Chapter 9
Objects and Classes

Dr. Asem Kitana
Dr. Abdallah Karakra

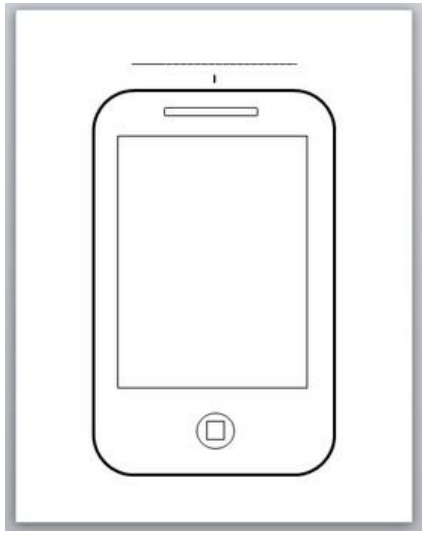
Before Begin

- Everything in the real world is object

Objects of the same Kind

Objects of the same type are defined using a common class

Class Smartphone



Smartphone

iphone

nokia

samsung

Objects of the same Kind

Objects of the same type are defined using a common class

Class Animal

Objects of the same Kind

Objects of the same type are defined using a common class

Vehicle Class

Vechicle class

Object

Object = ID + State + Behavior

- *Identity*: is the variable name when instantiated.
- *State* of an object: Consists of a set of *data fields* (also known as *properties* , *data value, attributes*)
- *Behavior* of an object: is defined by a set of methods (functions).

Objects of the same Kind

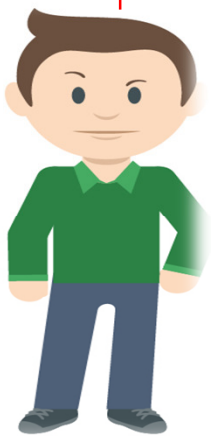
Person

Name:
Age:
….
move();
run();
…

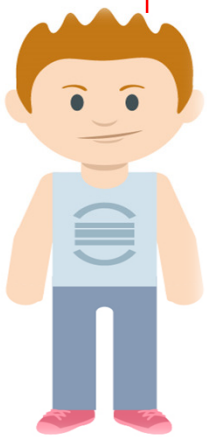
class

objects

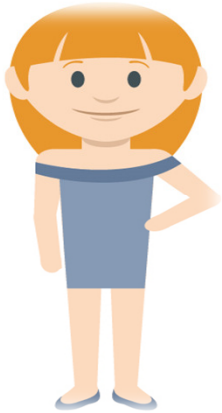
Name: Bob
Age: 12
….



Name: Sam
Age: 13
….



Name: Jack
Age: 15
….



Name: Alice
Age: 10
….

Name: Sandy
Age: 11
….

Examples

```
class Person{

    // attributes
    int age;
   String firstName;
   String lastName;
    …
  // methods
   void speak(){
   }
   void listen(){
   }
   ……

}
```

```
class Vehicle{

    // attributes
   String name;
   int model;
   double speed;
    …
  // methods
   void changeDirection(){
   }
   void move(){
   }
   void stop(){
   }
   ……
}
```

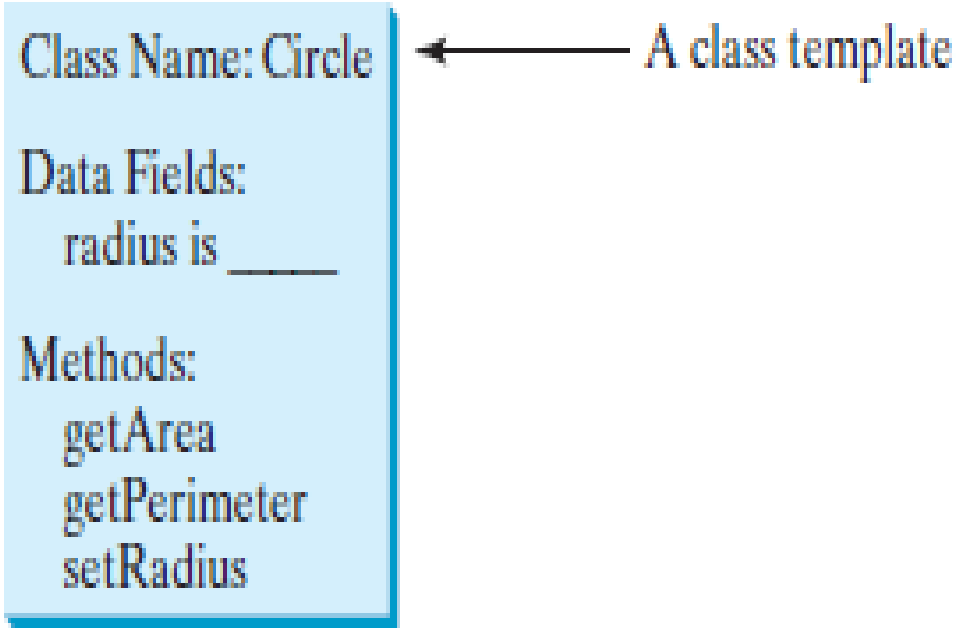
Class & Object

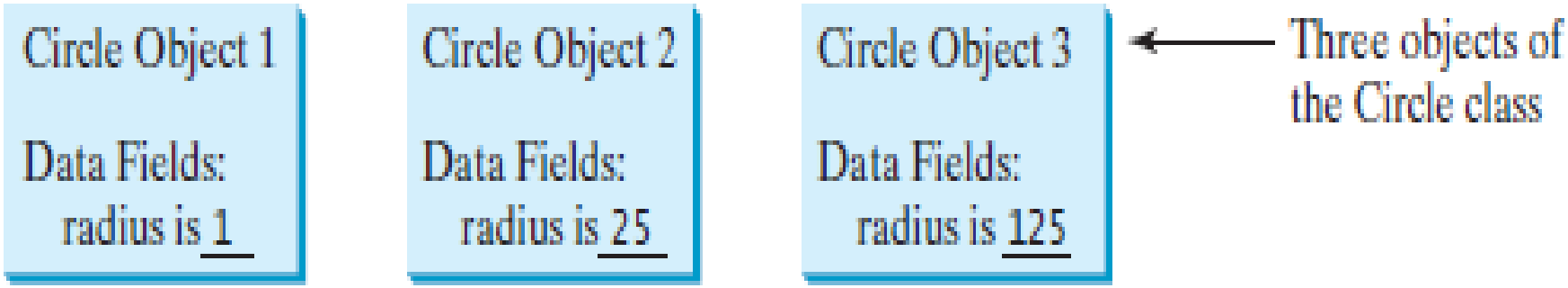
- A *class* is a template, or *contract* that defines what an object's data fields and methods will be.

