02 - Objects and Classes Part 2

CS202: Introduction to Object Oriented Programming

Victor Mejia
CSULA

Slides adapted from Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

Today's Topics:

- *Quick Recap - Defining Classes For Objects*
- *Quick Recap - Accessing Objects via Reference Variables*
- Default Values
- Using Classes from the Java Library
- Static Variables, Constants, and Methods
- Visibility Modifiers
- Data Field Encapsulation
- Scope of Variables
- The **this** Reference

Quick Recap -
Defining Classes for Objects

OOP Concepts

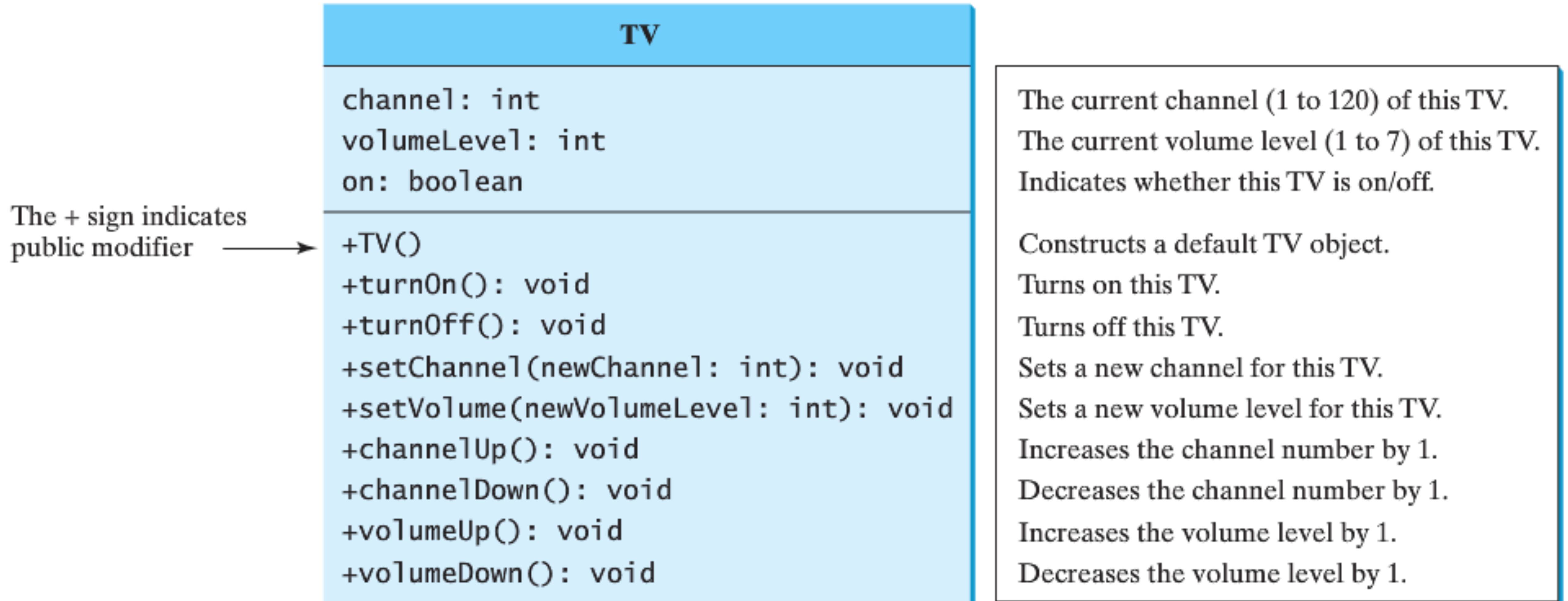
✓ state

- referred to as **properties, attributes, data fields**
- these properties contain information about the object, or **instance**
- Example: A **Person** object has **name** and **age**

OOP Concepts

✓ behavior
  - referred to as **actions**
  - defined by **methods** on the object
  - methods are **invoked** on an object
  - these methods perform an action on the object
  - Example (*getters*): A **Circle** object has methods **getArea()**, **getPerimeter()**
  - Example (*setters*): A **Circle** object has methods **setRadius(radius)**

