# Introduction

# Introduction

- Using what you know, how would we model a Student in the GET System?

```
public class StudentGETSample {
  public static void main(String[] args) {
    // student 1
    String name = "Buzz Lightyear";
    int age = 20;
    boolean isGraduateStudent = false;

    System.out.println("Name: " + name + " Age: " + age +
      " Graduate Student: " + isGraduateStudent);
  }
}
```

# Introduction

- This code was just for one student
- what about thousands of students?

```
public class StudentGETSample {
  public static void main(String[] args) {
    // student 1
    String name = "Buzz Lightyear";
    int age = 20;
    boolean isGraduateStudent = false;

    System.out.println("Name: " + name + " Age: " + age +
      " Graduate Student: " + isGraduateStudent);
  }
}
```

# Introduction

- OOP solves this kinds of problems and provides numerous benefits using **objects**
  - **Modularity**: source code maintained independently, easily passed around in the system
  - **Information-hiding**: details of internal implementation hidden from the outside world
  - **Code re-use**: use other software developer's code
  - **Pluggability and debugging ease**: remove a particular object and plug in your own

Source: http://docs.oracle.com/javase/tutorial/java/concepts/object.html

# Defining Classes for Objects

# 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.
  - 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**.

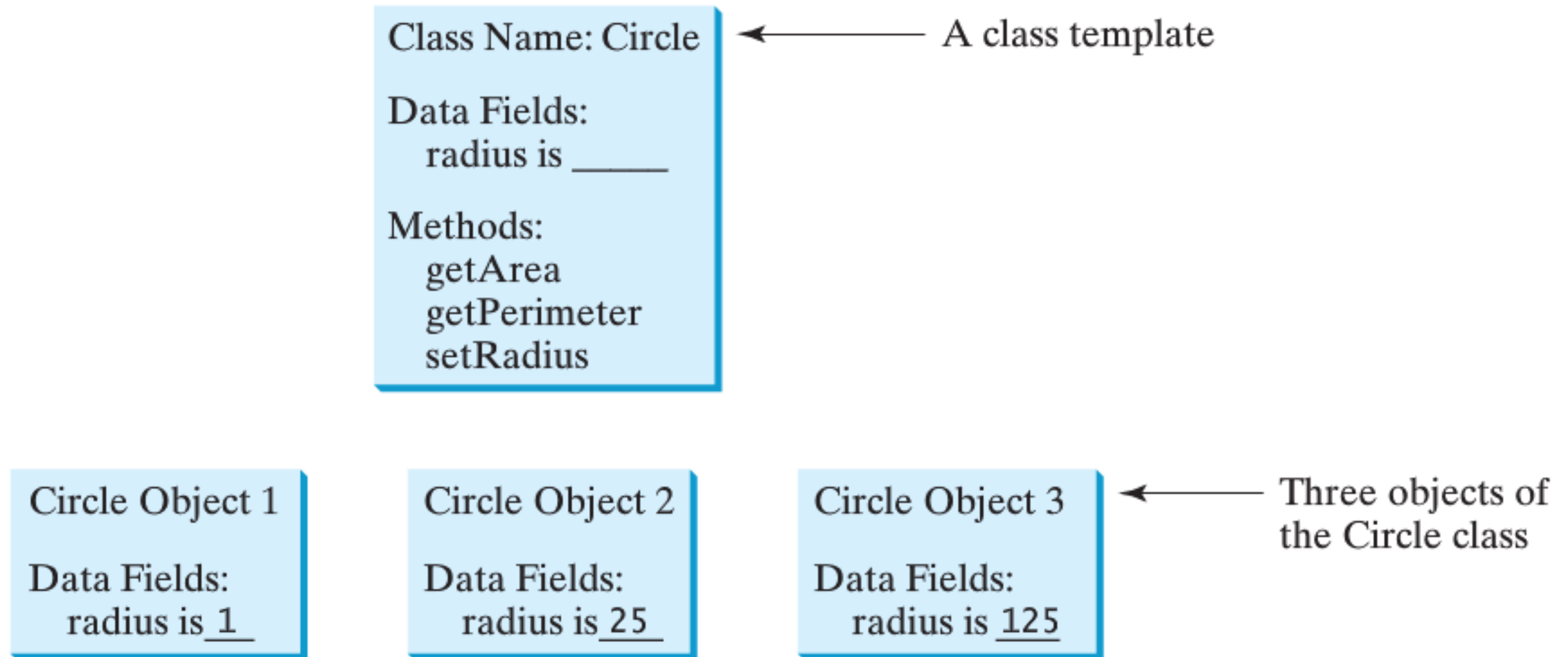
