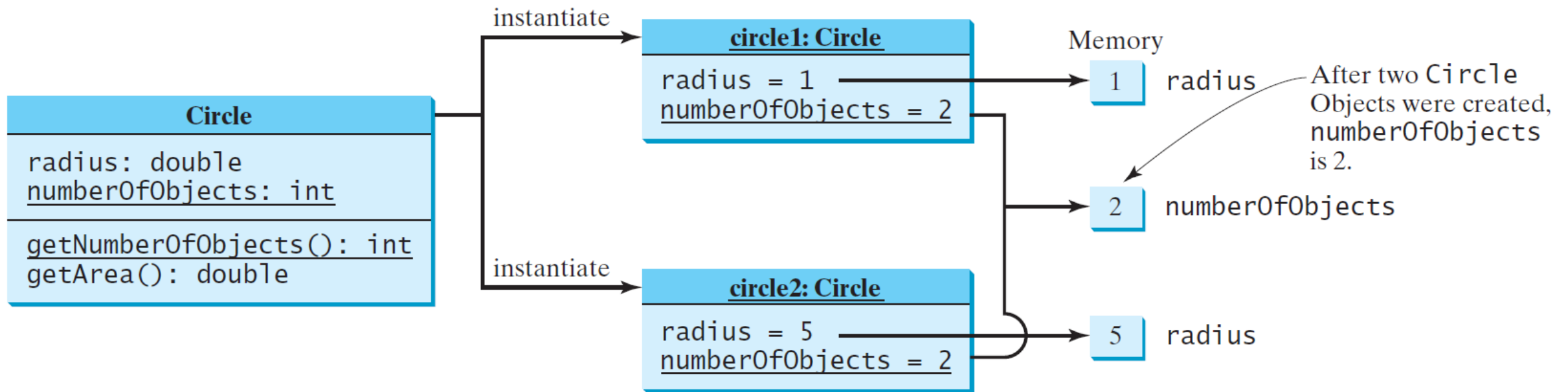
To declare static variables, constants, and methods, use the static modifier.



# Static Variables, Constants, and Methods, cont.

UML Notation:

underline: static variables or methods



```

public class CircleWithStaticMembers {
    /** The radius of the circle */
    double radius;

    /** The number of the objects created */
    static int numberOfObjects = 0;

    /** Construct a circle with radius 1 */
    CircleWithStaticMembers() {
        radius = 1.0;
        numberOfObjects++;
    }
}

```

```

/** Construct a circle with a specified radius */
CircleWithStaticMembers(double newRadius)
{
    radius = newRadius;
    numberOfObjects++;
}

/** Return numberOfObjects */
static int getNumberOfObjects() {
    return numberOfObjects;
}

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

```





## Static Variable

1. It is a variable which belongs to the class and not to the instance (object).
2. Static variables are initialized only once, at the start of the execution.  
Static variables will be initialized first, before the initialization of any instance variables.
3. A single copy to be shared by all instances of the class.
4. A static variable can be accessed directly by the class name and doesn't need any object.

Syntax : < class - name > . < static - variable - name >



## Static Method

1. It is a method which belongs to the class and not to the instance (object).
2. A static method can access only static data. It can not access non-static data (instance variables).
3. A static method can call only other static methods and can not call a non-static method from method inside.
4. A static method can be accessed directly by the class name and doesn't need any create an instance (object) to access it.

**Syntax :** < class - name>.<static - method - name>(..)

5. A static method cannot refer to “this” or “super” keywords in anyway.

**Note:** main method is static, since it must be accessible for an application to run, before any instantiation takes place.



```

public class Checkstatic {

public static void main(String[] args) {

Check c1=new Check();
Check c2=new Check();

    c2.setX(20);
    System.out.println(c1.getX());
}

    c1.setX(30);
    System.out.println(c2.getX());
}
}

```

```

class Check{
static int x;

Check(){
    x=10;
}

Check(int xvalue)
{
    x=xvalue;
}

public void setX(int xvalue){
    x=xvalue;
}

public int getX(){
    return x;
}}

```

```

public class Checkstatic {

public static void main(String[] args) {

Check c1=new Check();
Check c2=new Check();

    System.out.println(c1.x);
    c2.setX(20);
    System.out.println(c1.x);

    c1.setX(30);
    System.out.println(c1.getX());
}
}

```

```

class Check{
static int x;

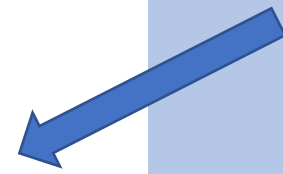
Check(){
    x=10;
}

Check(int xvalue)
{
    x=xvalue;
}

public void setX(int xvalue){
    x=xvalue;
}

public int getX(){
    return x;
}}