- An object is an instance of a class. You can create many instances of a class.

- Creating an instance is referred to as *instantiation*

Class & Object





Examples : State & Behavior

Person class

Attributes of Person :

1.Name of Person
2.Gender
3.Skin Color
4.Hair Color

Behaviors of Person:

1. Talking
2. Walking
3. Eating

Vehicle class

Attributes of Vehicle:

1. Color
2. name
3. Model
4. Speed
5. Mileage

Behaviors of Vehicle

1. Turn left
2. Turn right
3. Press break

Examples : State & Behavior

Animal class

Attributes of Animal :

1.Color
2. name
3. height
4. age

Behaviors of Animal :

1. eating
2. Sleeping
3. ..

OO Programming Features

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstrtaction

Later !

OO Programming Concepts

Object-oriented programming (OOP) involves programming using objects. An *object* represents an entity in the real world that can be uniquely 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 *attributes*) with their current values.
- The *behavior* of an object is defined by a set of methods.

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.
- A class provides a special type of methods, known as constructors, which are invoked to construct objects from the class.

Examples

```
class Person{

    // Data fields
    int age;
    String firstName;
    String lastName;
    ...
   // Behaviors
    void speak(){
    }
    void listen(){
    }
    ......

}
```

Variables to define data fields

Methods to define behaviors

Check Point

1) The relationship between a class and an object is best described as
A) objects are the instance data of classes
B) objects and classes are the same thing
C) classes are programs while objects are variables
D) classes are instances of objects
E) objects are instances of classes

2) The behavior of an object is defined by the object's
A) constructor
B) instance data
C) methods
D) visibility modifiers
E) all of the above

Check Point

Types in Java are divided into two categories. The primitive types are `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`. All other types are _____ types.

A) `static`

B) reference

C) declared

D) source

Classes

```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * 3.14159;
  }
}
```

Data field

Constructors

Method

Constructors

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

A constructor can perform any action, but they are designed to perform initializing actions, such as initializing the data fields of objects.

Constructors

```
Circle() {          //no-arg constructor.

}

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

Example – Part1

```
1  public class TestSimpleCircle {
2   /** Main method */
3   public static void main(String[] args) {
4   // Create a circle with radius 1
5   SimpleCircle circle1 = new SimpleCircle();
6   System.out.println("The area of the circle of radius "
7   + circle1.radius + " is " + circle1.getArea());
8
9   // Create a circle with radius 25
10  SimpleCircle circle2 = new SimpleCircle(25);
11  System.out.println("The area of the circle of radius "
12  + circle2.radius + " is " + circle2.getArea());
13
14  // Create a circle with radius 125
15  SimpleCircle circle3 = new SimpleCircle(125);
16  System.out.println("The area of the circle of radius "
17  + circle3.radius + " is " + circle3.getArea());
18  System.out.println("The perimeter of the circle of radius "
19          + circle3.radius + " is " + circle3.getPerimeter());
20 }}
21  // Define the circle class with two constructors
22  class SimpleCircle {
23  double radius;
```

Example – Part2

```
21  // Define the circle class with two constructors
22  class SimpleCircle {
23  double radius;
24
25  /** Construct a circle with radius 1 */
26  SimpleCircle() {
27  radius = 1;
28  }
29  /** Construct a circle with a specified radius */
30  SimpleCircle(double newRadius) {
31  radius = newRadius;
32  }
33  /** Return the area of this circle */
34  double getArea() {
35  return radius * radius * Math.PI;
36  }
37 //** Return the perimeter of this circle */
38  double getPerimeter() {
39  return 2 * radius * Math.PI;
40  }}
```

Example – Part3

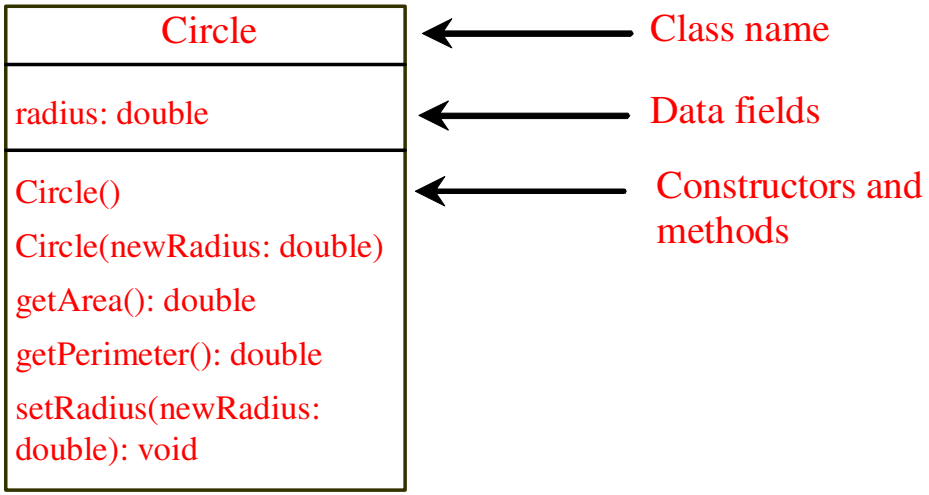
Problems Javadoc Declaration Console X

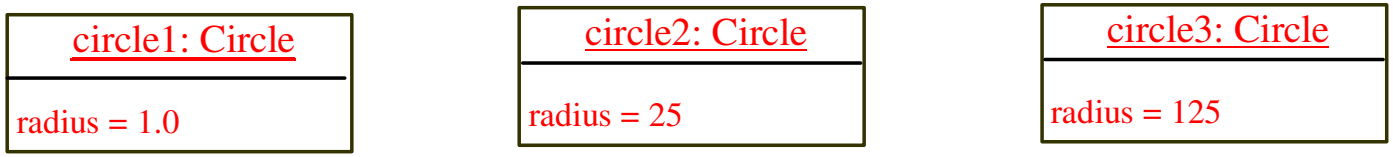
```
<terminated> TestSimpleCircle [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (Dec 7, 2022, 12:10:42 PM -
The area of the circle of radius 1.0 is 3.141592653589793
The area of the circle of radius 25.0 is 1963.4954084936207
The area of the circle of radius 125.0 is 49087.385212340516
The perimeter of the circle of radius 125.0 is 785.3981633974482
```

UML Class Diagram

Unified Modeling Language (UML) notation

UML Class Diagram





Creating Objects Using Constructors

```
new ClassName();//create object of type ClassName
```

Example:

```
new Circle();//create object of type Circle

new Circle(5.0);
```

Creating Objects

```
class Rectangle {
  double width;
  double height;

  Rectangle() {
    width = 4.0;
    height = 5.3;
  }

  double area() {
    return width * height;
  }
}
```

Rectangle rectObj = new Rectangle();

```
class Rectangle {
  double width;
  double height;

  Rectangle(double w, double h) {
    width = w;
    height = h;
  }

  double area() {
    return width * height;
  }
}
```

Rectangle rectObj = new Rectangle(4.0,5.2);

Creating Objects

```
class Rectangle {
  double width;
  double height;

 Rectangle() {
    width = 1.0;
    height = 5.4;
  }

  Rectangle(double w, double h) {
    width = w;
    height = h;
  }

  double area() {
    return width * height;
  }
}
```

```
Rectangle rectObj = new Rectangle();
```

```
Rectangle rectObj2 = new Rectangle(4.0,5.0);
```

rectObj
width=1.0
heigh=5.4

rectObj2
width=4.0
heigh=5.0

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(such as initializing the data fields of objects).

Default Constructor

A class may be defined without constructors. In this case, a **public 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 Constructor

```
class Rectangle {
  double width;
  double height;

  double area() {
    return width * height;
  }

}
```

```
Rectangle mybox1 = new Rectangle();
```

Correct for both

No-arg constructor

```
class Rectangle {
  double width;
  double height;

  Rectangle() {
    width = 10;
    height = 10;
  }

  double area() {
    return width * height;
  }
}
```

Declaring/Creating Objects in a Single Step

```
ClassName objectRefVar = new ClassName();
```

Example:

Assign object reference

Create an object

```
Circle myCircle = new Circle();
```

Accessing Object's Members

Referencing the object's data:

`objectRefVar.data`

e.g., `myCircle.radius`

Invoking the object's method:

`objectRefVar.methodName(arguments)`

e.g., `myCircle.getArea()`

```
class Circle {
  /** The radius of this circle */
  double radius = 1;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  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 new radius for this circle */
  void setRadius(double newRadius) {
    radius = newRadius;
  }
}
```

Constructors Example 1.1

```
1  public class Student {
2    String firstName;
3    String lastName;
4    int age;
5
6    public static void main(String args[]) {
7
8   Student stud = new Student();
9   System.out.println(stud.firstName + " " + stud.lastName + " "
10  + stud.age);
11
12  Student stud2 = new Student("Mahmoud", "Hani", 26);
13  System.out.println(stud2.firstName + " " + stud2.lastName + " "
14  + stud2.age);
15   }
16
17   Student() {
18       firstName = "Naser";
19       lastName = "Jaber";
20       age = 30;
```

Constructors Example 1.2

```
17    Student() {
18        firstName = "Naser";
19        lastName = "Jaber";
20        age = 30;
21        }
22
23    Student(String f, String l, int a) {
24            firstName = f;
25            lastName = l;
26            age = a;
27        }
28 }
```

Problems | Javadoc | Declaration | Console

```
<terminated> Student (2) [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\
Naser Jaber 30
Mahmoud Hani 26
```

Constructors Example 2

```java
1  public class Student {
2    String firstName;
3    String lastName;
4    int age;
5
6    public static void main(String args[]) {
7
8        Student stud = new Student();
9
10       stud.firstName = "Ahmad";
11       stud.lastName = "Ali";
12       stud.age = 22;
13
14    System.out.println(stud.firstName + " " + stud.lastName + " "
15                + stud.age);
16   }
17 }
```

Problems Javadoc Declaration Console

```
<terminated> Student (1) [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (Dec 7, 2022, 12:28:34 PM – 12:28:34 P
Ahmad Ali 22
```

Constructors Example 3.1

```
1 public class Rectangle {
2     double width;
3     double hight;
4 public static void main(String[] args) {
5  // Create a rectangle with width 1.0 and hight 5.4
6  Rectangle rect1 = new Rectangle();
7  System.out.println("The area of the regtangle of"
8  + " width " + rect1.width + " and hight " + rect1.hight +
9  " is " + rect1.area());
10
11 //Create a rectangle with width 7.2 and hight 11.5
12 Rectangle rect2 = new Rectangle(7.2, 11.5);
13 System.out.println("The area of the regtangle of"
14 + " width " + rect2.width + " and hight " + rect2.hight +
15 " is " + rect2.area());
16  }
```

Constructors Example 3.2

```
17
18  Rectangle() {
19  width = 1.0;
20  hight = 5.4;
21  }
22
23  Rectangle(double w, double h) {
24  width = w;
25  hight = h;
26  }
27
28  /** Return the area of the rectangle */
29  double area() {
30  return width*hight;
31  }}
```

Problems | Javadoc | Declaration | Console

```
<terminated> Rectangle [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (Dec 7, 2022, 12:33:42 PM
The area of the regtangle of width 1.0 and hight 5.4 is 5.4
The area of the regtangle of width 7.2 and hight 11.5 is 82.8
```

Constructors Example 4

```java
public class Rectangle {
    double width;
    double hight;
 public static void main(String[] args) {

 Rectangle rect1 = new Rectangle();
 rect1.width = 2.0;
 rect1.hight = 6.0;

 System.out.println("The area of the regtangle of"
 + " width " + rect1.width + " and hight " + rect1.hight +
 " is " + rect1.area());
 }

 double area() {
 return width*hight;
 }}
```

Problems | Javadoc | Declaration | Console

```
<terminated> Rectangle (1) [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (Dec 7, 2022, 12:36:23 PM
The area of the regtangle of width 2.0 and hight 6.0 is 12.0
```

Check Point

1) Which of the following reserved words in Java is used to create an instance of a class?

A) new

B) public or private, either could be used

C) import

D) class

E) public

Check Point

2) A class constructor usually defines

A) the number of methods in the class

B) how an object is interfaced

C) the number of instance data in the class

D) how an object is initialized

E) if the instance data are accessible outside of the object directly

Check Point

What are the differences between constructors and methods?

Constructors are special kinds of methods that are called when creating an object using the **new operator**. Constructors **do not have a return type-not even void**.

When will a class have a default constructor?

A class has a default constructor only if the class does not define any constructor.

Check Point

What is wrong in the following code?

```
class Test {
  public static void main(String[] args) {
    A a = new A();
    a.print();
  }
}

class A {
  String s;

  A(String newS) {
    s = newS;
  }

  public void print() {
    System.out.print(s);
  }
}
```

The program does not compile because new A() is used in class Test, but class A **does not have a no-args constructor**.

Reference Data Fields

The data fields can be of reference types. For example, the following Student class contains a data field name of the String type.