# OOP Concepts

✓ 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** properties

# 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)**

# Objects




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

# OOP Concepts - Netflix

| Object | States | Behaviors |
|---|---|---|
| Movie | title, genre, year, length, rating | set the name, set the rating, play movie |
| Actor | name, age, movies appeared in | set the name, find movies actor appeared in |
| CreditCard | number, expiration date, name, zip code, security code | charge amount, refund amount, get card number |

# OOP Concepts - GET System

| Object | States | Behaviors |
| --- | --- | --- |
| Student | name, age, CIN, status, major, standing | add class, retrieve current schedule |
| Instructor | name, age, payroll, courses taught, department | drop student, set grade |
| Course | course id, capacity, prerequisites, time and date, instructor | add student, set time, set permissions |

# Classes

*Classes* are constructs that define objects of the same type.

a template, blueprint

an object is an **instance** of a class

object → class, apple pie → recipe

many apple pies from a recipe

many objects from a single class

# Classes

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/instances from the class.

# Unified Modeling Language (UML)

**data field:**

```
dataFieldName : dataFieldType
```

**method:**

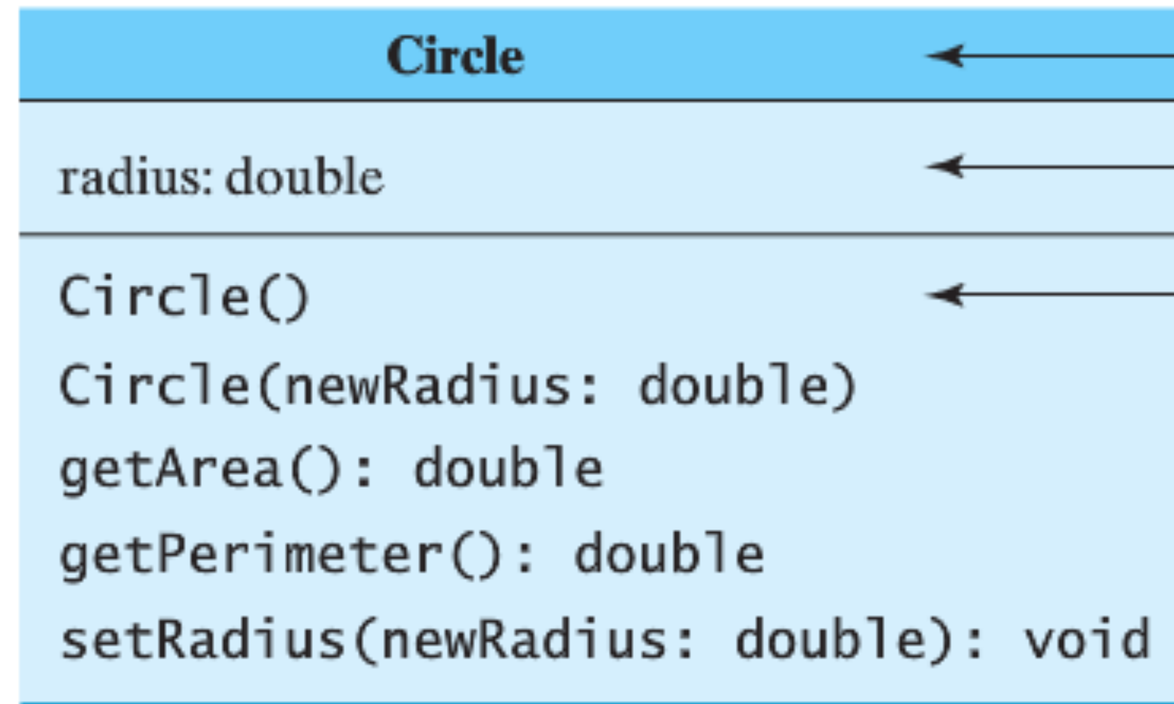
```
methodName(parameterName: parameterType) : returnType
```