Example: TV Class



TV.java
TestTV.java

Quick Recap -
Accessing Objects via Reference Variables

Declaring Object Reference Variables

A class is a *reference type*

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

To declare a reference variable, use the syntax:

```
ClassName objectRefVar;
```

Example:

```
Circle myCircle;
```

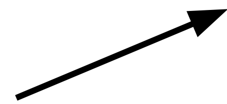
Declaring/Creating Objects in a Single Step

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

Example:

Create an object

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



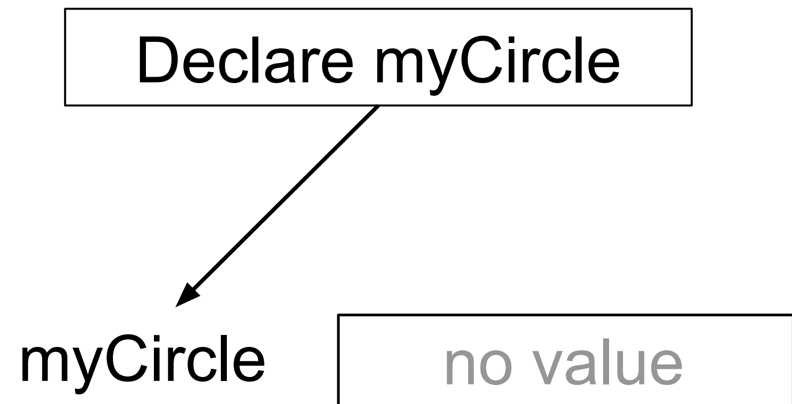
Assign object reference

"`myCircle` is a variable that contains a reference to a `Circle` object" (that's the technical description)

It's OK to say that `myCircle` is a `Circle` object

Trace Code

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

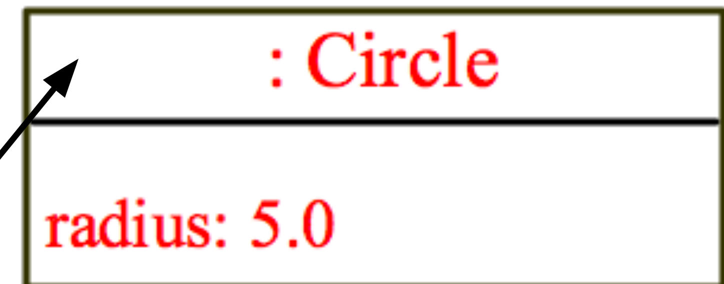


Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

myCircle

no value



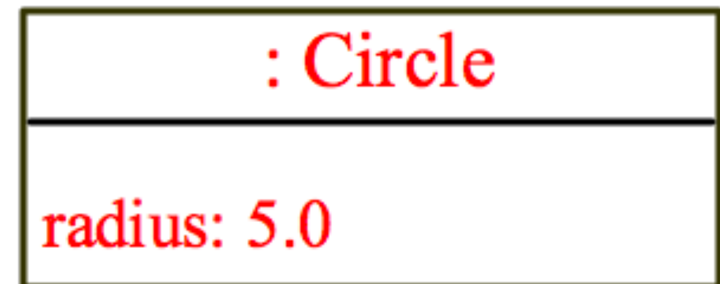
Create a circle

Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

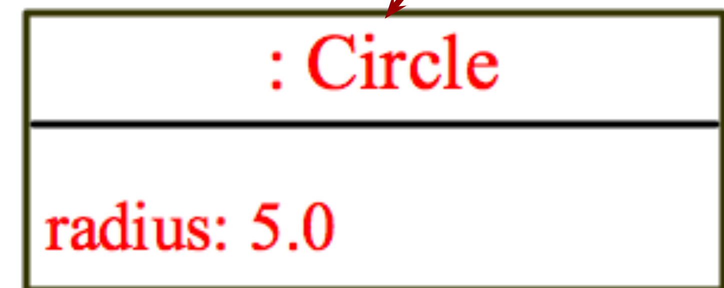
Assign object reference to myCircle

myCircle reference value



Trace Code, cont.