```
public class Student {
  String name; // name has default value null
  int age; // age has default value 0
  boolean isScienceMajor; // isScienceMajor has default value false
  char gender; // gender has default value '\u0000'
}
```

The null Value

If a data field of a reference type does not reference any object, the data field holds a special literal value, null.

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
- '\u0000' for a char type.

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

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

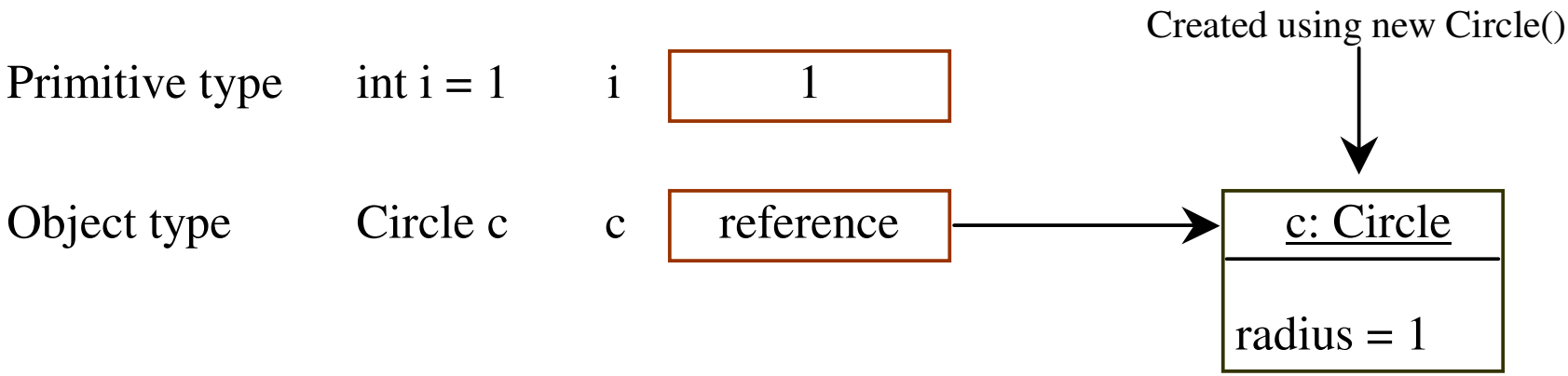
Check Point

What is the output of the following code?

```
public class A {
  boolean x;

  public static void main(String[] args) {
    A a = new A();
    System.out.println(a.x);
  }
}
```

false

Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types

Primitive type assignment i = j

Before:

i 1

j 2

After:

i 2

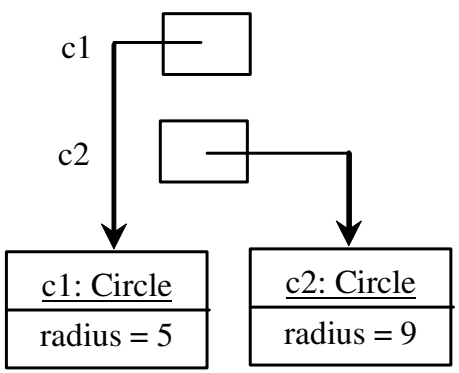
j 2

```
int i=1;
int j=2;
i=j;
```

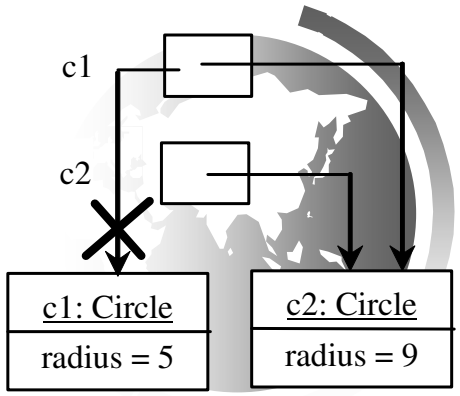
Object type assignment c1 = c2

```
Circle c1= new Circle (5);
Circle c2= new Circle (9);
 c1=c2;
```

Before:



After:



Garbage Collection

As shown in the previous figure, after the assignment statement c1 = c2, c1 points to the same object referenced by c2. The object previously referenced by c1 is no longer referenced. This object is known as garbage. Garbage is automatically collected by JVM.

Example: Output

```
class Mystery{

    int x;

    Mystery(int newX){

        x= newX;
    }
    public static void main(String [] args){

        Mystery obj1= new Mystery(1);
        Mystery obj2= new Mystery(3);
        System.out.println("obj1.x = " + obj1.x + "  obj2.x = " + obj2.x);
        obj2=obj1;
        System.out.println("obj1.x = " + obj1.x + "  obj2.x = " + obj2.x);
    }

}
```

```
obj1.x = 1  obj2.x = 3
obj1.x = 1  obj2.x = 1
```

Instance Variables, and Methods

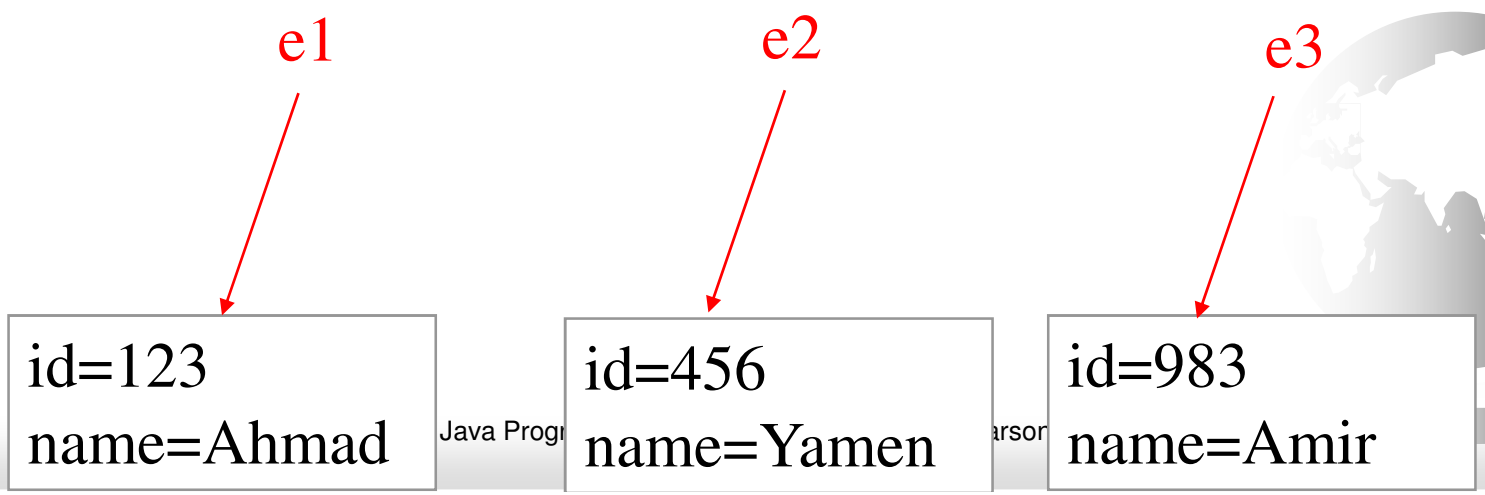
Instance variables(**non-static variables**) belong to a specific instance.

Instance methods(**non-static methods**) are invoked by an instance of the class.

Example