**constructor:**

```
ClassName(parameterName: parameterType)
```

# UML Class Diagram

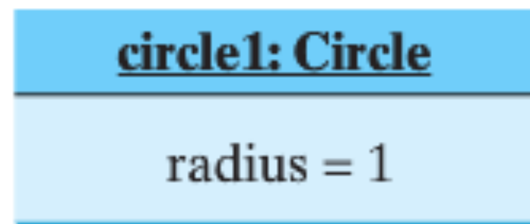
UML Class Diagram

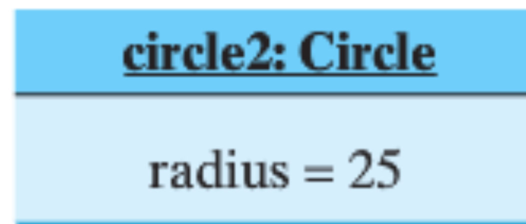


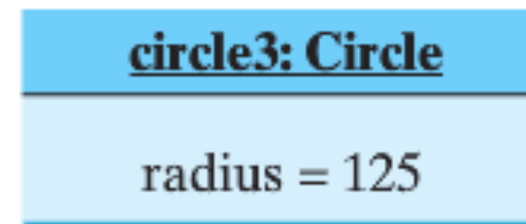

Class name

Data fields

Constructors and methods

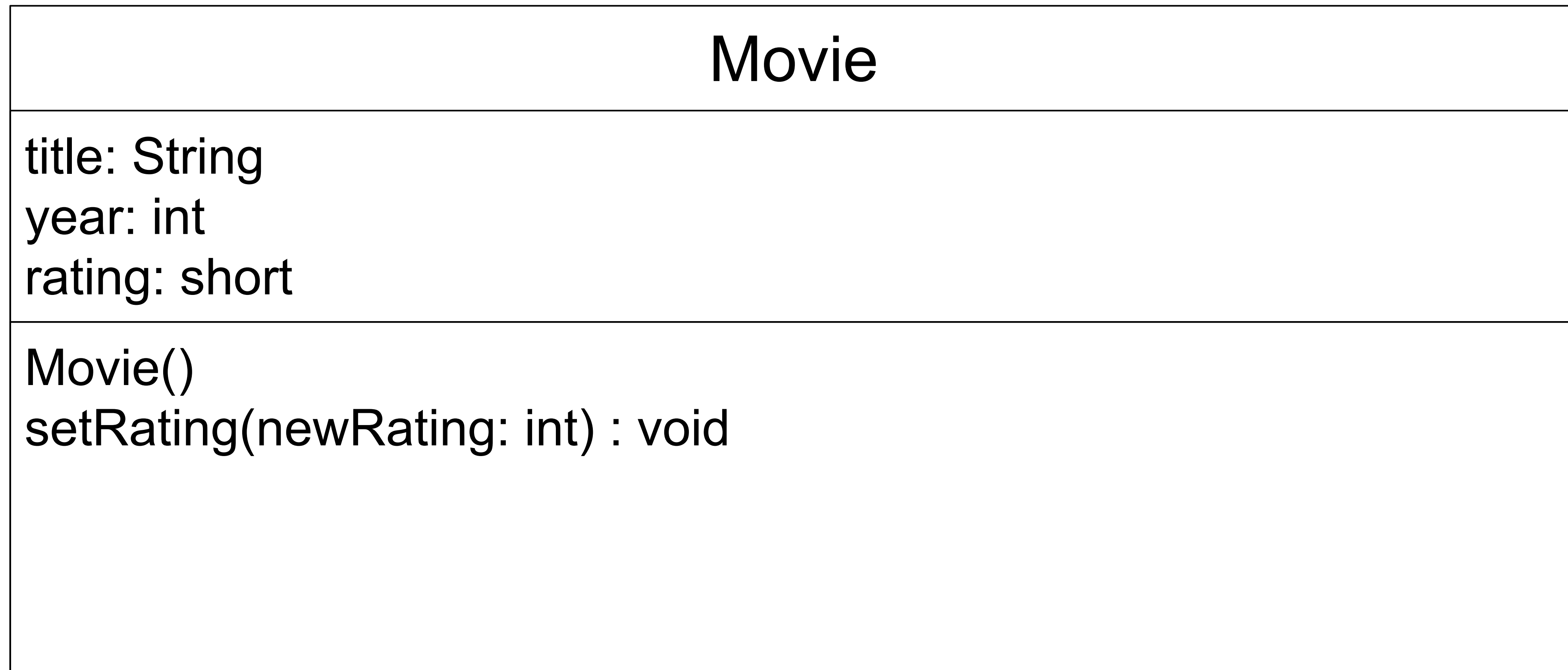



UML notation for objects

# UML Class Diagram



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

# Example: Defining Classes and Creating Objects

`TestSimpleCircle.java`

`SimpleCircle.java`

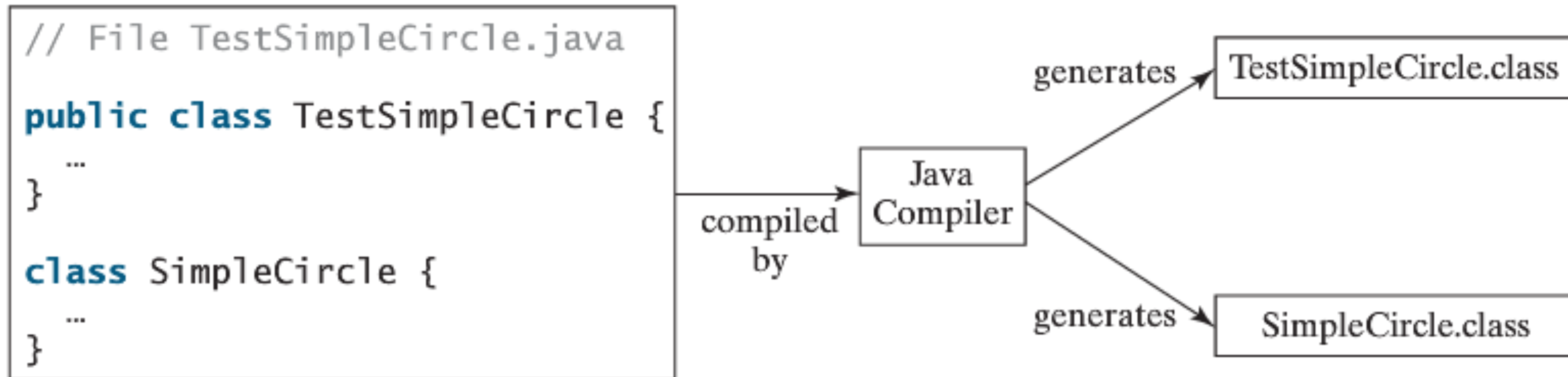


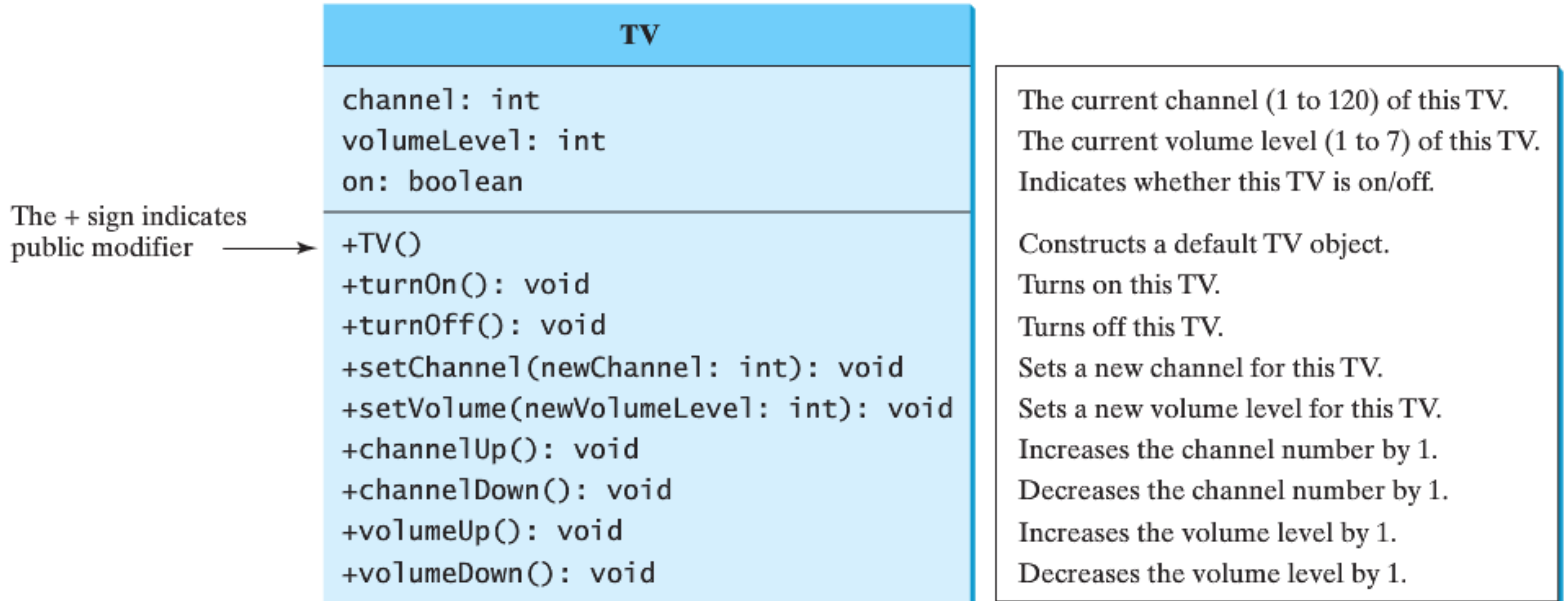

**FIGURE 9.5** Each class in the source code file is compiled into a **.class** file.

You can put 2 classes into one file, but **only one** can be a **public** class

the **public** class must have the same name as the file name

you can define a class with a main method (see **SimpleCircle.java**) for testing purposes

# Example: TV Class




TV.java
TestTV.java

# Packages

A package is a namespace that organizes a set of related classes and interfaces (covered later)

like different folders in your computer

software can contain hundreds or thousands of classes, so it makes sense to place them in a package

Source: [http://docs.oracle.com/javase/tutorial/java/concepts/package.html](http://docs.oracle.com/javase/tutorial/java/concepts/package.html)

# Constructing Objects Using Constructors

# Constructors

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

```
Circle() {
}

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

# Constructors, cont.

- must have the same name as the class itself
- doesn’t have a return type—not even `void`
- invoked using the `new` operator when an object is created
- plays the role of initializing objects
- A constructor with no parameters is referred to as a *no-arg constructor*

# 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*.

**Always, always specify a constructor**

# 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;
```

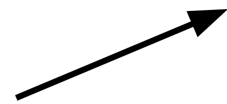
# Declaring/Creating Objects in a Single Step

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

Example:

Create an 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

# Accessing Object's Members

- object's members are
  - data fields
  - methods

- Referencing the object's data:

```
objectRefVar.data
```

*e.g.,* `myCircle.radius`

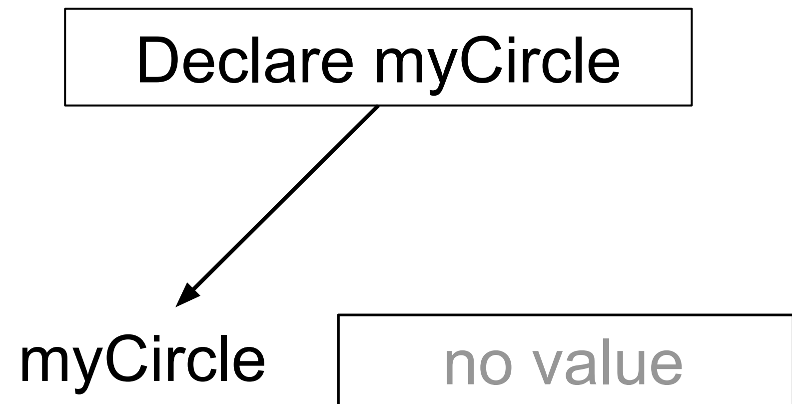
- Invoking the object's method:

```
objectRefVar.methodName(arguments)
```

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

# Trace Code

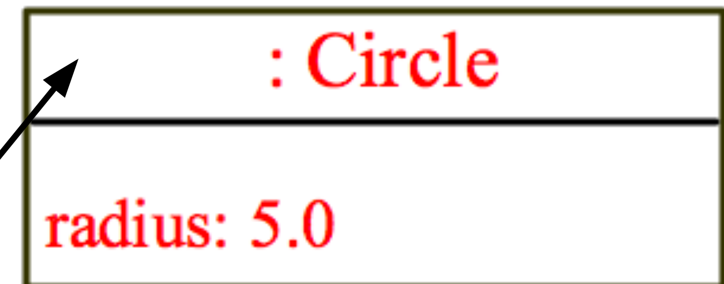
```
Circle myCircle = new Circle(5.0);

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



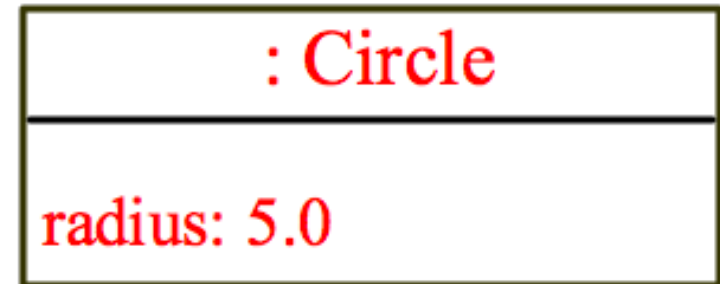

Create a circle

# Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

myCircle | reference value

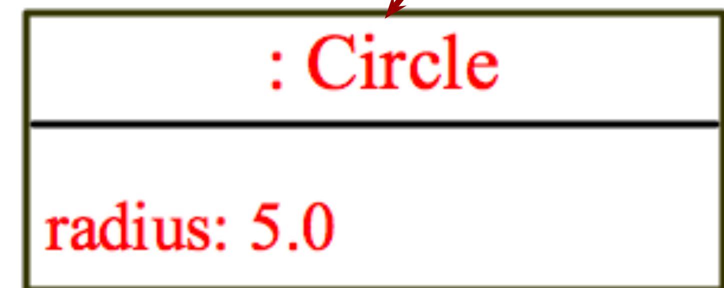
Assign object reference to myCircle



# Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

myCircle reference value



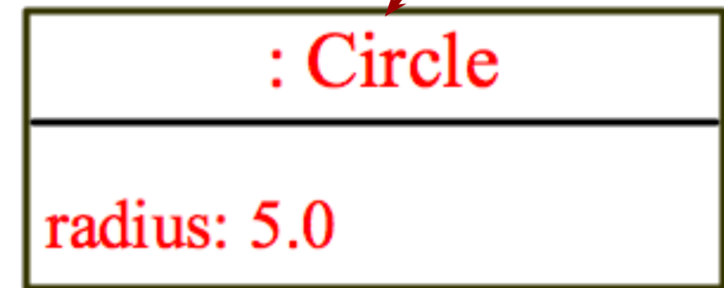

yourCircle no value

Declare yourCircle

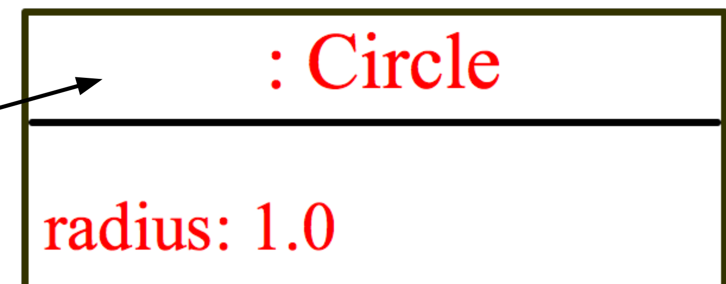
# Trace Code, cont.

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

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

myCircle | reference value




yourCircle | no value

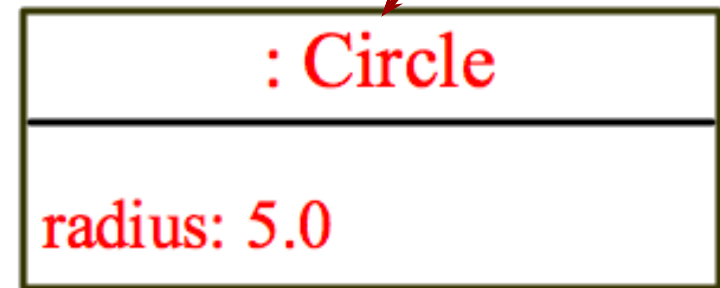



Create a new Circle object

# Trace Code, cont.

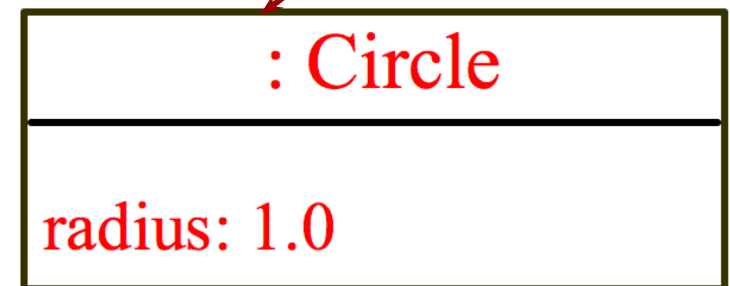
```
Circle myCircle = new Circle(5.0);