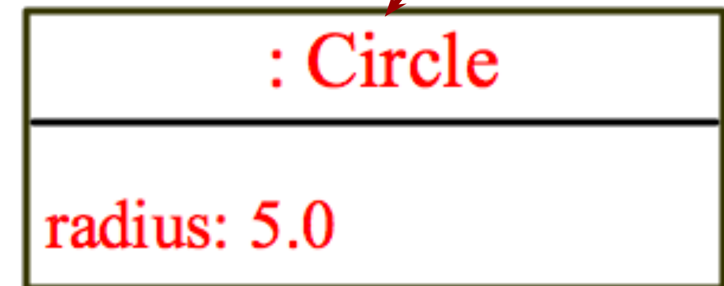
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```



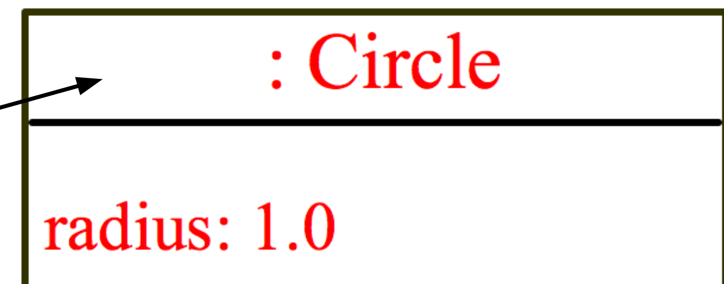
Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

myCircle reference value



yourCircle no value

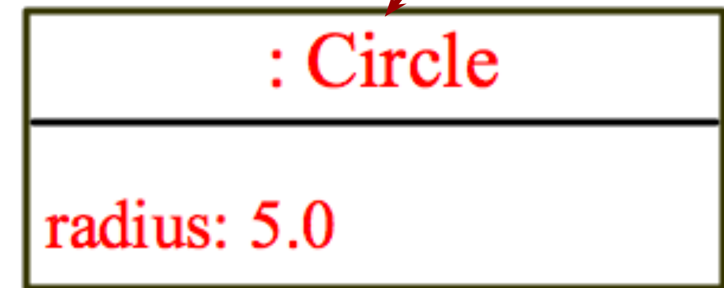


Create a new Circle object

Trace Code, cont.

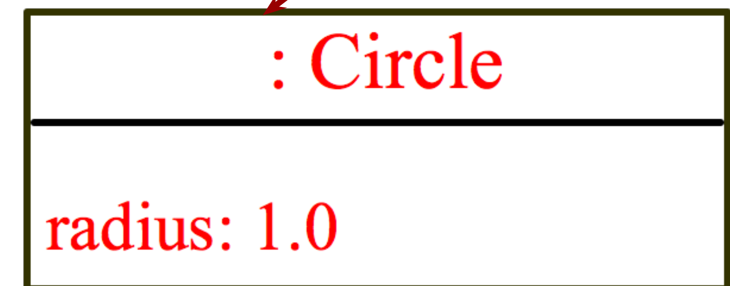
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

