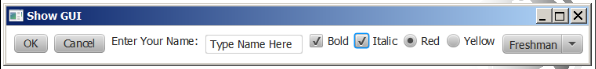
Chapter 9 Objects and Classes

CS1: Java Programming
Colorado State University

Original slides by Daniel Liang
Modified slides by Chris Wilcox

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?



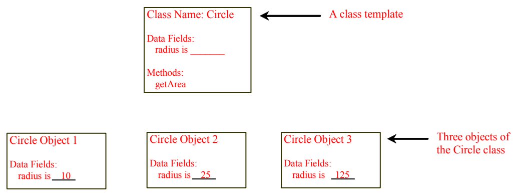
Objectives

- 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).
- 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 **get** and **set** 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).

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



An object has both a state and behavior. The state defines the object, and the behavior defines what the object does.

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.

Classes

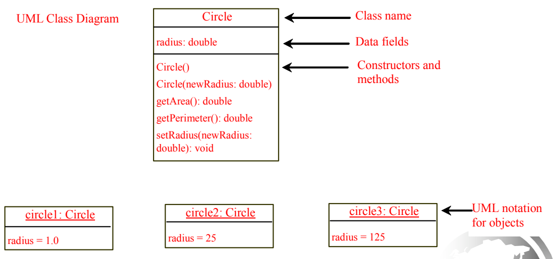
```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** 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

UML Class Diagram

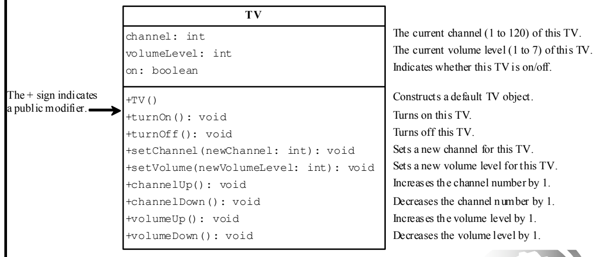


Example: Defining Classes and Creating Objects

Objective: Demonstrate creating objects, accessing data, and using methods.

TestSimpleCircle Run

Example: Defining Classes and Creating Objects


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

TV

TestTV Run

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.

Creating Objects Using Constructors

```
new ClassName();
```

Example:

```
new Circle();

new Circle(5.0);
```

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*.

Declaring Object Reference Variables

To reference an object, assign the object to a reference variable.

To declare a reference variable, use the syntax:

```
ClassName objectRefVar;
```

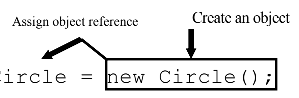
Example:

```
Circle myCircle;
```

Declaring/Creating Objects in a Single Step

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

Example:



```
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()`

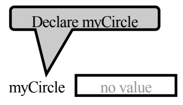
animation

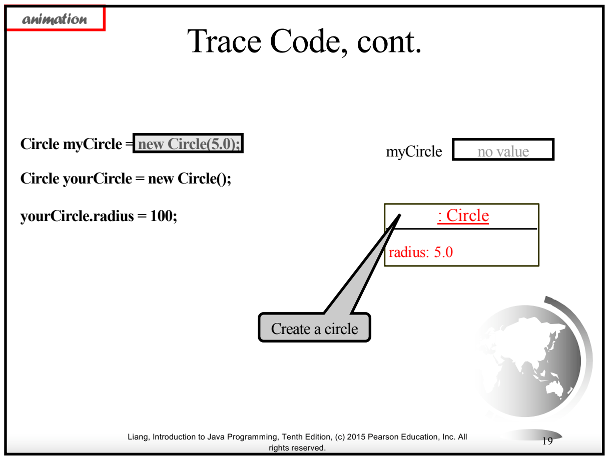
Trace Code

Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;

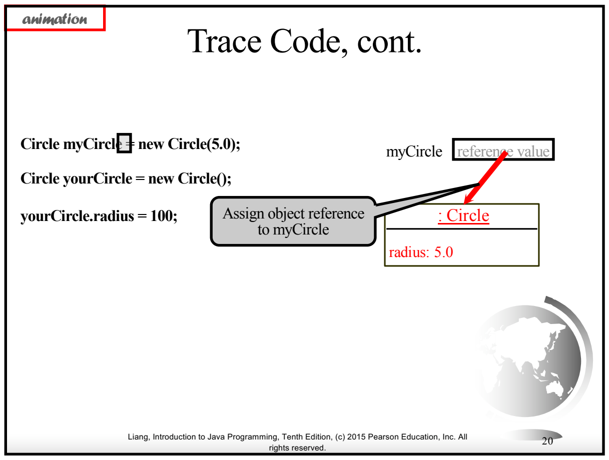




animation

Trace Code, cont.