Circle yourCircle = new Circle();

yourCircle.radius = 100;
```

myCircle reference value




yourCircle reference value

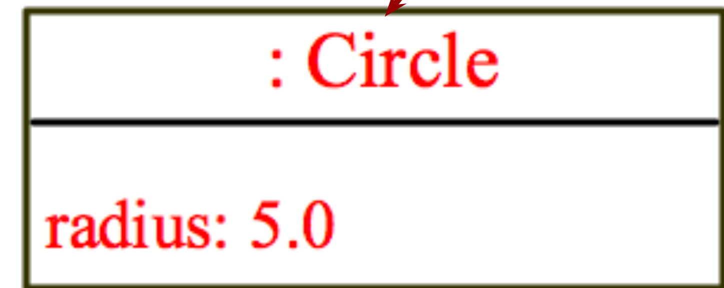
Assign object reference to yourCircle



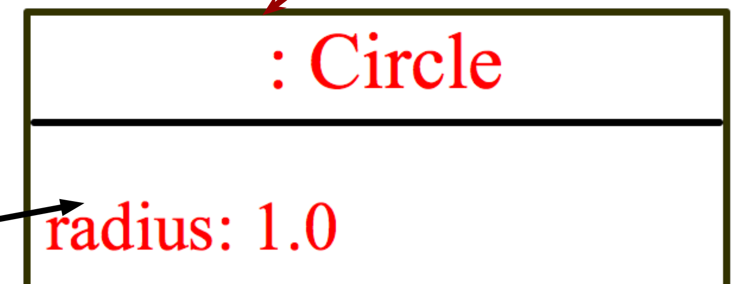
# Trace Code, cont.

```
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