myCircle | reference value



yourCircle | reference value

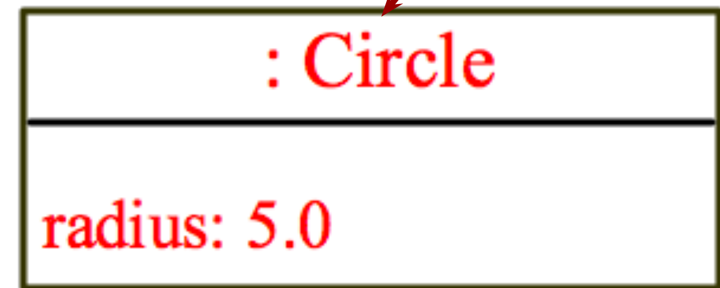
Assign object reference to yourCircle



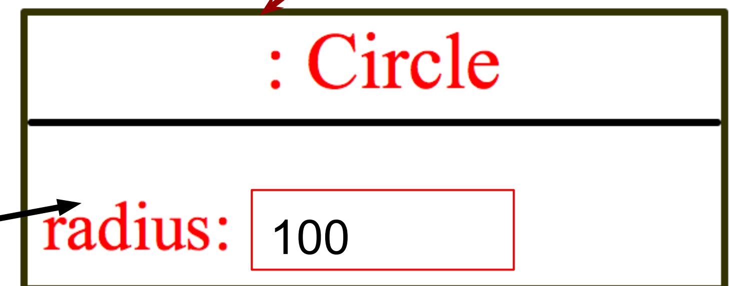
Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

myCircle | reference value

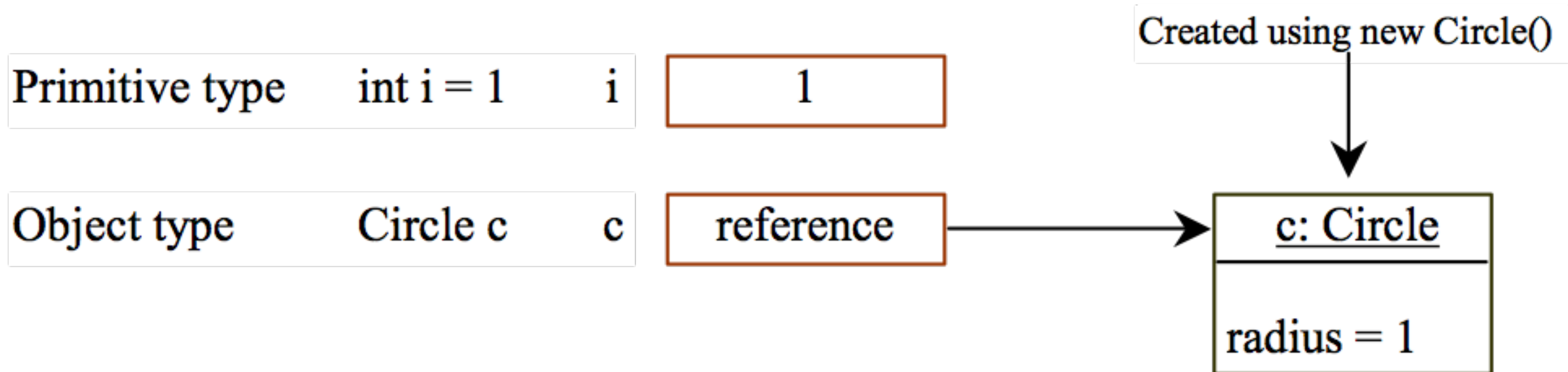


yourCircle | reference value



Change radius in
yourCircle

Differences between Variables of Primitive Data Types and Object Types



Example: Primitive vs. Object Types

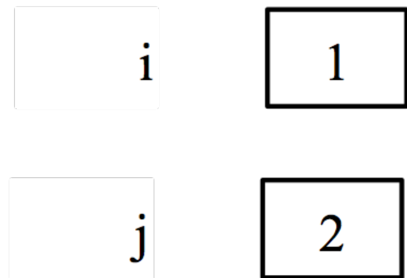
Student.java
TestObjectTypes.java

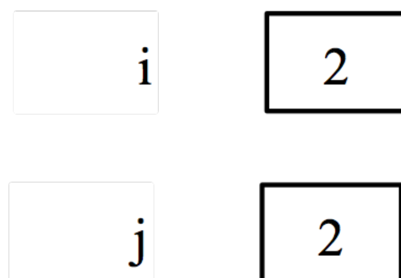
Copying Variables of Primitive Data Types and Object Types

Primitive type assignment i = j

Before:

After:

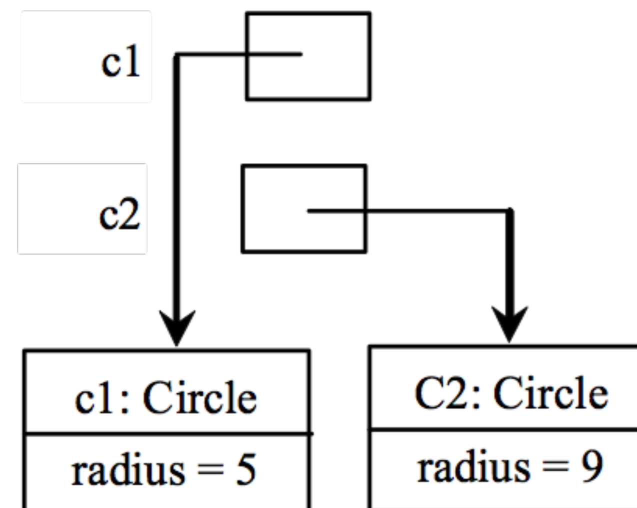


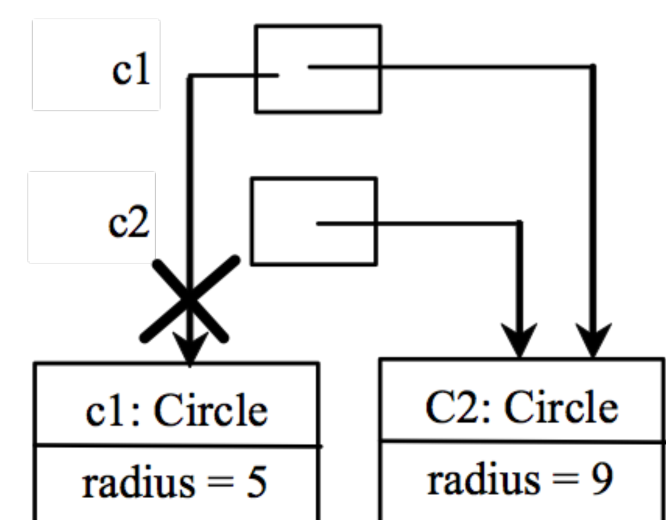


Object type assignment c1 = c2

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.

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.

Default Values

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

- 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

```
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);
  }
}
```

Example: Default Values

`Student.java`

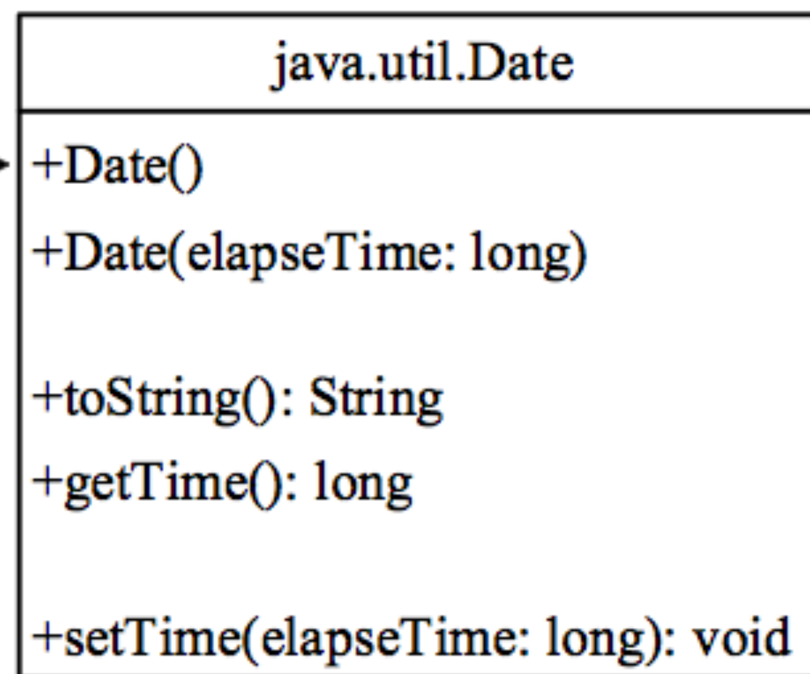
`TestStudentDefaultValues.java`

Using Classes from the Java Library

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



Constructs a Date object for the current time.

Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.

Returns a string representing the date and time.

Returns the number of milliseconds since January 1, 1970, GMT.

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 <u>`Sun Mar 09 13:50:19 EST 2003`</u>

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.java

http://docs.oracle.com/javase/8/javafx/api/javafx/geometry/Point2D.html

Static Variables, Constants, and Methods

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()`?
  - No! All the methods used before this chapter are **static methods**, which are defined using the `static` keyword.
  - `getArea()` is non-static. It must be invoked from an object using

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

Instance Variables, and Methods

Instance variables belong to a specific instance.

Instance methods are invoked by an instance of the class.

Static Variables, Constants, and Methods

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

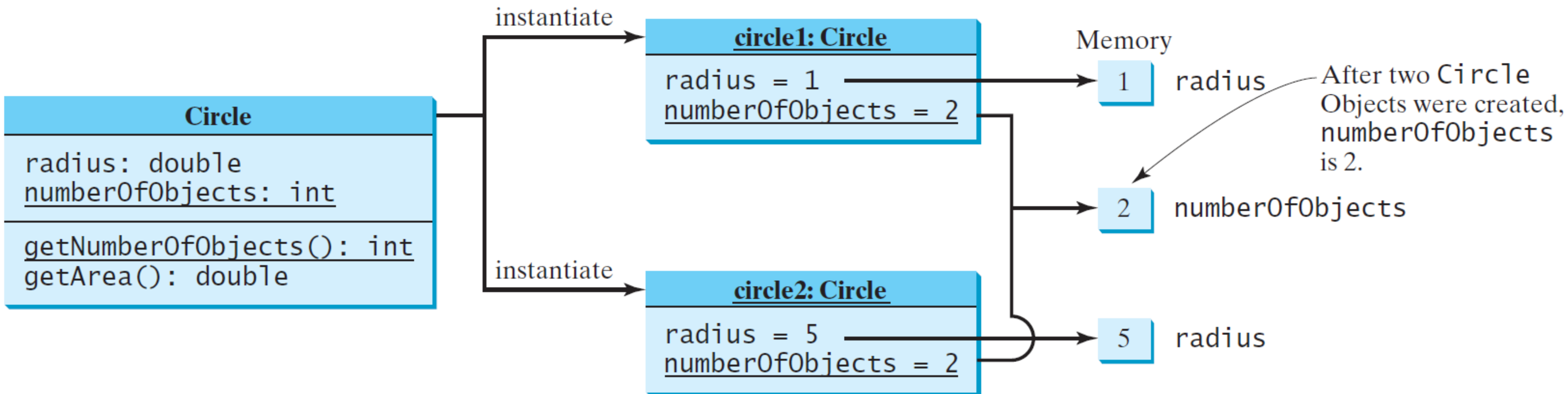
Static methods are not tied to a specific object.

Static constants (use `final`) are final variables shared by all the instances of the class.

Static Variables, Constants, and Methods

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

Static Variables, Constants, and Methods



Example: Static Variables, Constants, and Methods

CircleWithStaticMembers.java

TestCircleWithStaticMembers.java

MathConstants.java

Visibility Modifiers

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;

class C1 {
  ...
}
```

```
package p1;

public class C2 {
  can access C1
}
```

```
package p2;

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

- public : unrestricted access
- private : restricted access to within a class
- default : restricted access to within a package

```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();
  }
}
```