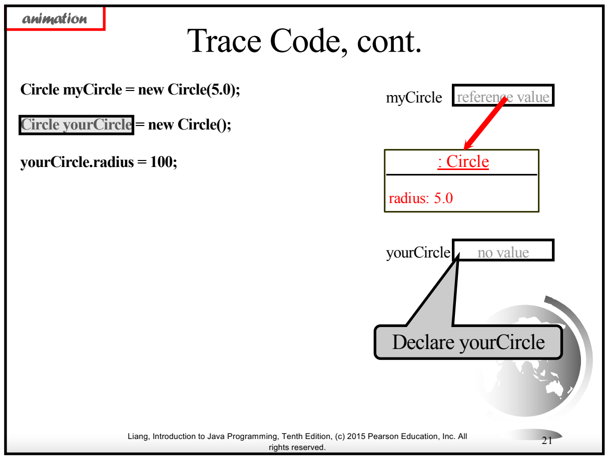
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

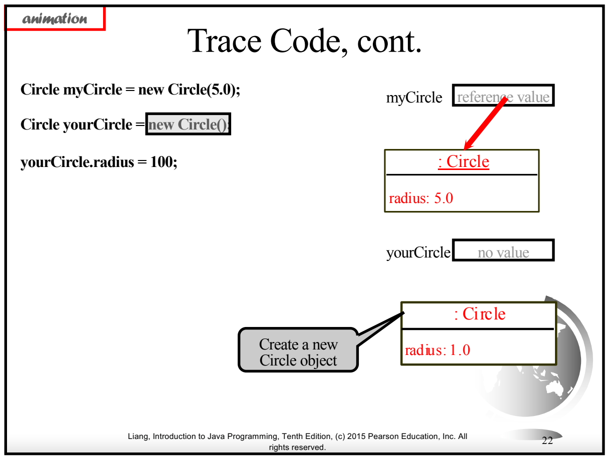
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

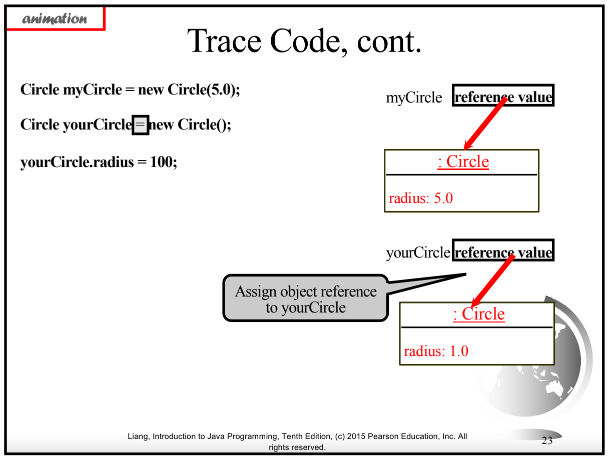
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

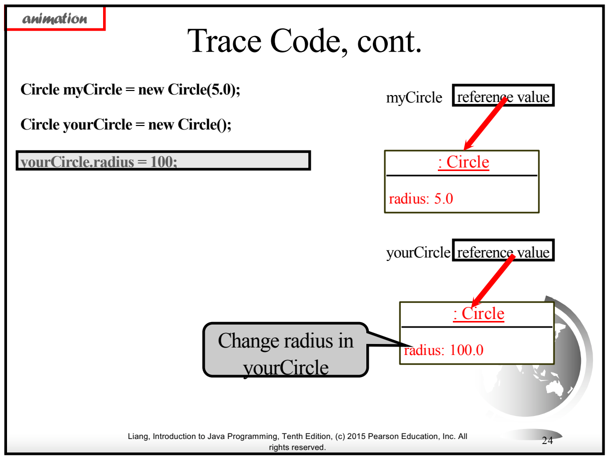
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



animation

Trace Code, cont.

```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

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

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; // c 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, 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);
  }
}
```