```



The static field Check.x should be accessed in a static way



```

public class Checkstatic {
    public static void main(String[] args) {

        System.out.println(Check.x);

        Check.setX(20);

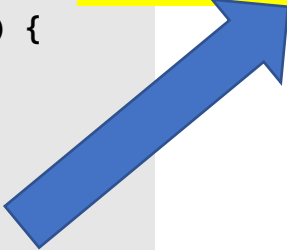
        System.out.println(Check.getX());
    }
}

class Check{
    static int x;

    Check(){
        x=10;
    }
    Check(int xvalue)
    {
        x=xvalue;
    }
    public static void setX(int xvalue){
        x=xvalue;
    }
    public static int getX(){
        return x;
    }
}

```

We accessed in a static way  
 Since no instance object created



```

public class Checkstatic {
    public static void main(String[] args) {

        Check c1=new Check();
        Check c2=new Check();

        /*System.out.println(c1.x); syntax error :using set to change value
        of x or get to return value of x
        c2.setX(20);
        //System.out.println(c1.x);syntax error

        System.out.println(c1.getX());

        c1.setX(30);
        System.out.println(c2.getX());
    }
}

class Check{
    private static int x;
    Check(){
        x =10;
    }
    Check(int xvalue)
    {
        x=xvalue;
    }
    public void setX(int xvalue){
        x=xvalue;
    }
    public int getX(){
        return x;
    }
}

```

# Visibility Modifiers and Accessor/Mutator Methods

**By default**, the class, variable, or method 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 methods **can be accessed only** by the **declaring class**.

The get and set methods are used to read and modify **private properties**.(variables)





```

package p1;

public class C1 {
    public int x;
    int y;
    private int z;

    public void m1() {
    }
    void m2() {
    }
    private void m3() {
    }
}

```

```

package p1;

public class C2 {
    void aMethod() {
        C1 o = new C1();
        can access o.x;
        can access o.y;
        cannot access o.z;

        can invoke o.m1();
        can invoke o.m2();
        cannot invoke o.m3();
    }
}

```

```

package p2;

public class C3 {
    void aMethod() {
        C1 o = new C1();
        can access o.x;
        cannot access o.y;
        cannot access o.z;

        can invoke o.m1();
        cannot invoke o.m2();
        cannot invoke o.m3();
    }
}

```

The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.



Most Restrictive



Least Restrictive

Access Modifiers ->	private	Default/no-access	protected	public
Inside class	Y	Y	Y	Y
Same Package Class	N	Y	Y	Y
Same Package Sub-Class	N	Y	Y	Y
Other Package Class	N	N	N	Y
Other Package Sub-Class	N	N	Y	Y

Same rules apply for inner classes too, they are also treated as outer class properties

```
package p1;

class C1 {
    ...
}
```

```
package p1;

public class C2 {
    can access C1
}
```

```
package p2;

public class C3 {
    cannot access C1;
    can access C2;
}
```

The default modifier on a **class restricts access to within a package**, and the **public modifier enables unrestricted access**.

