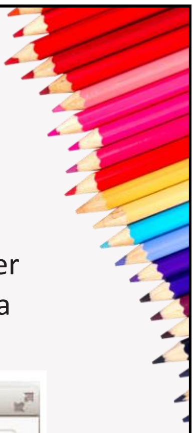
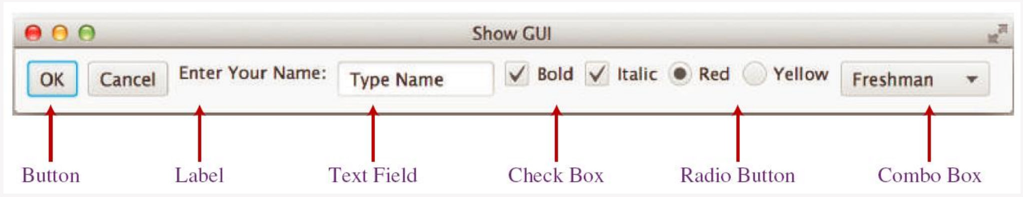
Chapter 9
Objects and Classes



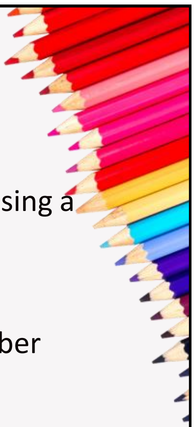
Motivations

- After learning the preceding chapters, you are capable of solving many programming problems using selections, loops, methods, and arrays. However, these Java features are not sufficient for developing graphical user interfaces and large scale software systems. Suppose you want to develop a graphical user interface as shown below. How do you program it?



- *Object-oriented programming enables you to develop large-scale software and GUIs effectively.*

2



Objectives (1)

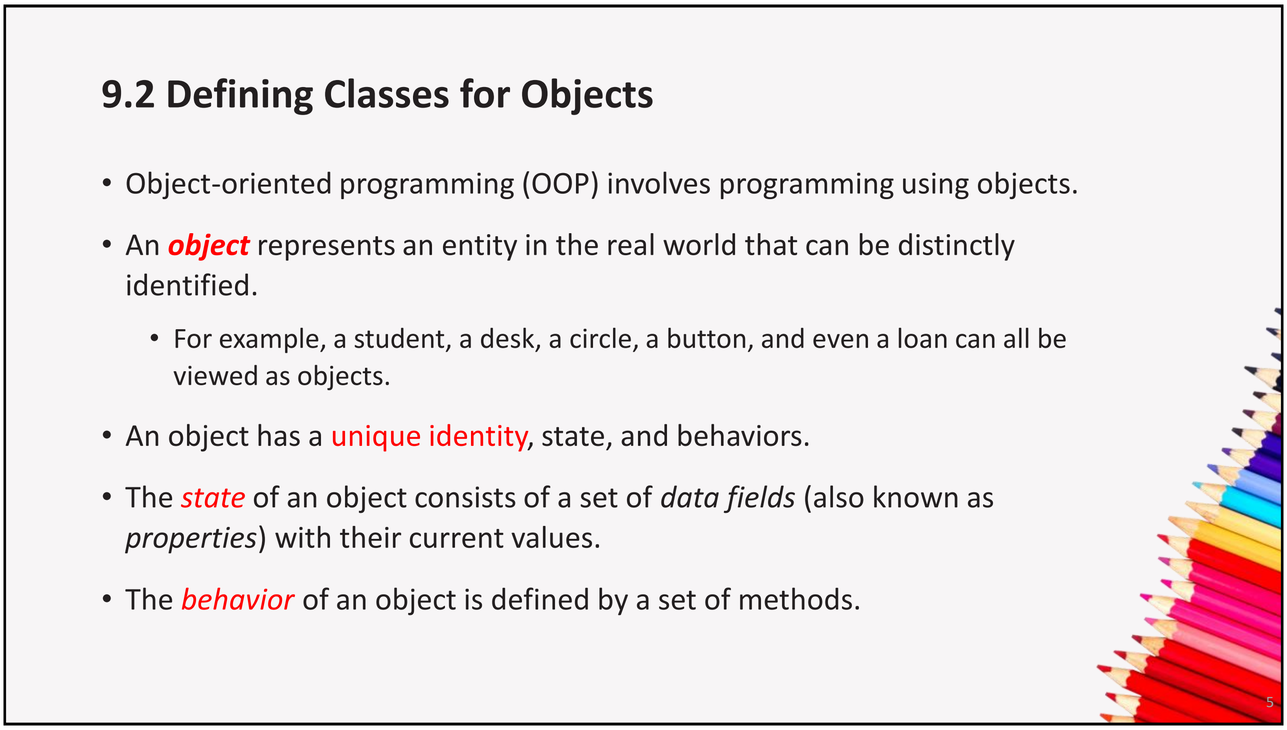
- To describe objects and classes, and use classes to model objects (§9.2).
- To use UML graphical notation to describe classes and objects (§9.2).
- To demonstrate how to define classes and create objects (§9.3).
- To create objects using constructors (§9.4).
- To access objects via object reference variables (§9.5).
- To define a reference variable using a reference type (§9.5.1).
- To access an object's data and methods using the object member access operator (.) (§9.5.2).
- To define data fields of reference types and assign default values for an object's data fields (§9.5.3).
- To distinguish between object reference variables and primitive data type variables (§9.5.4).

3

Objectives (2)

- To use the Java library classes Date, Random, and Point2D (§9.6).
- To distinguish between instance and static variables and methods (§9.7).
- To define private data fields with appropriate getter and setter methods (§9.8).
- To encapsulate data fields to make classes easy to maintain (§9.9).
- To develop methods with object arguments and differentiate between primitive-type arguments and object-type arguments (§9.10).
- To store and process objects in arrays (§9.11).
- To create immutable objects from immutable classes to protect the contents of objects (§9.12).
- To determine the scope of variables in the context of a class (§9.13).
- To use the keyword this to refer to the calling object itself (§9.14).

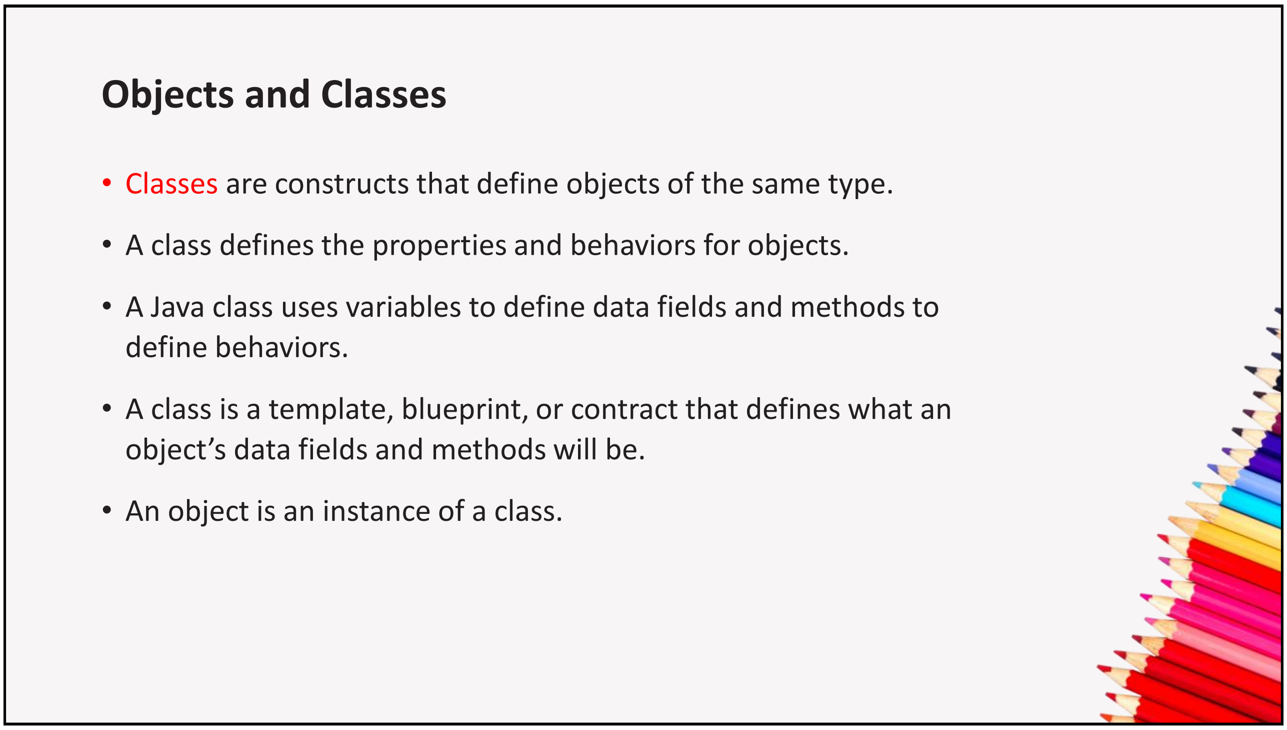
4



9.2 Defining Classes for Objects

- 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.