```
class Employee{

    int id;
    String name;

    Employee (int newId, String newName){
            id=newId;
            name=newName;
    }
    public static void main (String [] args){

            Employee e1= new Employee(123,"Ahamd");
            Employee e2= new Employee(456,"Yamen");
            Employee e3= new Employee(983,"Amir");

    }

}
```

Instance variables



Static Variables, Constants, and Methods

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

Static methods are **not tied to a specific object**.

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

Static Variables, Constants, and Methods, cont.

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

Static Variable

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

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

```java
class TestStatic {

   static int count=6;
   int var;

   public TestStatic(){
     count+=1;

   }

  public static void main(String args[]){
    TestStatic ts1= new TestStatic();
    ts1.var=3;
    TestStatic ts2= new TestStatic();
    ts2.var=5;
    TestStatic ts3= new TestStatic();
    ts3.var=7;
    System.out.println("count= "+ ts1.count + ", var= "+ ts1.var);
    System.out.println("count= "+ ts2.count + ", var= "+ ts2.var);
    System.out.println("count= "+ ts3.count + ", var= "+ ts3.var);
  }
}
```

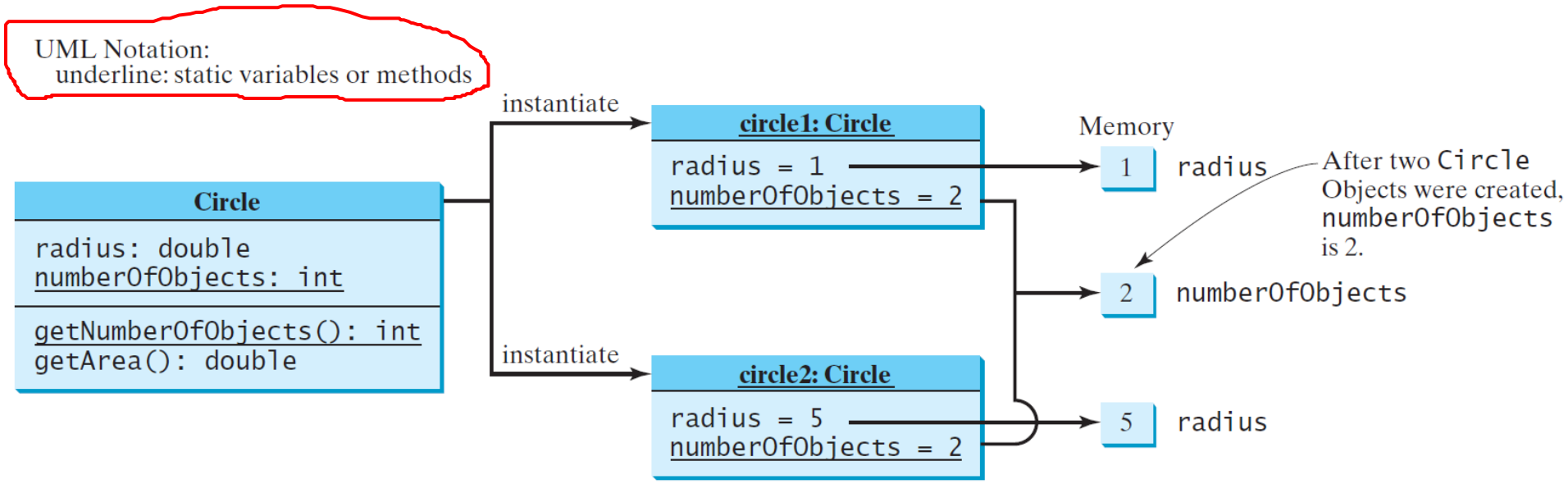
```
count= 9, var= 3
count= 9, var= 5
count= 9, var= 7
```