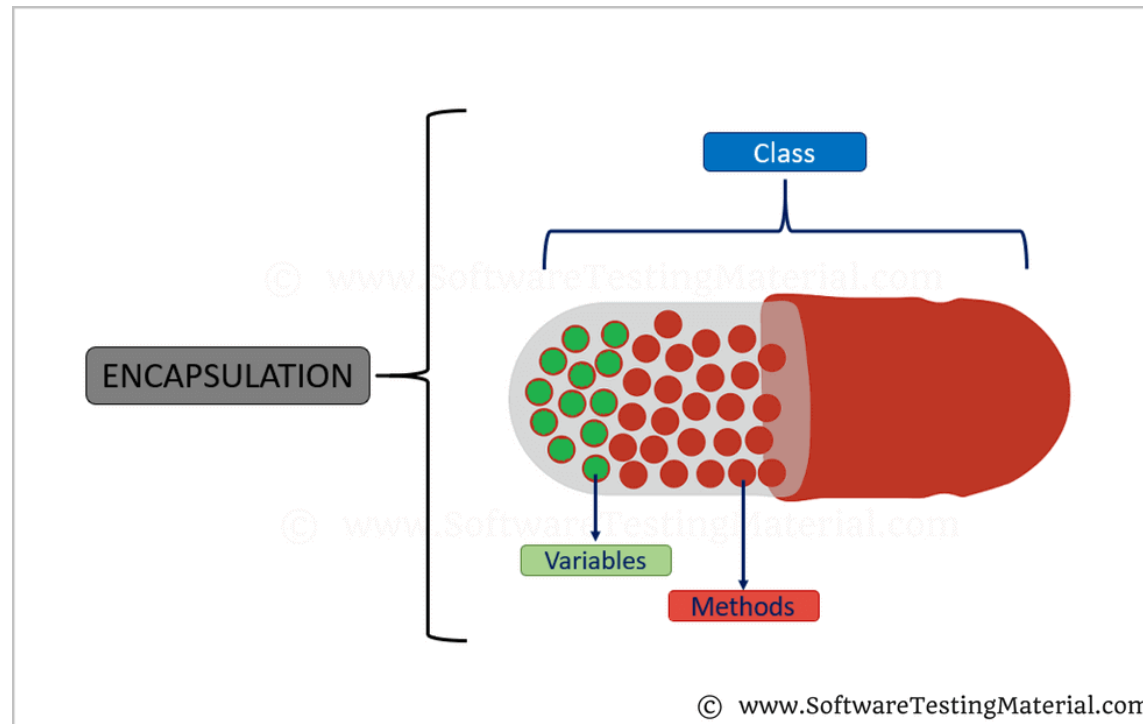
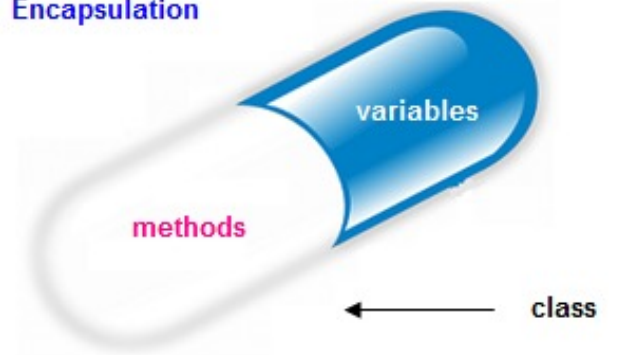


# Why Data Fields Should Be private?

To protect data.

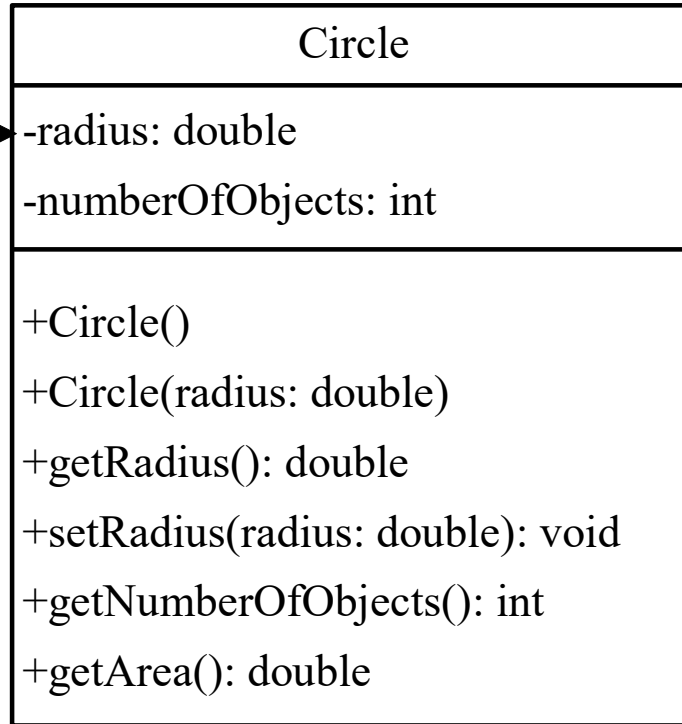
To make code easy to maintain.

Encapsulation



# Example of Data Field Encapsulation

The - sign indicates  
private modifier



CircleWithPrivateDataFields

TestCircleWithPrivateDataFields

Run





# Passing Objects to Methods

```
public class TestPassObject {
    /** Main method */
    public static void main(String[] args) {
        // Create a Circle object with radius 1
        CircleWithPrivateDataFields myCircle = new CircleWithPrivateDataFields(1);
        // Print areas for radius 1, 2, 3, 4, and 5.
        int n = 5;
        printAreas(myCircle, n);

        // See myCircle.radius and times
        System.out.println("\n" + "Radius is " + myCircle.getRadius());
        System.out.println("n is " + n);
    }
    /** Print a table of areas for radius */
    public static void printAreas(CircleWithPrivateDataFields c, int times) {
        System.out.println("Radius \t\tArea");
        while (times >= 1) {
            System.out.println(c.getRadius() + "\t\t" + c.getArea());
            c.setRadius(c.getRadius() + 1);
            times--;
        }
    }
}
liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum
```