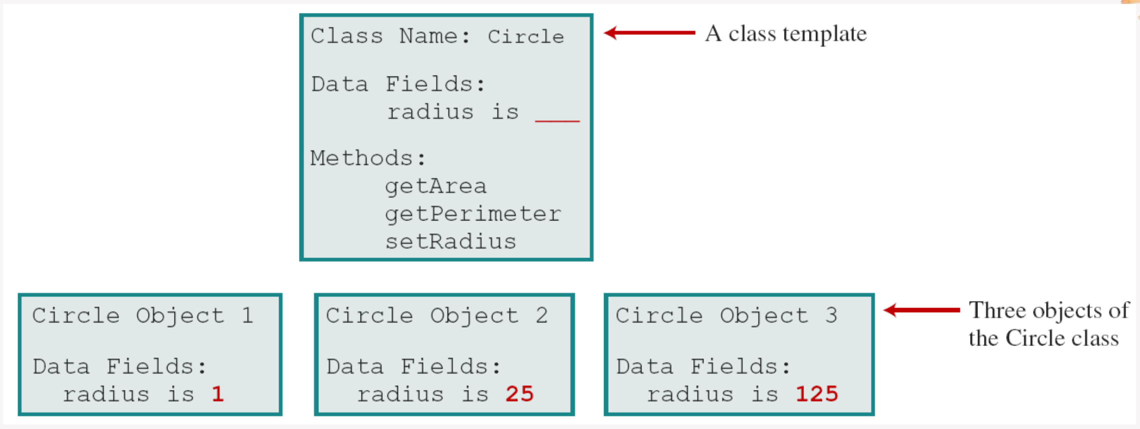
5



Objects and Classes

- Classes are constructs that define objects of the same type.
- A class defines the properties and behaviors for objects.
- A Java class uses variables to define data fields and methods to define behaviors.
- A class is a template, blueprint, or contract that defines what an object’s data fields and methods will be.
- An object is an instance of a class.

A class is a template for creating objects

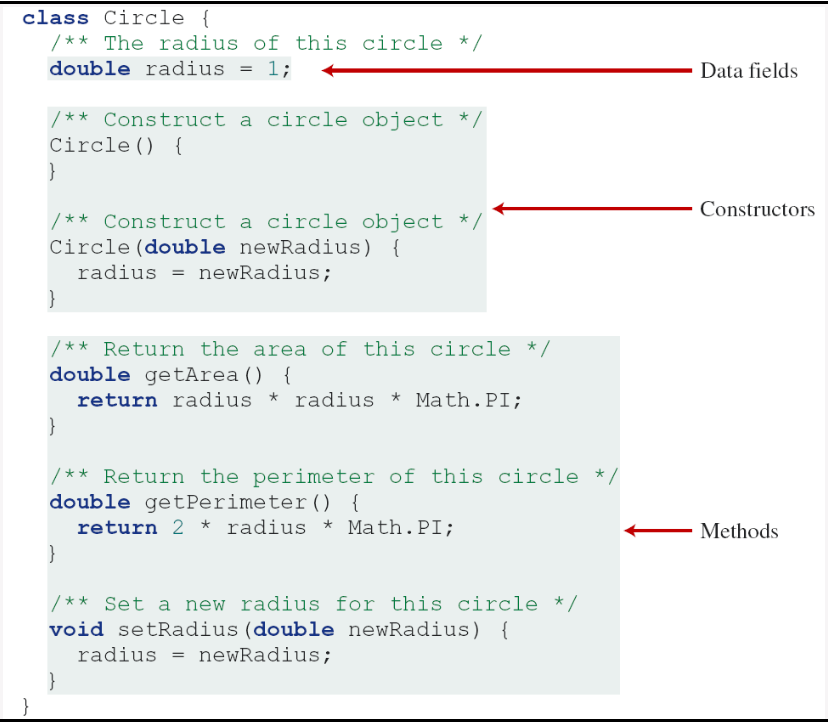


Objects and Classes

- Creating an instance is referred to as *instantiation*.
- *The terms object and instance are often interchangeable.*
- A Java class uses variables to define data fields and methods to define actions. Additionally, a class provides methods of a special type, known as *constructors, which are invoked to* create a new object.
- A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects.

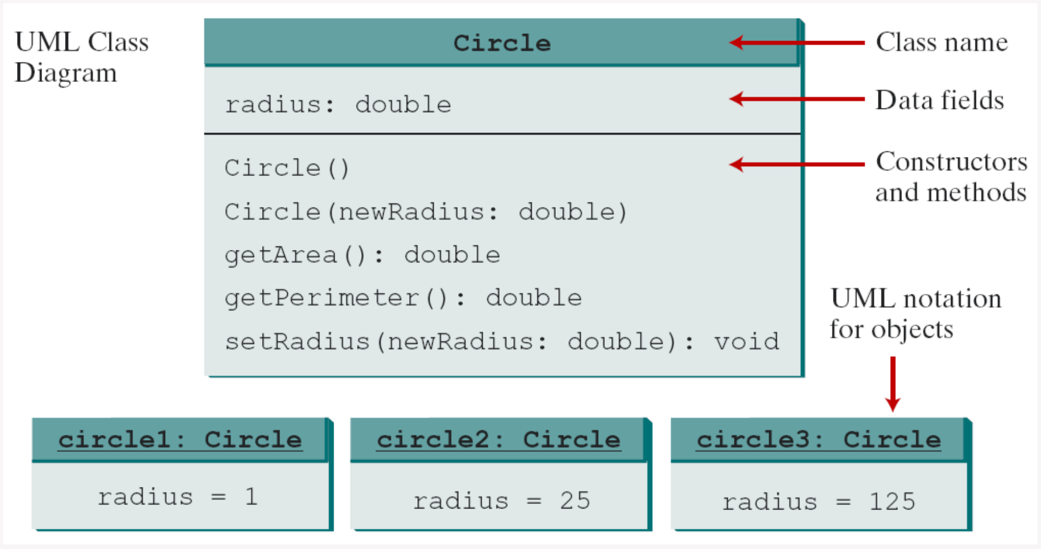
8

Classes



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

UML Class Diagram



Example: Defining Classes and Creating Objects

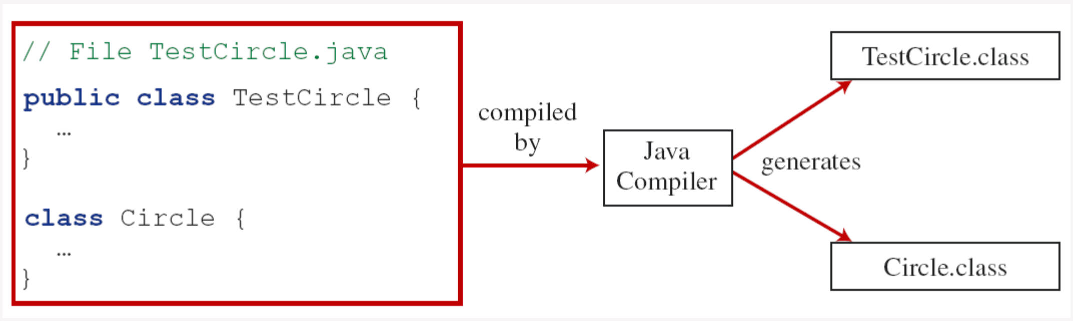
- *Classes are definitions for objects and objects are created from classes.*
- Objective: Demonstrate creating objects, accessing data, and using methods.