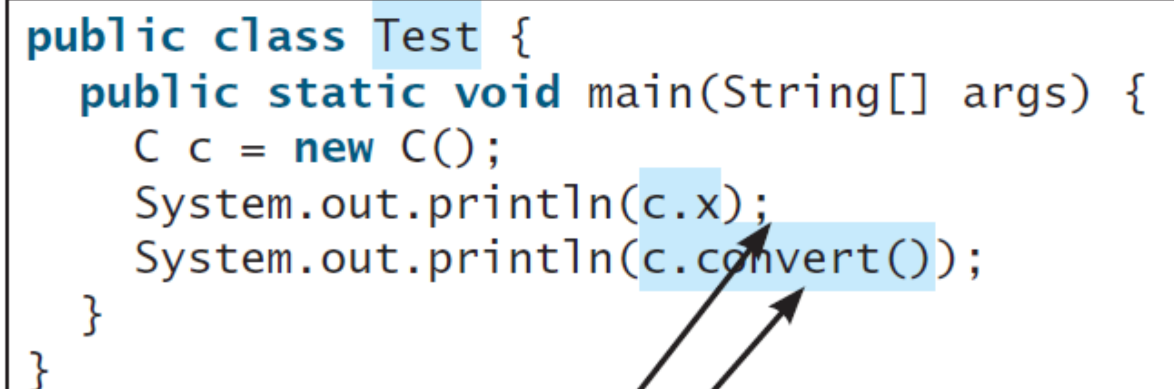
- public : unrestricted access
- private : restricted access to within a class
- default : restricted access to within a package

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?

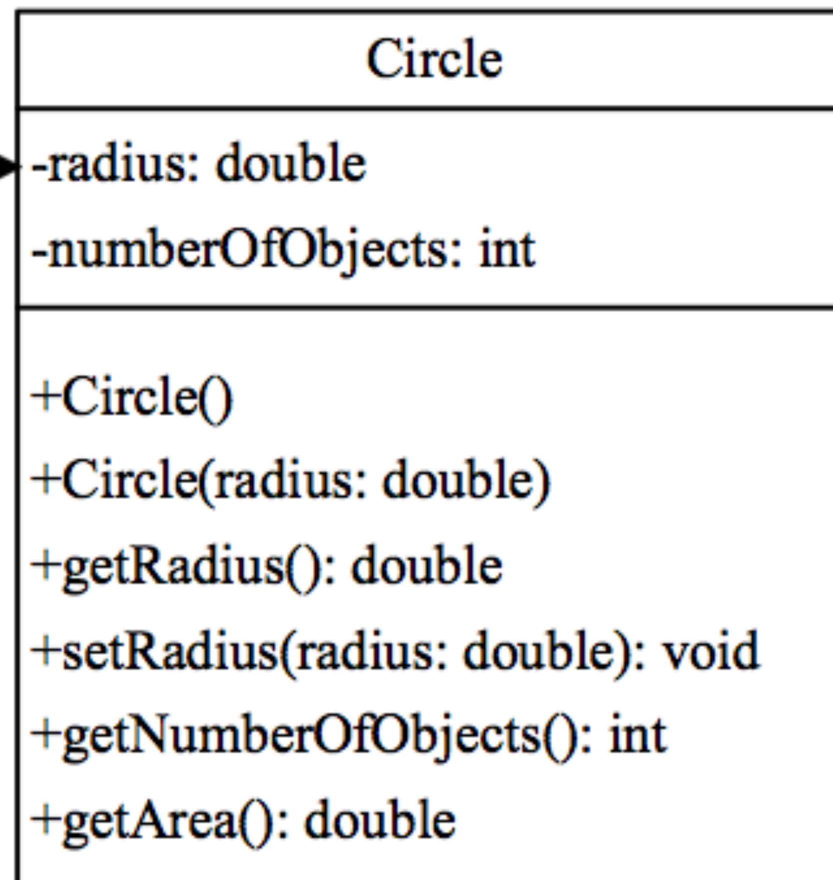
To protect data.

To make code easy to maintain.

Data Field Encapsulation

Example of Data Field Encapsulation

The - sign indicates private modifier



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.java
TestCircleWithPrivateDataFields.java

Scope of Variables

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

- 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.

The `this` Keyword

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

(a)

Equivalent

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

  ...

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

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

(b)

Calling Overloaded Constructor

```
public class Circle {
  private double radius;

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

  public Circle() {
    this(1.0);
  }

  ...
}
```

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

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