NCP2103: Object-Oriented Programming
(Java Programming)

Methods, Classes, and Objects,

Module 3

Errol John M. Antonio
Assistant Professor
Computer Engineering Department
University of the East – Manila Campus

Copyright belongs to Farrell, J. (2016). Java Programming. 8th Edition. Course Technology, Cengage Learning.

Learning Objectives

At the end of this module, students shall be able to:

- Create methods
- Add parameters to methods
- Create methods that return values
- Study class concepts
- Create a class
- Create instance methods in a class

Learning Objectives

At the end of this module, students shall be able to:

- Declare objects and use their methods
- Organize classes
- Create constructors
- Appreciate classes as data types

Module Outline

I: Creating Methods

- **Method**
  - A program module that contains series of statements that carries out task.
  - **`main()`** – execute automatically when you run a program
- Execute method
  - **Invoke** or **call** from another method
- Calling method (client method)
  - Makes method call
- Called method
  - Invoked by calling method

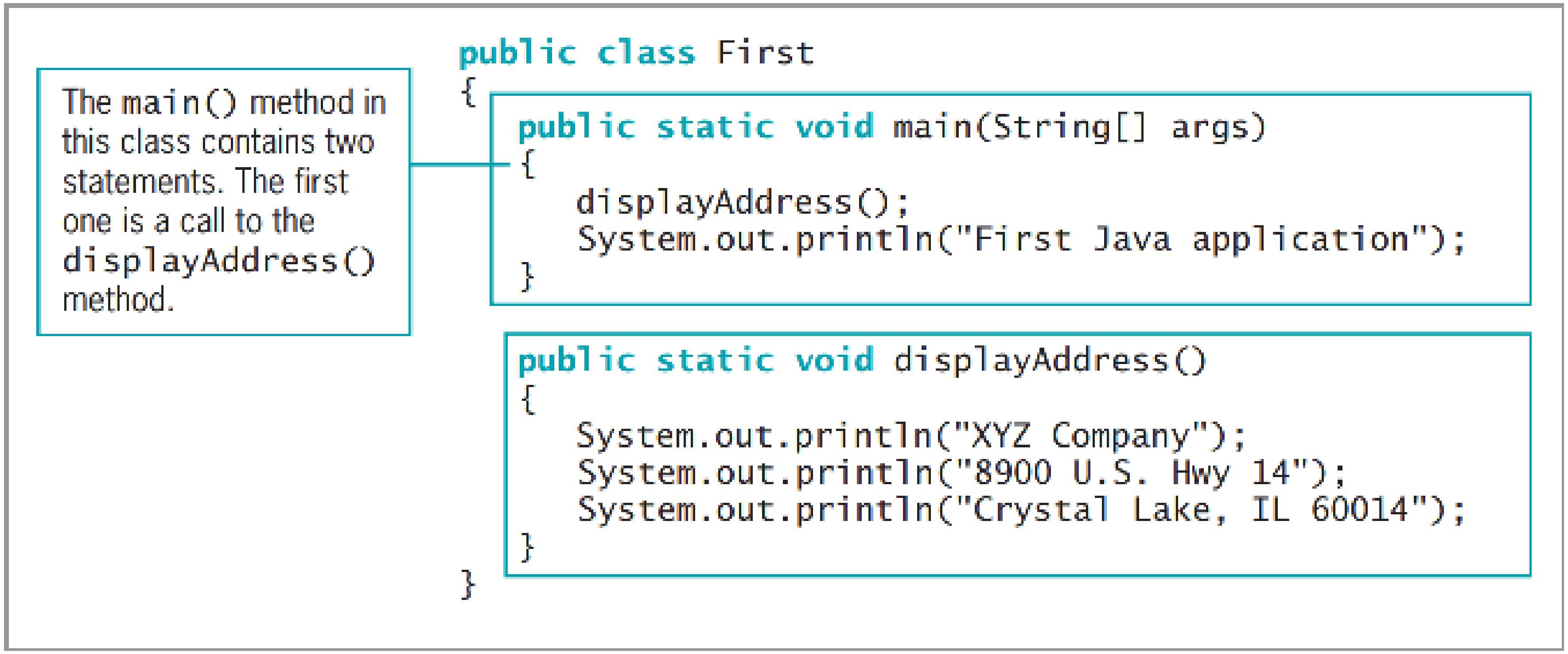
Creating Methods

- Method must include:
  - **Method header**
    - Also called declaration
  - **Method body**
    - Between a pair of curly braces
    - Contains the statements that carry out the work
    - Also called implementation