Static Method

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

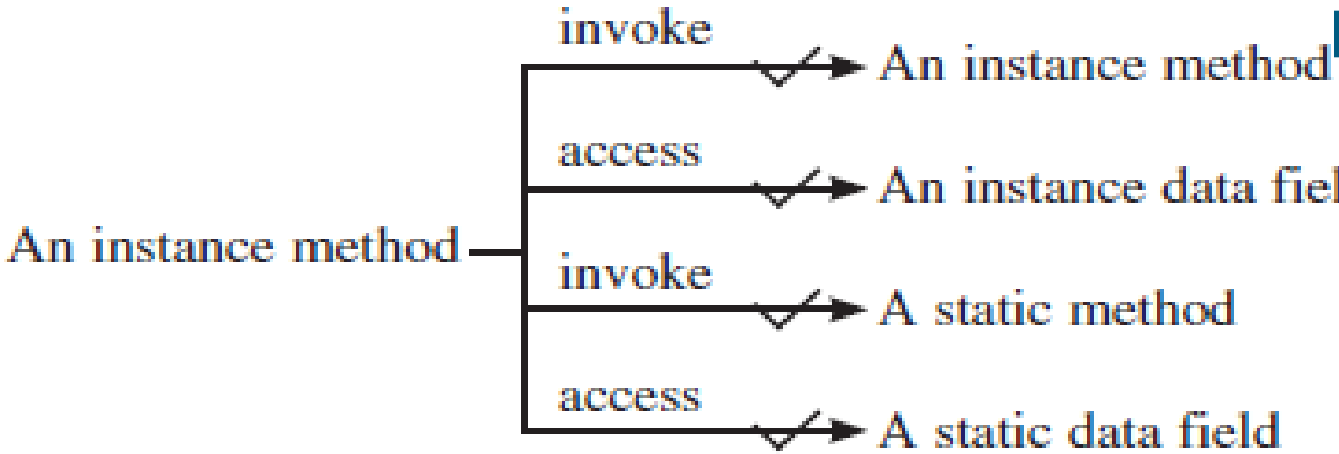
Syntax : ***<class-name>.<static-method-name>***

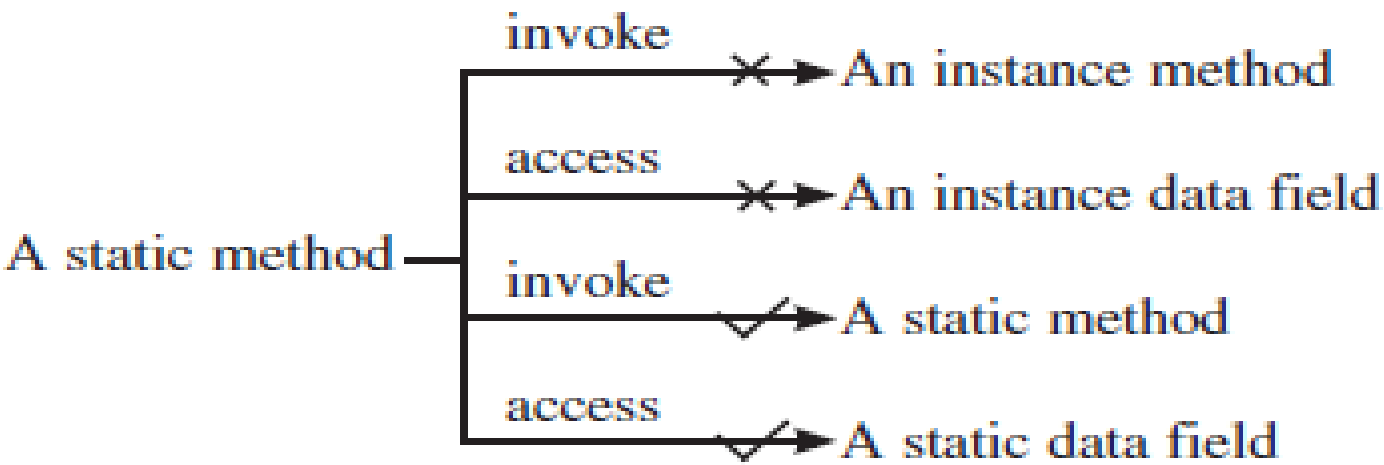
Static Variables, Constants, and Methods, cont.



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

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

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

Instance and Static and Methods



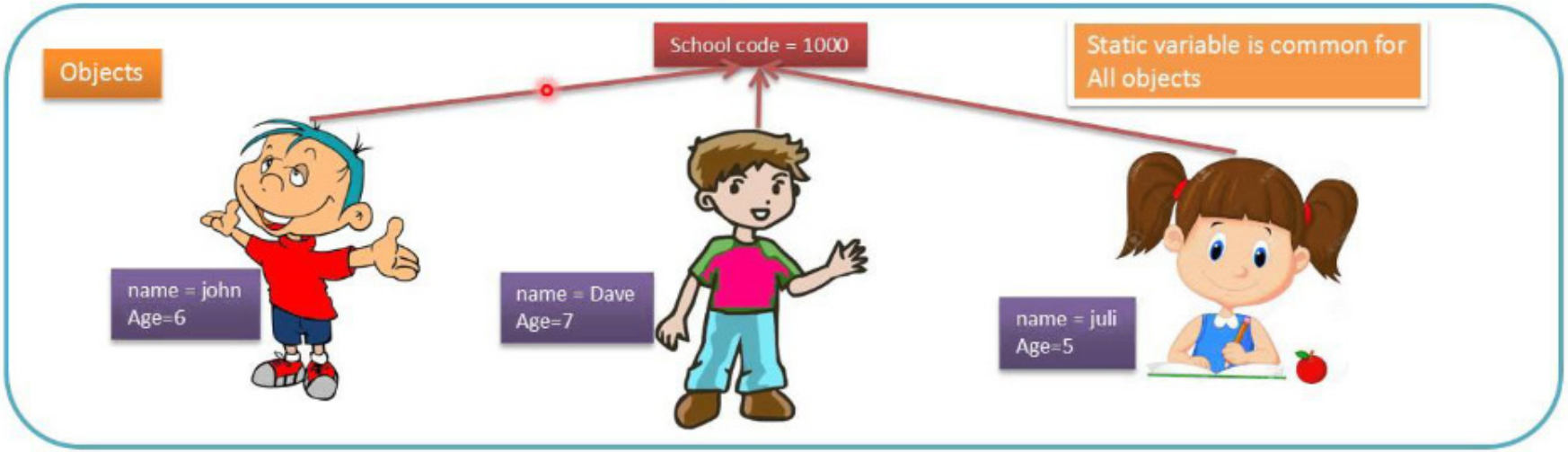


Example

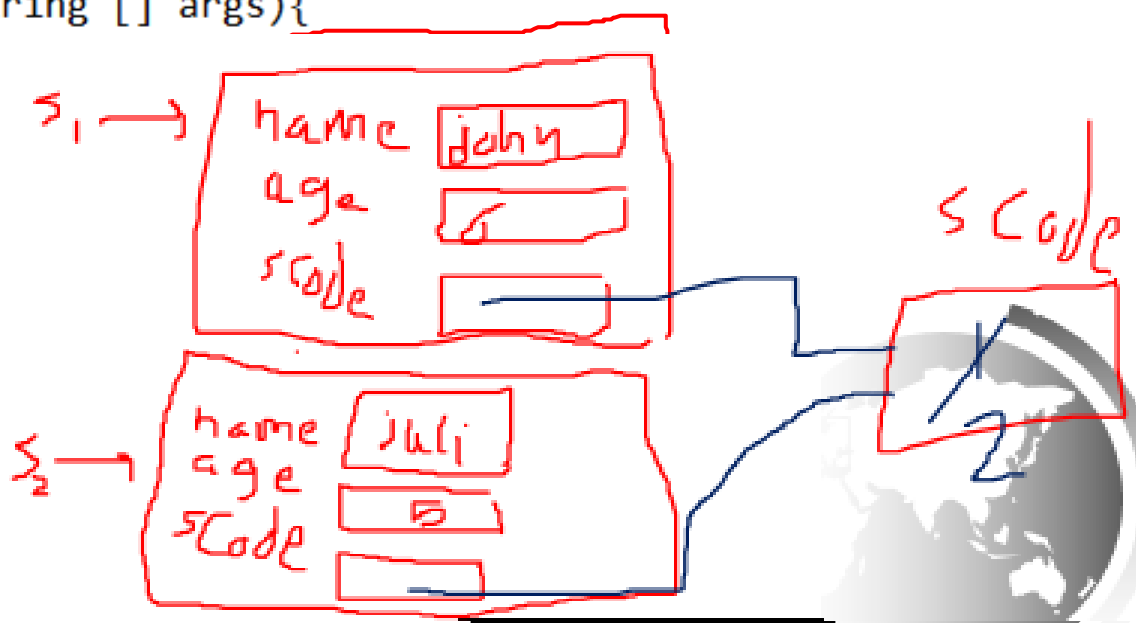
Create a class students based on the following UML.

Student
name: String age : int <u>schoolCode:int</u>
Student() printInfo():void

Increase the school code
Print the student information



```
class Student{
     String name;
     int age;
     static int schoolCode;

     Student (){
         schoolCode++;
     }

     void printInfo(){
        System.out.println (name + ", " + age + ", " + schoolCode );
     }

     public static void main(String [] args){

      Student s1=new Student();
      Student s2=new Student();
      s1.name="john";
      s1.age=6;

      s2.name="juli";
      s2.age=5;

      s1.printInfo();
      s2.printInfo();

    }

}
```




```
john, 6, 2
juli, 5, 2
```

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 + sign indicates public modifer

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.

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`.