TestSimpleCircle Run

AlternaviteCircle Run

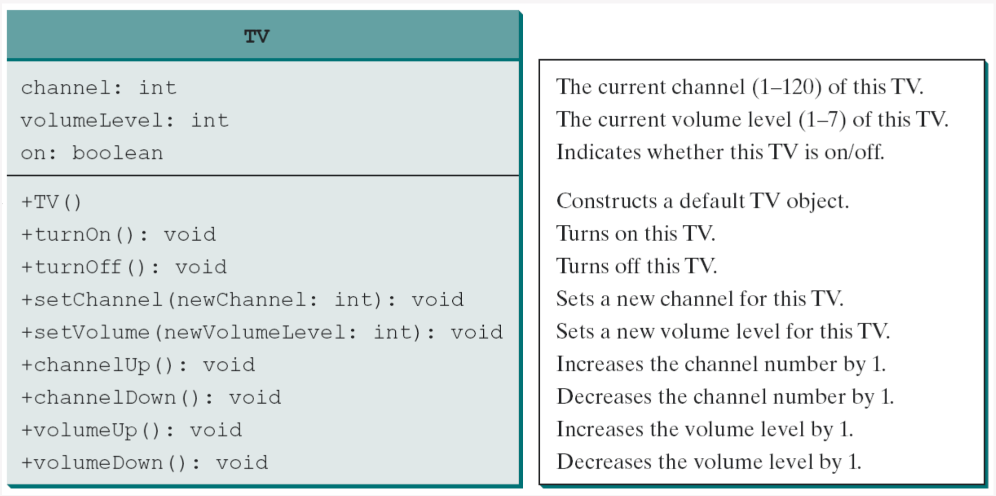
Note

- You can put the two classes into one file, but only one class in the file can be a *public class*.
- Each class in the source code file is compiled into a **.class** file.



```
// File TestCircle.java
public class TestCircle {
  …
}

class Circle {
  …
}
```

Example: Defining Classes and Creating Objects



TV	
`channel: int`	The current channel (1–120) of this TV.
`volumeLevel: int`	The current volume level (1–7) of this TV.
`on: boolean`	Indicates whether this TV is on/off.
`+TV()`	Constructs a default TV object.
`+turnOn(): void`	Turns on this TV.
`+turnOff(): void`	Turns off this TV.
`+setChannel(newChannel: int): void`	Sets a new channel for this TV.
`+setVolume(newVolumeLevel: int): void`	Sets a new volume level for this TV.
`+channelUp(): void`	Increases the channel number by 1.
`+channelDown(): void`	Decreases the channel number by 1.
`+volumeUp(): void`	Increases the volume level by 1.
`+volumeDown(): void`	Decreases the volume level by 1.

TV

TestTV

Run

13

9.4 Constructing Objects Using Constructors

- *A constructor is invoked to create an object using the **new** operator.*
- Constructors are a special kind of methods that are invoked to construct objects. They have three peculiarities:
  1. A constructor must have the same name as the class itself.
  2. Constructors do not have a return type—not even **void.**
  3. Constructors are invoked using the **new** operator when an object is created. Constructors play the role of initializing objects.

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

Constructors

- To construct an object from a class, invoke a constructor of the class using the **new** operator, as follows:

```
new ClassName(arguments);
```

- Example:

```
new Circle();

new Circle(5.0);
```

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

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*.

9.5 Accessing Objects via Reference Variables

- Newly created objects are allocated in the memory. They can be accessed via reference variables.
- To reference an object, assign the object to a reference variable.
- To declare a reference variable, use the syntax:

```
ClassName objectRefVar;
```

- Example:

```
Circle myCircle;
```

- The next statement creates an object and assigns its reference to **myCircle:**

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

Declaring/Creating Objects in a Single Step

- You can write a single statement that combines the declaration of an object reference variable, the creation of an object, and the assigning of an object reference to the variable with the following syntax:

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

- Example:

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

Accessing Object's Members

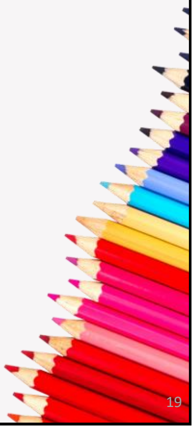
- An object's data and methods can be accessed through the dot (.) operator via the object's reference variable.
- Referencing the object's data:

  `objectRefVar.data`

  - e.g., `myCircle.radius`
- Invoking the object's method:

  `objectRefVar.methodName(arguments)`

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



Caution

- Recall that you use

  `Math.methodName(arguments) (e.g., Math.pow(3, 2.5))`

to invoke a method in the Math class.

- Can you invoke `getArea()` using `SimpleCircle.getArea()`?
- The answer is ***no***.

- All the methods used before this chapter are *static methods*, which are defined using the static keyword.
- However, `getArea()` is *non-static*. It must be invoked from an object using

  `objectRefVar.methodName(arguments) (e.g., myCircle.getArea()).`

- More explanations will be given in the section on "Static Variables, Constants, and Methods."

Reference Data Fields and the `null` Value

- The data fields can be of reference types.
- For example, the following Student class contains a data field name of the **`String`** type.
- If a data field of a reference type does not reference any object, the data field holds a special literal value, **`null`**.

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

21

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.

```
class TestStudent {
  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);
  }
}
```

- **NullPointerException** is a common runtime error. It occurs when you invoke a method on a reference variable with a **null** value.

22

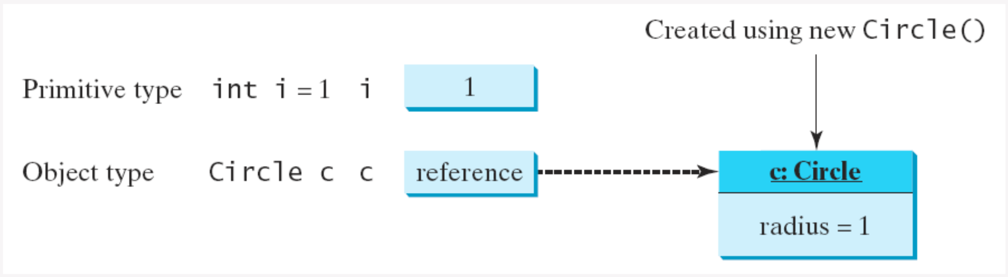
Default Value for a Data Field

- However, Java assigns **no default value to a *local variable*** inside a method.
- The following code has a compile error, because the local variables x and y are not initialized:

```
class TestLocalVariables {
  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);
  }
}
```

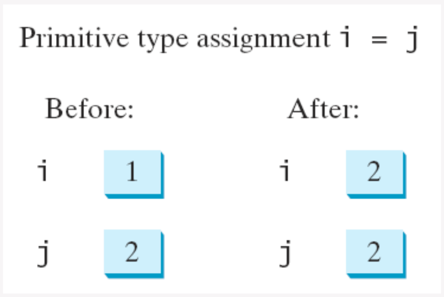
Differences between Variables of Primitive Types and Reference Types

- Every variable represents a memory location that holds a value.
- For a variable of a primitive type, the value is of the primitive type.
- For a variable of a reference type, the value is a reference to where an object is located.