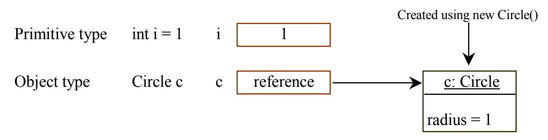
Example

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

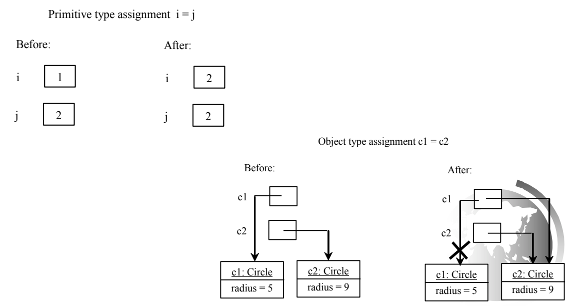
```
public class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```

Compile error: variable not initialized

Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types



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.

Garbage Collection, cont

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 + 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`.

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

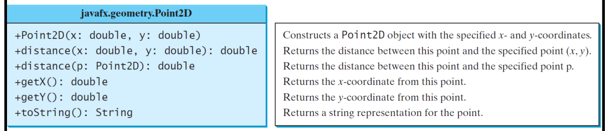
If two Random objects have the same seed, they will generate identical sequences of numbers. For example, the following code creates two Random objects with the same seed 3.

```
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 **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.

TestPoint2D Run

Instance Variables, and Methods

Instance variables belong to a specific instance.

Instance methods are invoked by an instance of the class.

Instance variables and methods are specified by omitting the **static** keyword.

Static Variables, Constants, and Methods

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

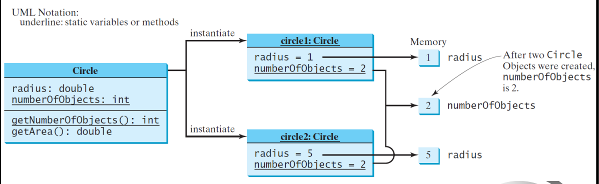
Static methods are not tied to a specific object.

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 Variables, Constants, and Methods, cont.



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

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.

```
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.

```
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.

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());
  }
}
```

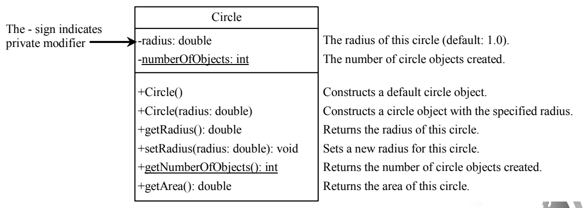
(b) This is wrong because x and convert are private in class C.

Why Data Fields Should Be private?

To protect data.

To make code easy to maintain.

Example of Data Field Encapsulation



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

CircleWithPrivateDataFields

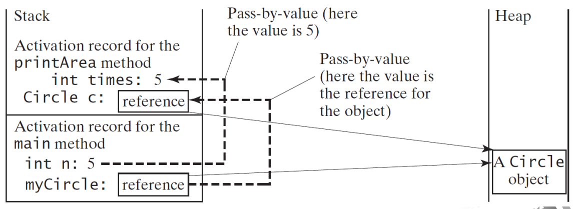
TestCircleWithPrivateDataFields Run

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)

TestPassObject Run

Passing Objects to Methods, cont.



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

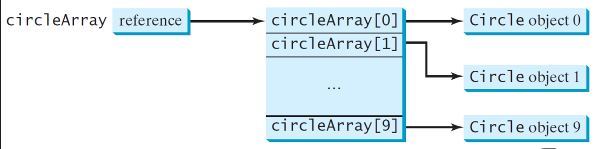
Array of Objects

```
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];
```



Array of Objects, cont.

Summarizing the areas of the circles

TotalArea Run

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*. 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!
  }
}
```

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.

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

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

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

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

```
Suppose that f1 and f2 are two objects of F.
F f1 = new F(); F f2 = new 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
```

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