```
Sun Mar 19 14:59:43 IST 2017
```

```
Sun Mar 19 13:00:22 UTC 2017
```

The Date Class Example

```
1  import java.util.Date;
2
3  public class DT {
4
5  public static void main(String[] args)
6  {
7  // creating a date object with specified time.
8  Date d1 = new Date();
9
10 System.out.println(d1.toString());
11
12 }
13 }
```

Problems | Javadoc | Declaration | Console

```
<terminated> DT [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (Dec 8, 2022
Thu Dec 08 08:10:11 EET 2022
```

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

The Random Class Example

```java
import java.util.Random;

public class Ran {

public static void main(String[] args)
{
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
System.out.print(random1.nextInt(1000) + " ");

}
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> Ran [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (Dec 8, 2022, 8:1
From random1: 734 660 210 581 128 202 549 564 459 961
```

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.**

Same Package P1

```java
package p1;

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

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

```java
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();
  }
}
```

```java
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**.

The **public modifier enables unrestricted access**.

Same package P1

```
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.

```java
package P1;

public class A {

        /*Default */
        static int x=5;  //remove static and check

        /*Private*/
        private static int m=9;  // You can use in the same class

        /*Public*/
        public static int w=7;


         public static void main (String []args){
            System.out.println("x (default) = " + x);
            System.out.println("m (private)= " + m);
            System.out.println("w (public )= " + w);
         }

}
```

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**.

```java
package P1;

public class B {

     /*default*/
     static int x2=1;  //remove static and check


    public static void main(String[] args) {
        System.out.println("x2 (default in package p1) = " + x2);
        System.out.println("A.x (default in package p1)= " + A.x);
        //System.out.println("A.m (private in package p1) = " + A.m);
    }

}
```

```java
package P1;

class C { // default class

    static int l=9; // default varaibale
    public static void main(String[] args) {

        System.out.println ("l= " + l);
    }

}
```

```java
package P2;

import P1.A;
/*import P1.C;*/ //Error change visibility to public

public class Test {


    public static void main(String[] args) {
        System.out.println("A.w= "+ A.w);


        //System.out.println("A.x= "+ A.x); //Error: Visibility of x is default, change to public
        //System.out.println("C.l= "+ C.l); // Error : create a class C (C not visible in this package)
    }

}
```

NOTE

An object cannot access its private members, as shown in (b). It is OK, however, if the object is declared in its own class, as shown in (a).

```java
public class C {
  private boolean x;

  public static void main(String[] args) {
    C c = new C();
    System.out.println(c.x);
    System.out.println(c.convert());
  }

  private int convert() {
    return x ? 1 : -1;
  }
}
```

(a) This is okay because object **c** is used inside the class **C**.

```java
public class Test {
  public static void main(String[] args) {
    C c = new C();
    System.out.println(c.x);
    System.out.println(c.convert());
  }
}
```

(b) This is wrong because **x** and **convert** are private in class **C**.

Why Data Fields Should Be private?

To protect data.

To make code easy to maintain.

Example of Data Field Encapsulation

The - sign indicates private modifier

Circle	
-radius: double	The radius of this circle (default: 1.0).
-numberOfObjects: int	The number of circle objects created.
+Circle()	Constructs a default circle object.
+Circle(radius: double)	Constructs a circle object with the specified radius.
+getRadius(): double	Returns the radius of this circle.
+setRadius(radius: double): void	Sets a new radius for this circle.
+getNumberOfObjects(): int	Returns the number of circle objects created.
+getArea(): double	Returns the area of this circle.

Passing Objects to Methods

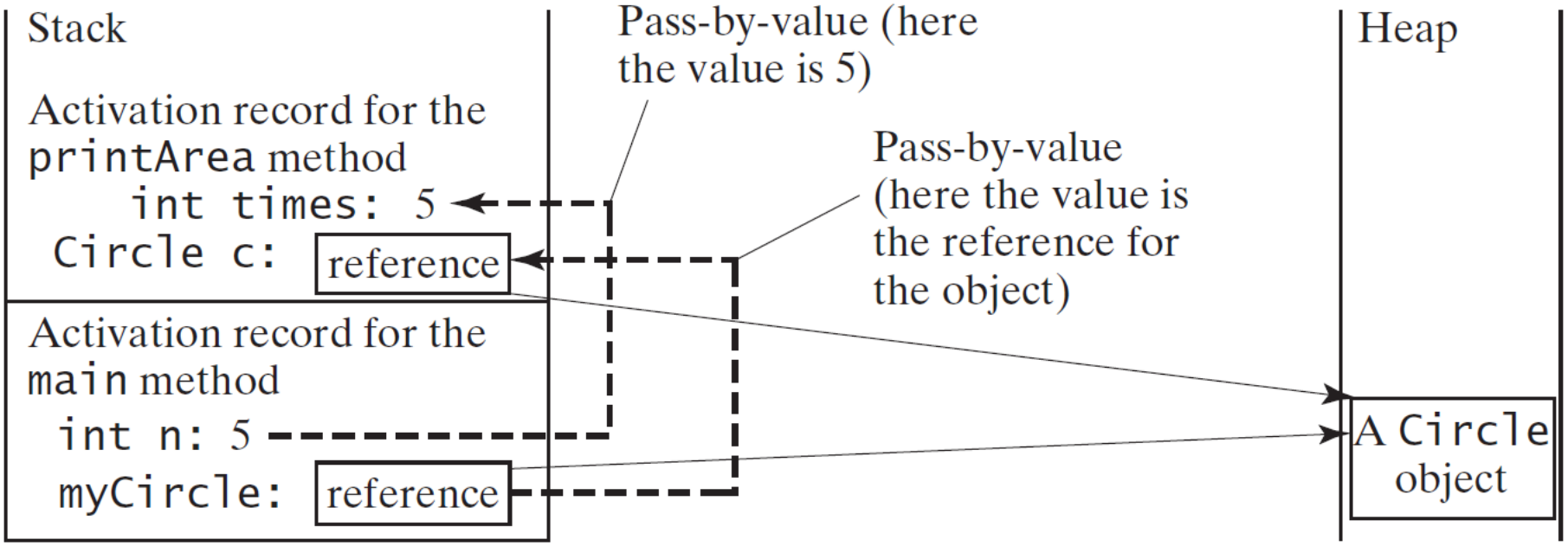
- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)

```
class Apple {
        public String color="red";
}

public class Main {
        public static void main(String[] args) {
                Apple apple = new Apple();
                System.out.println(apple.color);

                changeApple(apple);
                System.out.println(apple.color);
        }

        public static void changeApple(Apple appleObj){
                appleObj.color = "green";
        }
}
```

Output:

```
red
green
```

Passing Objects to Methods, cont.