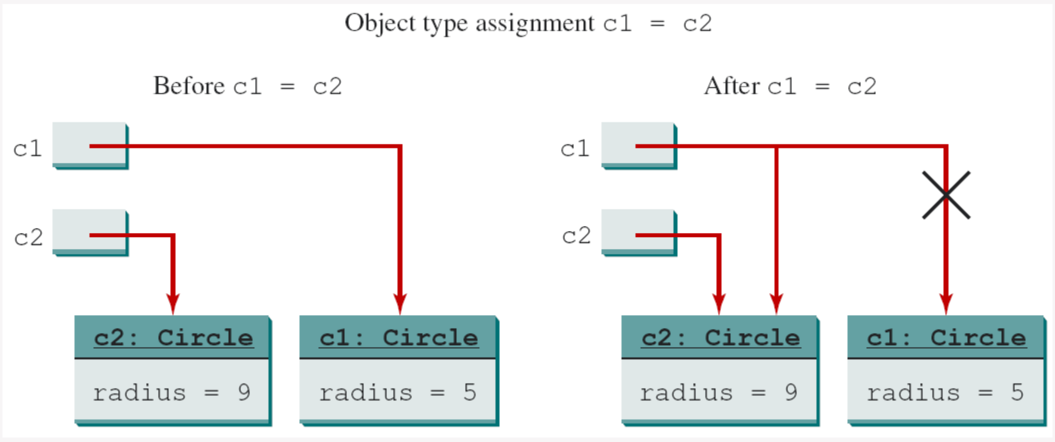
Copying Variables of Primitive Data Types

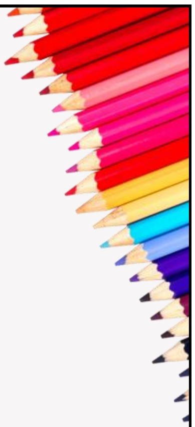
- When you assign one variable to another, the other variable is set to the same value.
- For a variable of a primitive type, the real value of one variable is assigned to the other variable.



Copying Variables of Object Types

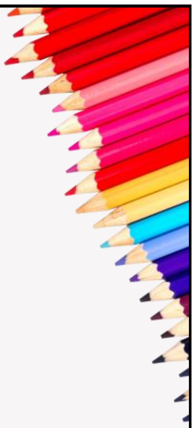
- For a variable of a reference type, the reference of one variable is assigned to the other variable.





Garbage Collection

- The JVM will *automatically* collect the space if the object is not referenced by any variable.
  - 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 occupies memory space, so the Java runtime system detects garbage and automatically reclaims the space it occupies. This process is called *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.

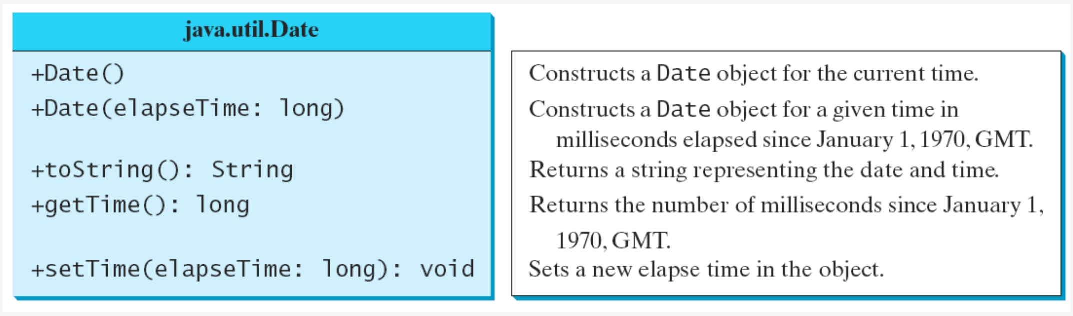


9.6 Using Classes from the Java Library

- The Java API contains a rich set of classes for developing Java programs.
- the **java.util.Date class**
  - A Date object represents a specific date and time.
- the **java.util.Random** class
  - A Random object can be used to generate random values.
- the **javafx.geometry.Point2D** class
  - A Point2D object represents a point with x- and y-coordinates.

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.



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

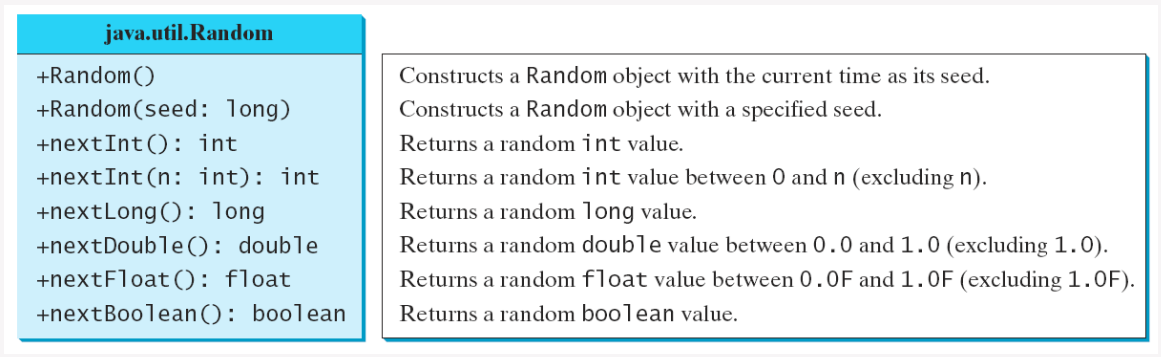
For example, the following code

```
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`.

The Random Class

- A 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 (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.

The Random Class Example

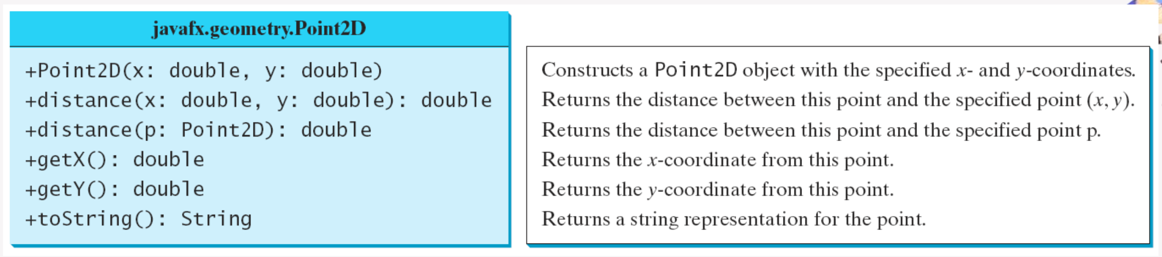
If two Random objects have the same seed, they will generate identical sequences of numbers. For example, the following code creates two Random objects with the same seed 3.

```
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 Point2D Class

- Java API has a conveninent Point2D class in the javafx.geometry package for representing a point in a two-dimensional plane.



javafx.geometry.Point2D	
`+Point2D(x: double, y: double)`	Constructs a `Point2D` object with the specified *x*- and *y*-coordinates.
`+distance(x: double, y: double): double`	Returns the distance between this point and the specified point (*x*, *y*).
`+distance(p: Point2D): double`	Returns the distance between this point and the specified point p.
`+getX(): double`	Returns the *x*-coordinate from this point.
`+getY(): double`	Returns the *y*-coordinate from this point.
`+toString(): String`	Returns a string representation for the point.

- JDK9+: The **Point2D** class is defined in the **javafx.geometry** package, which is in the JavaFX module. To run this program, you need to install JavaFX.

Point2D

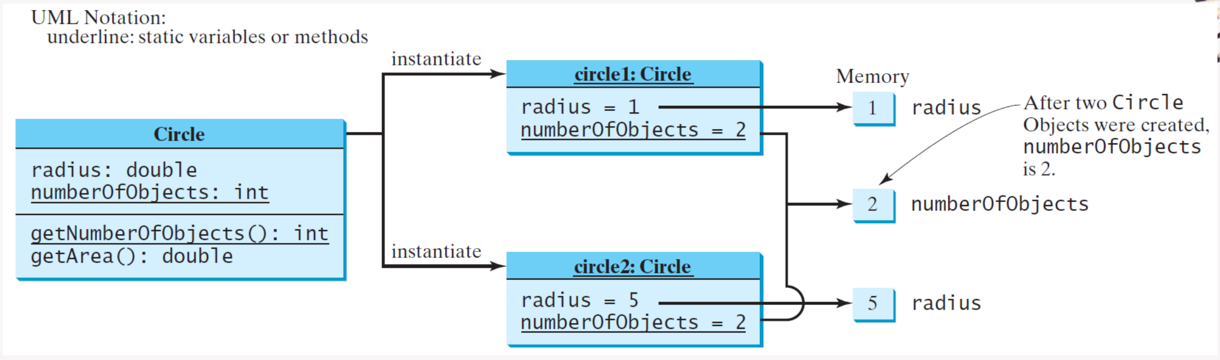
Run

9.7 Static Variables, Constants, and Methods

- Instance variables belong to a specific instance.
- Instance methods are invoked by an instance of the class.
- 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.
- To declare static variables, constants, and methods, use the *static modifier*.