# Passing Objects to Methods

- ❑ Passing by value for primitive type value (the value is passed to the parameter)
- ❑ Passing by value for reference type value (the value is the reference to the object)

Radius	Area
1.0	3.141592653589793
2.0	12.566370614359172
3.0	28.274333882308138
4.0	50.26548245743669
5.0	78.53981633974483
Radius is 6.0	n is 5



# Array of Objects

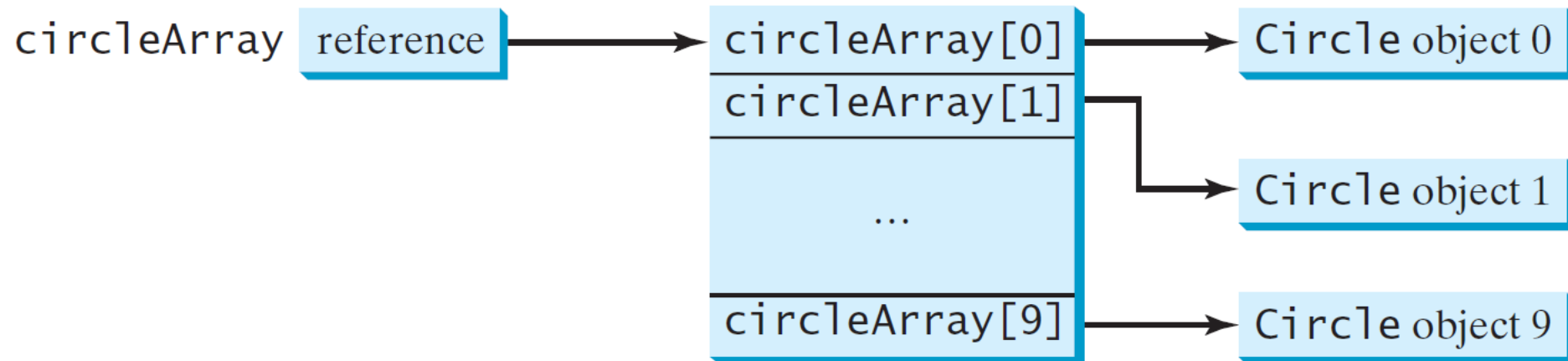
```
Circle[] circleArray = new Circle[10];
```

An array of objects is actually an *array of reference variables*. So invoking `circleArray[1].getArea()` involves two levels of referencing as shown in the next figure. `circleArray` references to the entire array. `circleArray[1]` references to a `Circle` object.



# Array of Objects, cont.

```
Circle[] circleArray = new Circle[10];
```



# Immutable( Cannot change) Objects and Classes

If the contents of an object cannot be changed once the object is created, the object is called an immutable object and its class is called an immutable class. If you delete the set method in the Circle class in Listing 8.10, the class would be immutable because radius is private and cannot be changed without a set method.

A class with all private data fields and without mutators is not necessarily immutable. For example, the following class Student has all private data fields and no mutators, but it is mutable.





# Example

```
public class Student {
    private int id;
    private BirthDate birthDate;

    public Student(int ssn,
        int year, int month, int day) {
        id = ssn;
        birthDate = new BirthDate(year, month, day);
    }

    public int getId() {
        return id;
    }

    public BirthDate getBirthDate() {
        return birthDate;
    }
}
```

```
public class BirthDate {
    private int year;
    private int month;
    private int day;

    public BirthDate(int newYear,
        int newMonth, int newDay) {
        year = newYear;
        month = newMonth;
        day = newDay;
    }

    public void setYear(int newYear) {
        year = newYear;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Student student = new Student(111223333, 1970, 5, 3);
        BirthDate date = student.getBirthDate();
        date.setYear(2010); // Now the student birth year is changed!
    }
}
```



- ❑ The this keyword is the name of a reference that refers to an object itself. One common use of the this keyword is reference a class's *hidden data fields*.
- ❑ Another common use of the this keyword to enable a constructor to invoke another constructor of the same class.

Remember : . A static method cannot refer to “this” or “super” keywords in anyway.



# Calling Overloaded Constructor

```
public class Circle {  
    private double radius;
```

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

→ this must be explicitly used to reference the data field radius of the object being constructed

```
    public Circle() {  
        this(1.0);  
    }
```

→ this is used to invoke another constructor

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

↓  
Every instance variable belongs to an instance represented by this, which is normally omitted