Creating Methods

- **Method Declaration**
  - Optional access specifiers
  - Return type for method
  - Method name
  - Set of parentheses
    - Might contain data to be sent to the method
- Place entire method within class that will use it
  - Not within any other method
- Fully qualified identifier
  - Complete name that includes the class

Creating Methods



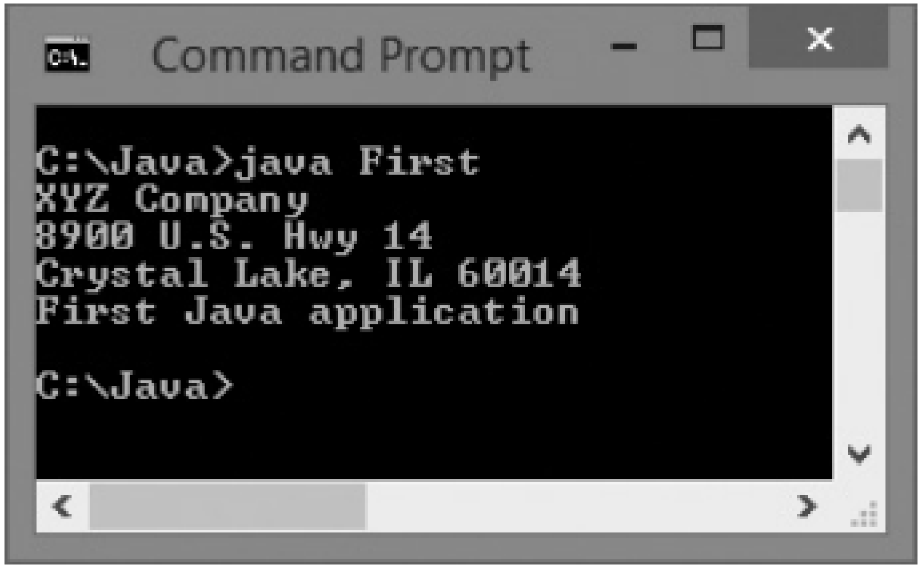
The main() method in this class contains two statements. The first one is a call to the displayAddress() method.

```
public class First
{
   public static void main(String[] args)
   {
      displayAddress();
      System.out.println("First Java application");
   }

   public static void displayAddress()
   {
      System.out.println("XYZ Company");
      System.out.println("8900 U.S. Hwy 14");
      System.out.println("Crystal Lake, IL 60014");
   }
}
```

Figure 3-4 First class with main() calling displayAddress()

Creating Methods


Command Prompt

```
C:\Java>java First
XYZ Company
8900 U.S. Hwy 14
Crystal Lake, IL 60014
First Java application

C:\Java>
```

Figure 3-5 Output of the `First` application, including the `displayAddress()` method

Creating Methods

```
public class First
{
   public static void main(String[] args)
   {
      nameAndAddress();
      System.out.println("First Java application");
   }
   public static void nameAndAddress()
   {
      System.out.println("Event Handlers Incorporated");
      System.out.println("8900 U.S. Hwy 14");
      System.out.println("Crystal Lake, IL 60014");
   }
}
```