Static Variables, Constants, and Methods, cont.

- Instance variables belong to the instances and have memory storage independent of one another. Static variables are shared by all the instances of the same class.



Example of Using Instance and Class Variables and Method

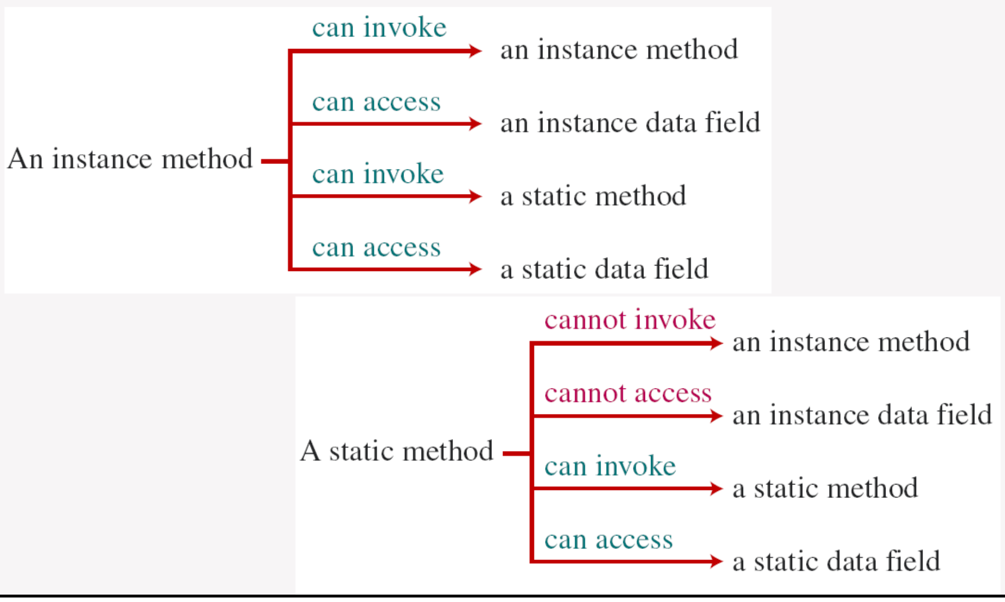
- Objective: Demonstrate the roles of instance and class variables and their uses.
- This example adds a class variable numberOfObjects to track the number of Circle objects created.

CircleWithStaticMembers

TestCircleWithStaticMembers

Run

Instance Method vs. Static Method



Wrong Code

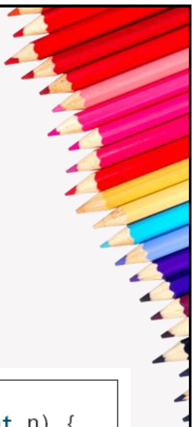
```java
public class A {
  int i = 5;
  static int  k = 2;

  public static void main(String[] args) {
    int j = i; // Wrong because i is an instance variable
    m1(); // Wrong because m1() is an instance method
  }

  public void m1() {
    // Correct since instance and static variables and methods
    // can be used in an instance method
    i = i + k + m2(i, k);
  }

  public static int m2(int i, int j) {
    return (int)(Math.pow(i, j));
  }
}
```

Correct Code

```java
public class A {
  int i = 5;
  static int k = 2;

  public static void main(String[] args) {
    A a = new A();
    int j = a.i; // OK, a.i accesses the object's instance variable
    a.m1(); // OK, a.m1() invokes the object's instance method
  }

  public void m1() {
    i = i + k + m2(i, k);
  }

  public static int m2(int i, int j) {
    return (int)(Math.pow(i, j));
  }
}
```

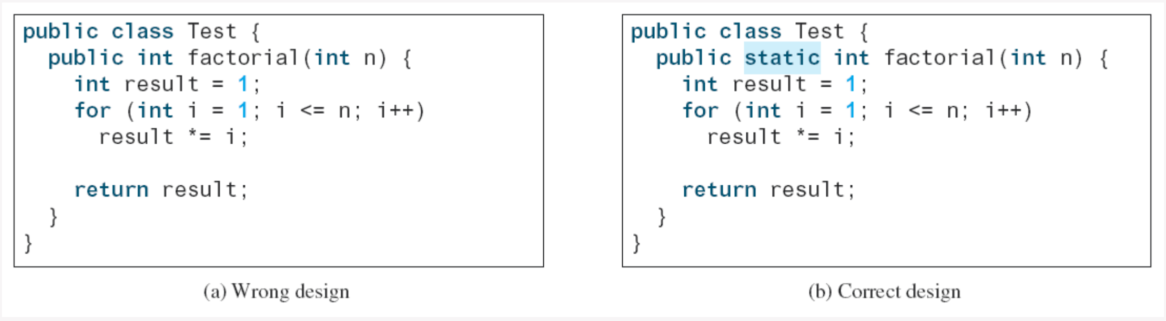
TIPs

- Use **`ClassName.methodName(arguments)`** to invoke a static method and **`ClassName.staticVariable`** to access a static variable. This improves readability,
- How do you decide whether a variable or a method should be an instance one or a static one?
  - A variable or a method that is dependent on a specific instance of the class should be an instance variable or method.
  - A variable or a method that is not dependent on a specific instance of the class should be a static variable or method.



Instance or static ?

- It is a common design error to define an instance method that should have been defined as static.
  - For example, the method **factorial(int n)** should be defined as static because it is independent of any specific instance.

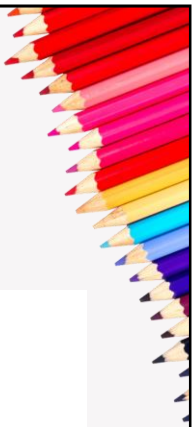


```
public class Test {
  public int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++)
      result *= i;

    return result;
  }
}
```

(a) Wrong design

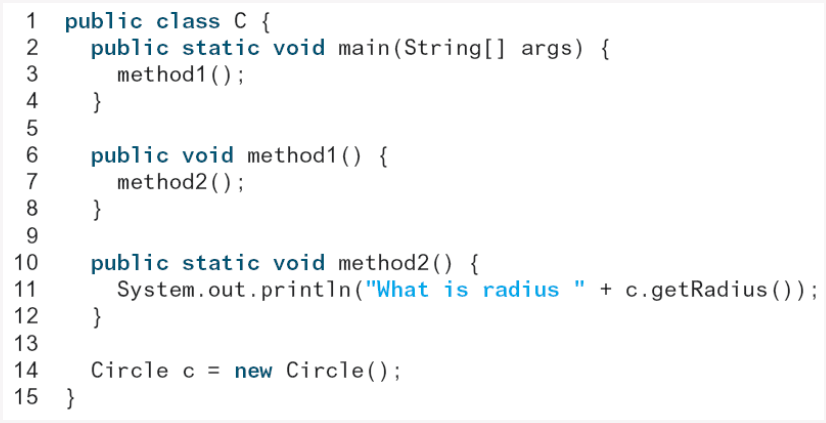
```
public class Test {
  public static int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++)
      result *= i;

    return result;
  }
}
```

(b) Correct design

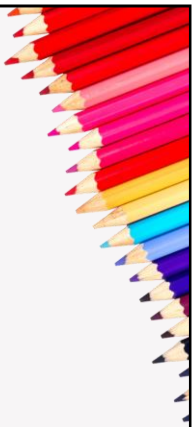


Check Point

- Can you invoke an instance method or reference an instance variable from a static method? Can you invoke a static method or reference a static variable from an instance method? What is wrong in the following code?

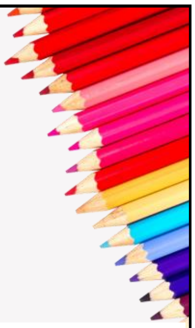


```
public class C {
  public static void main(String[] args) {
    method1();
  }

  public void method1() {
    method2();
  }

  public static void method2() {
    System.out.println("What is radius " + c.getRadius());
  }

  Circle c = new Circle();
}
```



9.8 Visibility Modifiers

- *Visibility modifiers can be used to specify the visibility of a class and its members.*
- 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.