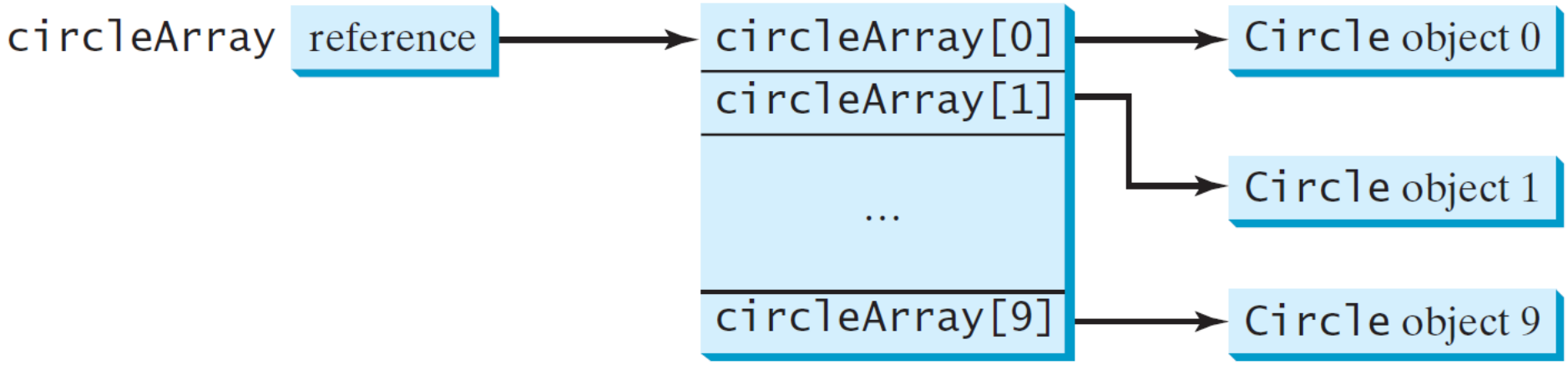
Array of Objects

```
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 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***.

- A class with all private data fields and without mutators is **not necessarily immutable.**

Immutable Objects and Classes

❖ If you delete the **set** method in the **Circle** class, the class would be **immutable** because **radius** is private and cannot be changed without a **set** method.

```
public class Circle {
  private double radius = 1;

  public double getArea() {
    return radius * radius * Math.PI;
  }
  public void setRadius(double r) {
      radius = r;
    }
}
```

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!
  }
}
```

What Class is Immutable?

For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.

Note:
getter called accessor
setter called mutator

Check Point

- If a class contains only private data fields and no setter methods, is the class immutable?

No. It must also contain no get methods that would return a reference to a mutable data field object.

- Is the following class immutable?

```
public class A {
  private int[] values;

  public int[] getValues() {
    return values;
  }
}
```

No, because values is a reference type.

Scope of Variables

- The scope of instance and static variables is the entire class. They can be declared anywhere inside a class.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be initialized explicitly before it can be used.

The this Keyword

- 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.

Reference the Hidden Data Fields

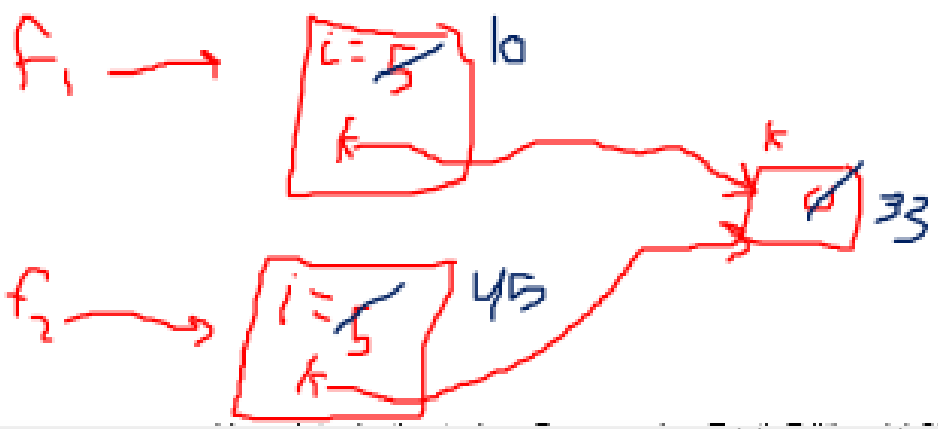
```java
public class F {
  private int i = 5;
  private static double k = 0;

  public void setI(int i) {
    this.i = i;
  }

  public static void setK(double k) {
    F.k = k;
  }

  // Other methods omitted
}
```

(a)

```
Suppose that f1 and f2 are two objects of F.

Invoking f1.setI(10) is to execute
   this.i = 10, where this refers f1

Invoking f2.setI(45) is to execute
   this.i = 45, where this refers f2

Invoking F.setK(33) is to execute
   F.k = 33. setK is a static method
```

(b)

FIGURE 9.21 The keyword **this** refers to the calling object that invokes the method.



The keyword this refers to the object that invokes the instance method setI, as shown in Figure 9.21b. The line F.k = k means that the value in parameter k is assigned to the static data field k of the class, which is shared by all the objects of the class

Overloading Methods and Constructors

- In a class, there can be **several methods with the same name**. However they **must** have **different *signature***.

- The signature of a method is comprised of its ***name***, its ***parameter types*** and the ***order of its parameter***.

- The signature of a method is **not** comprised of its ***return type*** nor ***its visibility*** nor its ***thrown exceptions***.

Calling Overloaded Constructor

```
public class Circle {
  private double radius;

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

  public Circle() {
    this(1.0);
  }

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

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

this is used to invoke another constructor

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