Figure 3-4 First class with `main()` calling `nameAndAddress()`

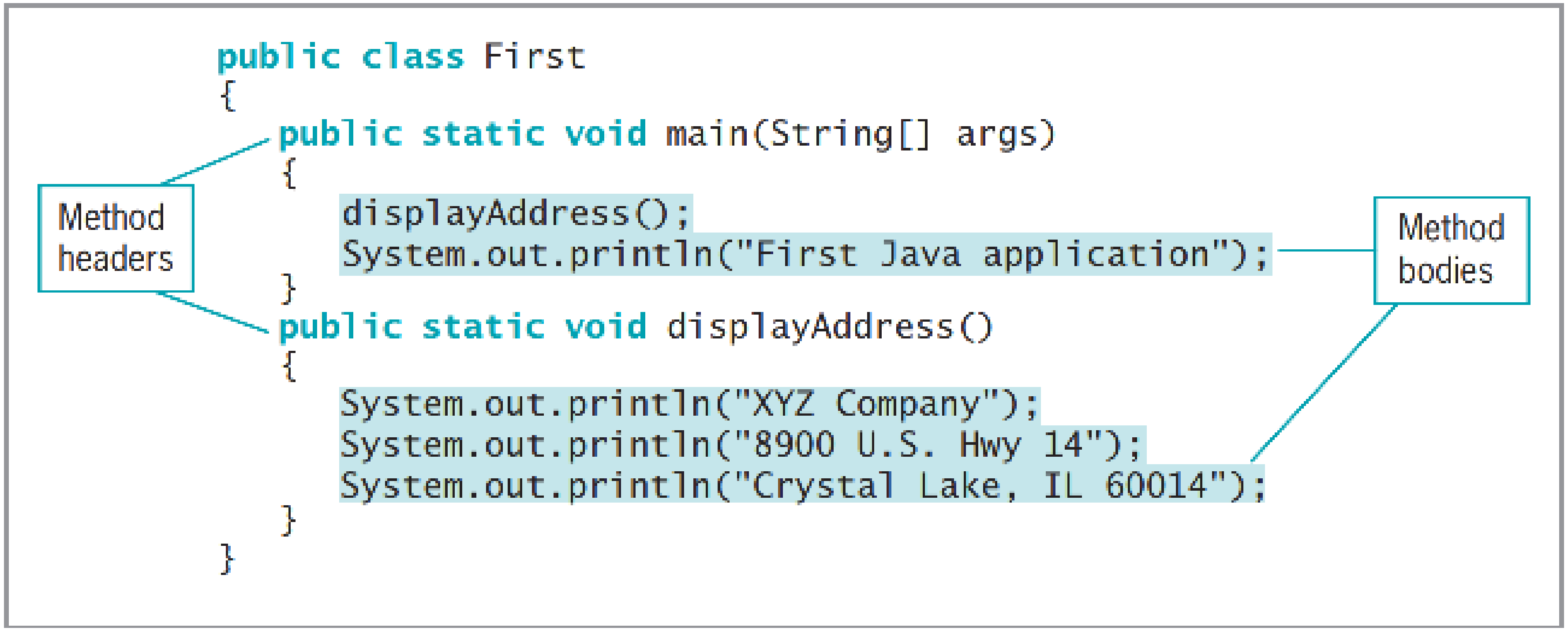
TWO TRUTHS AND A LIE

1. Any class can contain an unlimited number of methods.
2. During one program execution, a method might be called any number of times.
3. A method is usually written within another method.

II: Understanding Method Construction

- Every method must include the two parts:
  - **Method Header**
    - A method's header provides information about how other methods can interact with it. A method header is also called a **declaration**.
  - **Method Body**
    - Placed between a pair of curly braces—The method body contains the statements that carry out the work of the method. A method's body is called its **implementation**.

Understanding Method Construction



```
public class First
{
   public static void main(String[] args)
   {
      displayAddress();
      System.out.println("First Java application");
   }
   public static void displayAddress()
   {
      System.out.println("XYZ Company");
      System.out.println("8900 U.S. Hwy 14");
      System.out.println("Crystal Lake, IL 60014");
   }
}
```

Figure 3-6 The headers and bodies of the methods in the First class

Understanding Method Construction

- The method header is the first line of a method. It contains the following:
  - Optional access specifiers
  - A return type
  - An identifier
  - Parentheses

Understanding Method Construction

- The method header is the first line of a method. It contains the following:
  - **Optional access specifiers/modifiers**
    - `public`, `private`, `protected`, or, if left `unspecified`, package by default.
  - **A return type**
    - Describes the type of data the method sends back to its calling method. **Not all methods return a value to their calling methods**; a method that returns no data has a return type of **void**.