Visibility Modifiers

- 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.

```
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();
  }
}
```

Visibility Modifiers

- A nonpublic class has package access.

```
package p1;

class C1 {
  ...
}
```

```
package p1;

public class C2 {
  can access C1
}
```

```
package p2;

public class C3 {
  cannot access p1.C1;
  can access p1.C2;
}
```

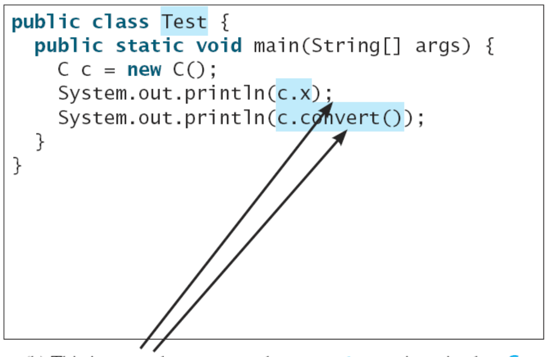
45

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).

```
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.



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

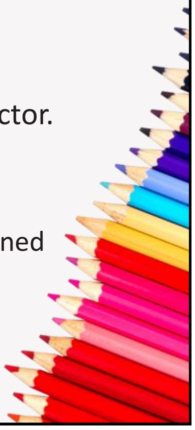
(b) This is wrong because x and convert are private in class C.

46

Note

- The **private** modifier applies only to the members of a class. The **public** modifier can apply to a class or members of a class.
  - Using the modifiers public and private on local variables would cause a compile error.
- In most cases, the constructor should be public. However, if you want to prohibit the user from creating an instance of a class, use a private constructor.
  - For example, there is no reason to create an instance from the Math class, because all of its data fields and methods are static. To prevent the user from creating objects from the Math class, the constructor in java.lang.Math is defined as follows:

```
private Math() {}
```



9.9 Data Field Encapsulation

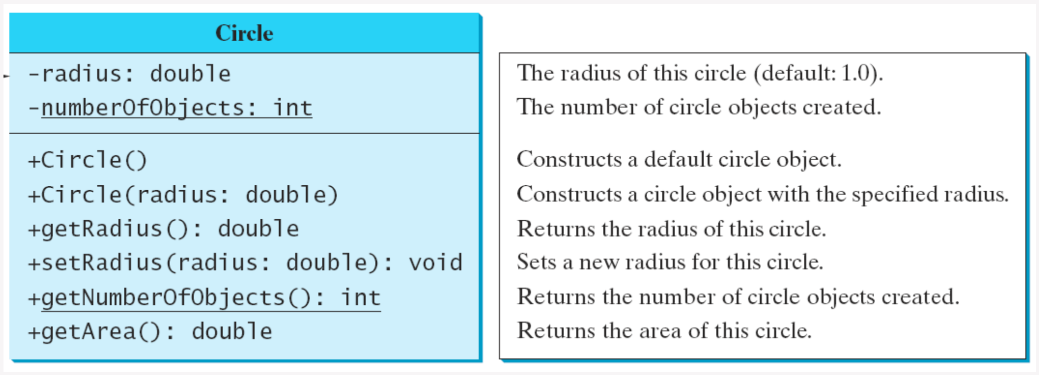
- Why Data Fields Should Be private?
  - To protect data.
  - To make code easy to maintain.
- *getter and setter*
  - To make a private data field accessible, provide a *getter method to return its value.*
  - To enable a private data field to be updated, provide a *setter method to* set a new value.
- A getter method is also referred to as an *accessor and a setter to a mutator.*
- A getter method has the following signature:

  public returnType get*PropertyName()*
- If the returnType is boolean, the getter method should be defined as follows by convention:

  public boolean is*PropertyName()*
- A setter method has the following signature:

  public void set*PropertyName(dataType propertyValue)*

Example of Data Field Encapsulation



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.

CircleWithPrivateDataFields

TestCircleWithPrivateDataFields

Run

49

9.10 Passing Objects to Methods

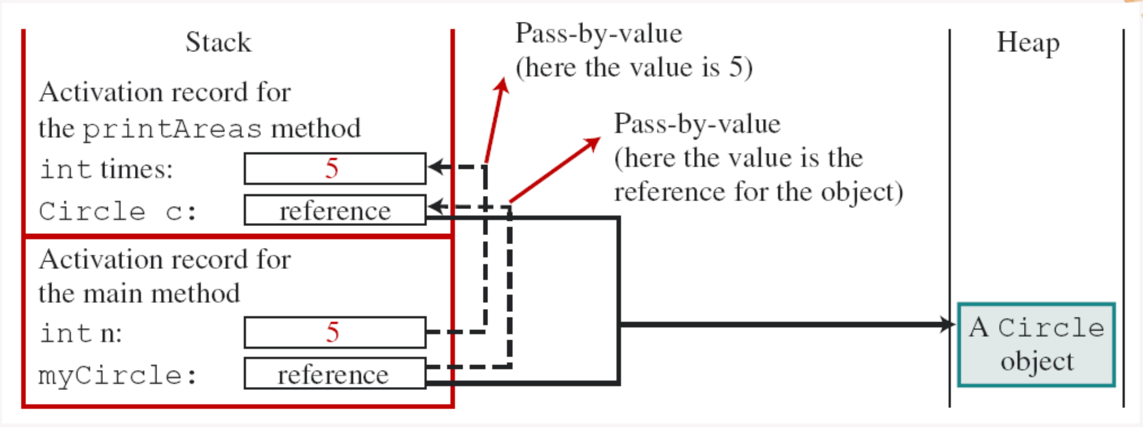
- *Passing an object to a method is to pass the reference of the object.*
- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)

TestPassObject

Run

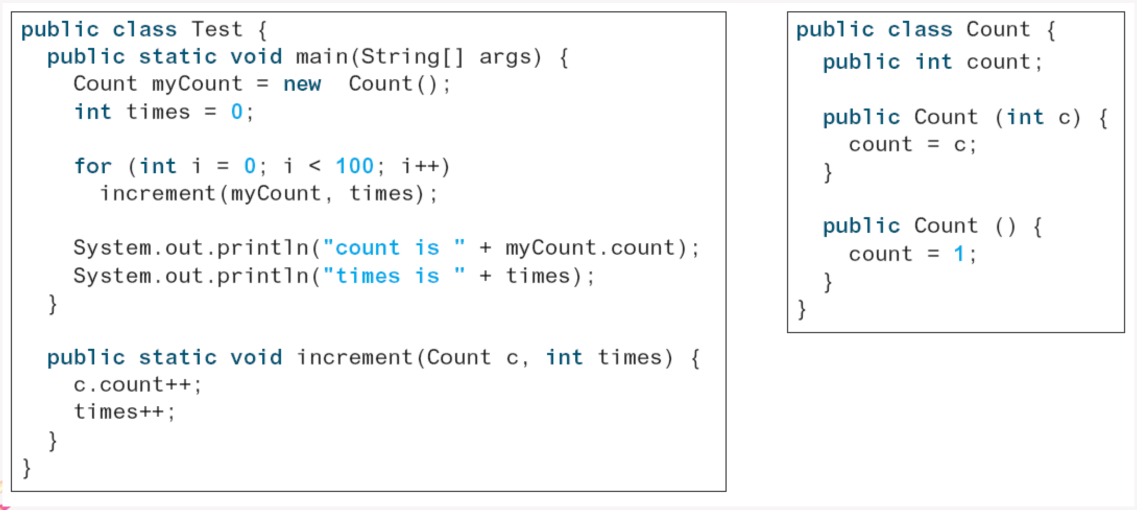
50

Passing Objects to Methods


Stack
Activation record for the printAreas method
int times: 5
Circle c: reference
Activation record for the main method
int n: 5
myCircle: reference
Pass-by-value (here the value is 5)
Pass-by-value (here the value is the reference for the object)
Heap
A Circle object

51

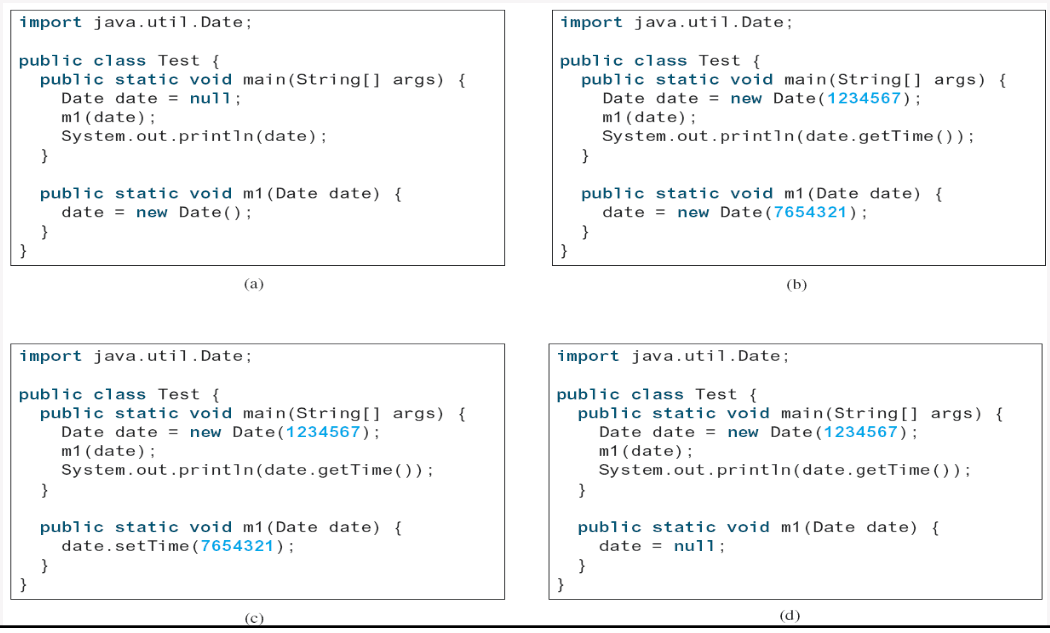
Check Point



```
public class Test {
  public static void main(String[] args) {
    Count myCount = new  Count();
    int times = 0;

    for (int i = 0; i < 100; i++)
      increment(myCount, times);

    System.out.println("count is " + myCount.count);
    System.out.println("times is " + times);
  }

  public static void increment(Count c, int times) {
    c.count++;
    times++;
  }
}
```

```
public class Count {
  public int count;

  public Count (int c) {
    count = c;
  }

  public Count () {
    count = 1;
  }
}
```

Check Point



```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = null;
    m1(date);
    System.out.println(date);
  }

  public static void m1(Date date) {
    date = new Date();
  }
}
```

(a)

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date = new Date(7654321);
  }
}
```

(b)

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date.setTime(7654321);
  }
}
```

(c)

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date = null;
  }
}
```

(d)

9.11 Array of Objects

- *An array can hold objects as well as primitive type values.*

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

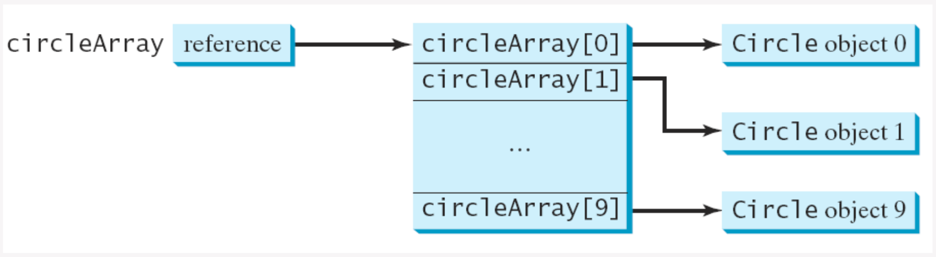
- *When an array of objects is created using the new operator, each element in the array is a reference variable with a default value of null.*

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

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

54

Array of Objects, cont.

- An array of objects is actually an array of reference variables. In an array of objects, an element of the array contains a reference to an object.



- Summarizing the areas of the circles

TotalArea Run

9.12 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*.
- *You can define immutable classes to create immutable objects.*
- If a class is immutable, then all its data fields must be private and it cannot contain public setter methods for any data fields.
- A class with all private data fields and no mutators is *not necessarily* immutable.

Is class Student immutable?

```
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;
  }
}
```

57

What Class is Immutable?

```
public class Test {
  public static void main(String[] args) {
    Student student = new Student(111223333, "John");
    java.util.Date dateCreated = student.getDateCreated();
    dateCreated.setTime(200000); // Now dateCreated field is changed!
  }
}
```

- For a class to be immutable, it must

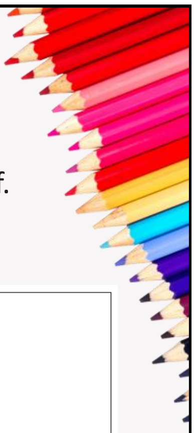
1. mark all data fields private and
2. provide no mutator methods and
3. no accessor methods that would return a reference to a mutable data field object.

58

9.13 The Scope of Variables

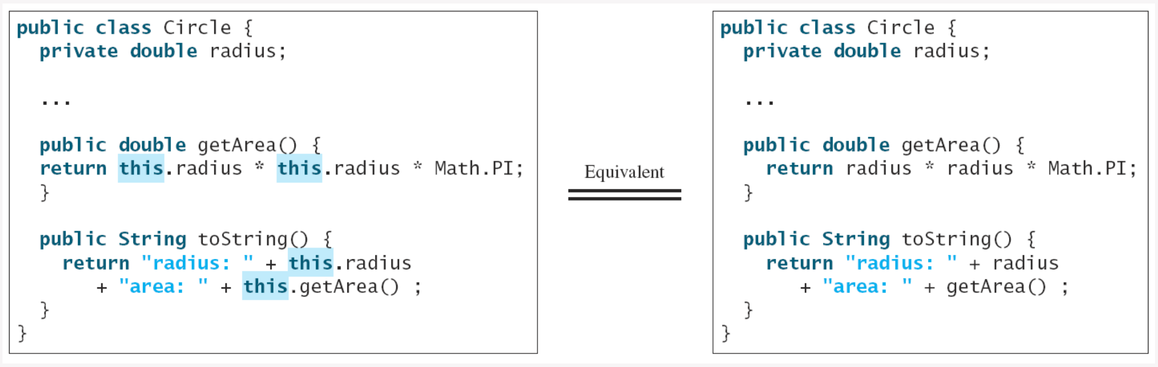
- Instance and static variables in a class are referred to as the *class's variables or data fields.*
- *The scope of instance and static variables is the entire class, regardless of where the variables are declared.*
  - A class's variables and methods can appear in any order in the class.
  - The exception is when a data field is initialized based on a reference to another data field.
- A variable defined inside a method is referred to as a *local variable.*
- 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.

59



9.14 The `this` Reference

- The **`this`** keyword is the name of a reference that refers to an object itself.



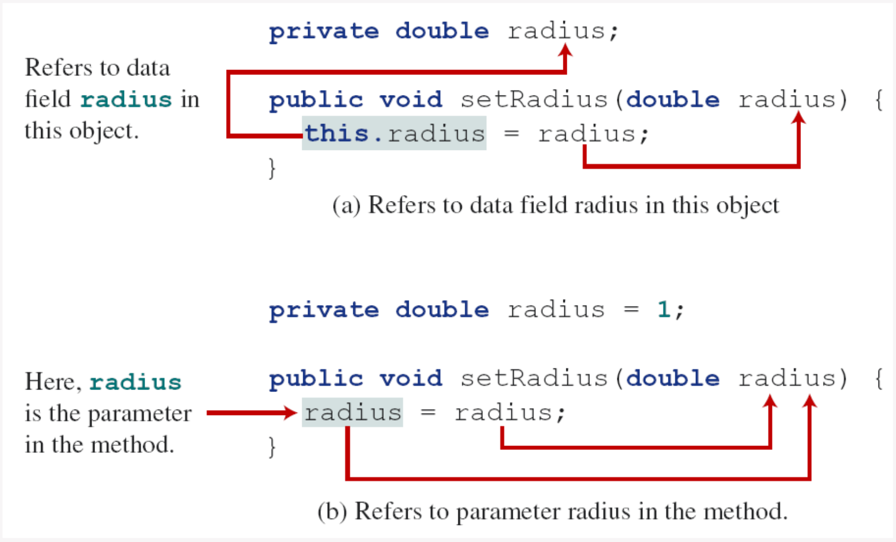
```
public class Circle {
  private double radius;

  ...

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

  public String toString() {
    return "radius: " + this.radius
      + "area: " + this.getArea() ;
  }
}
```

Equivalent

```
public class Circle {
  private double radius;

  ...

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

  public String toString() {
    return "radius: " + radius
      + "area: " + getArea() ;
  }
}
```

60

Using this to Reference Data Fields



Refers to data field **radius** in this object.

```
private double radius;

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

(a) Refers to data field radius in this object

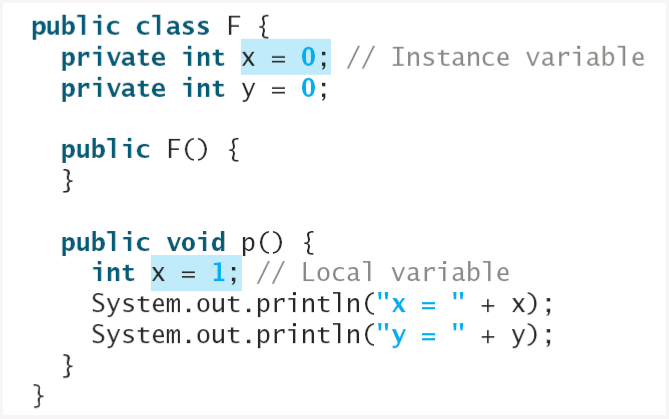
Here, **radius** is the parameter in the method.

```
private double radius = 1;

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

(b) Refers to parameter radius in the method.

the Hidden Data Fields

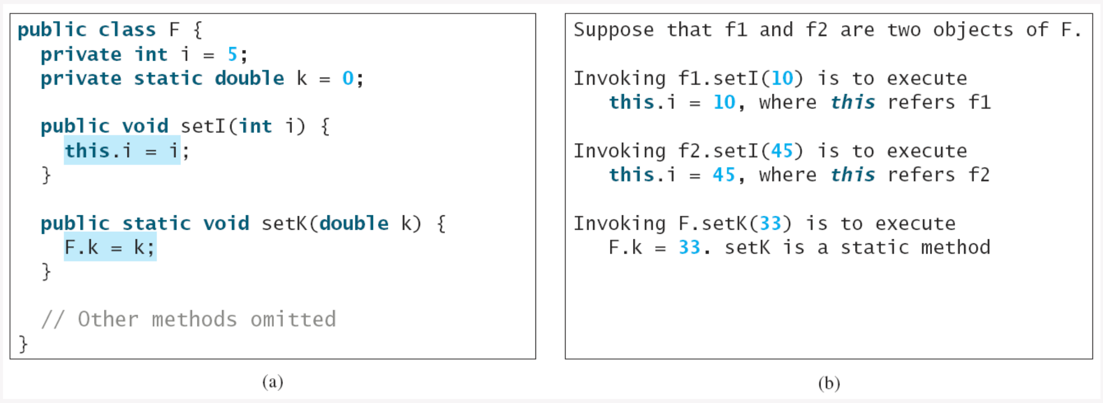
- If a local variable has the same name as a class's variable, the local variable takes precedence and the class's variable with the same name is *hidden.*



```
public class F {
  private int x = 0; // Instance variable
  private int y = 0;

  public F() {
  }

  public void p() {
    int x = 1; // Local variable
    System.out.println("x = " + x);
    System.out.println("y = " + y);
  }
}
```

Using this to Reference Hidden Data Fields

- One common use of the **this** keyword is reference a class's *hidden data fields*.



```
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
}
```

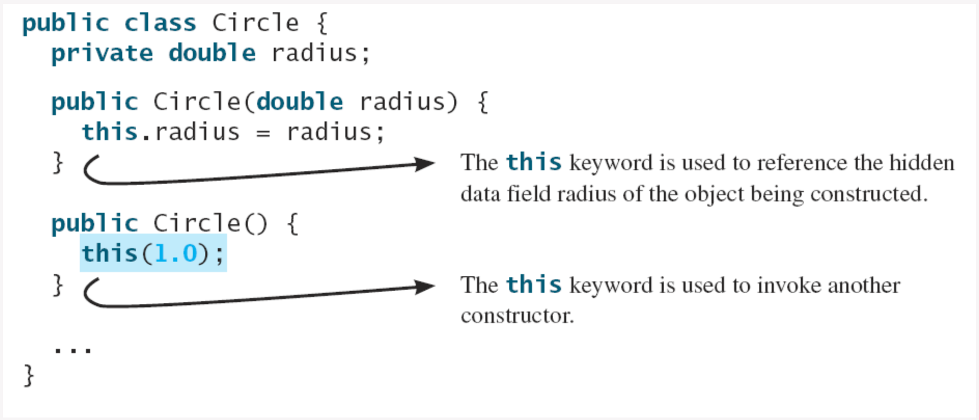
(a)

```
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)

Using this to Invoke a Constructor

- Another common use of the **this** keyword to enable a constructor to invoke another constructor of the same class.

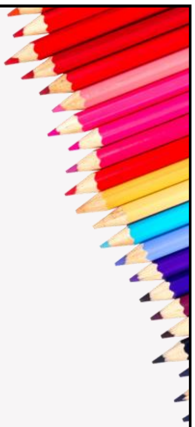


```
public class Circle {
  private double radius;

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

  public Circle() {
    this(1.0);
  }

  ...
}
```

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

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

Using this to Invoke a Constructor



- ***NOTE:*** Java requires that the this(arg-list) statement appear first in the constructor before any other executable statements.
- **Tip**
  - If a class has multiple constructors, it is better to implement them using **this(arg-list)** as much as possible.
  - In general, a constructor with no or fewer arguments can invoke a constructor with more arguments using **this(arg-list)**.
  - This syntax often simplifies coding and makes the class easier to read and to maintain.

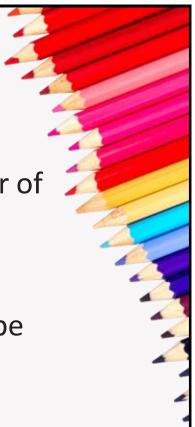
Chapter Summary

Chapter Summary

- A class is a template for objects. It defines the properties of objects and provides constructors for creating objects and methods for manipulating them.
- A class is also a data type. You can use it to declare object reference variables. An object reference variable that appears to hold an object actually contains a reference to that object. Strictly speaking, an object reference variable and an object are different, but most of the time the distinction can be ignored.
- An object is an instance of a class. You use the new operator to create an object, and thedot operator (.) to access members of that object through its reference variable.

Chapter Summary

- An *instance variable or method belongs to an instance of a class. Its use is associated* with individual instances. A *static variable is a variable shared by all instances of the* same class. A *static method is a method that can be invoked without using instances.*
- The scope of instance and static variables is the entire class, regardless of where the variables are declared.
- Visibility modifiers specify how the class, method, and data are accessed. A **public** class, method, or data is accessible to all clients. A **private** method or data is accessible only inside the class.
- You can provide a getter (accessor) method or a setter (mutator) method to enable clients to see or modify the data.



Chapter Summary

- All parameters are passed to methods using pass-by-value. For a parameter of a primitive type, the actual value is passed; for a parameter of a *reference type, the reference* for the object is passed.
- A Java array is an object that can contain primitive type values or object type values. When an array of objects is created, its elements are assigned the default value of **null.**
- Once it is created, an immutable object cannot be modified. To prevent users from modifying an object, you can define *immutable classes.*
- The keyword **this** can be used to refer to the calling object. It can also be used inside a constructor to invoke another constructor of the same class.

Programming Exercises

1, 2, 3, 4, 5, 6, 7,

8, 9, 10, 11, 13

70