Understanding Method Construction

- An identifier
  - A method's name can be any legal identifier.
- Parentheses
  - The parentheses might contain data to be sent to the method.

Understanding Method Construction

- The full name of the **`displayAddress()`** method is **`First.displayAddress()`**, which includes the class name (First), a dot, and the method name, which is displayAddress().
- (The name does not include an object because displayAddress() is a static method.)
- When you use a method within its own class, you do not need to use the fully qualified name (although you can); the simple method name alone is enough.

TWO TRUTHS AND A LIE

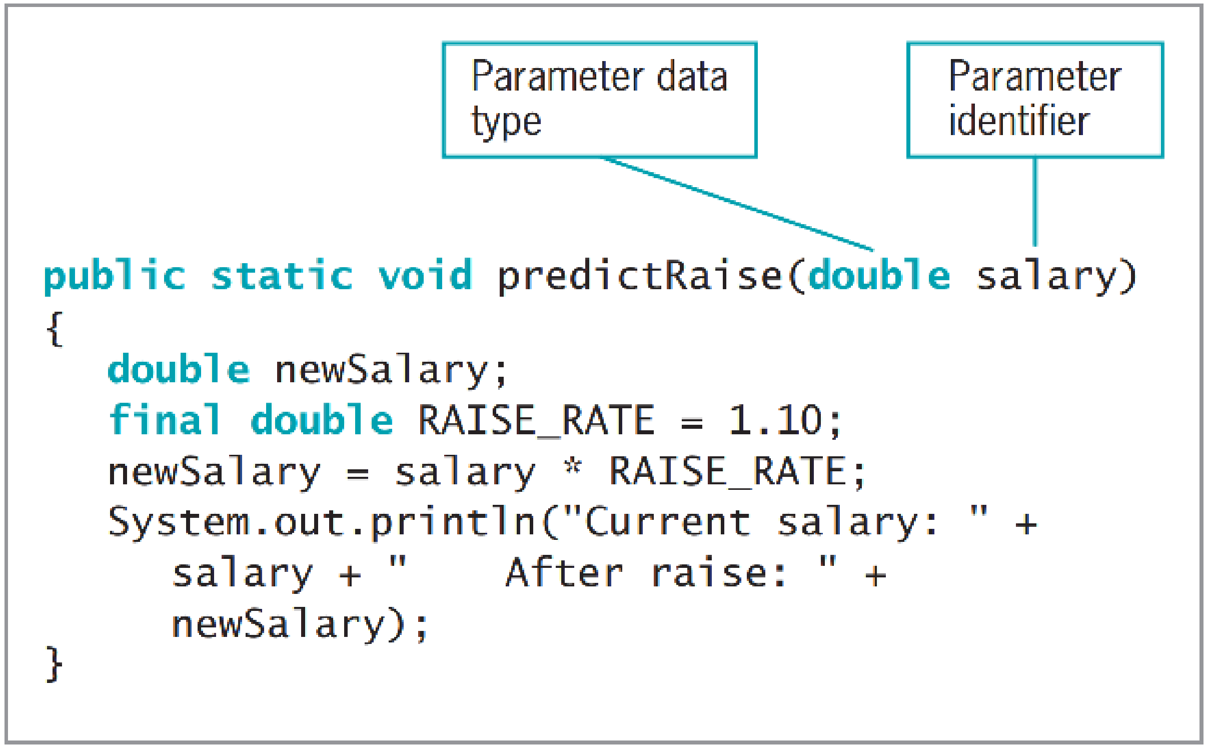
1. A method header is also called an implementation.
2. When a method is declared with public access, methods in other classes can call it.
3. Not all methods return a value, but every method requires a return type.

III: Adding Parameters to Methods

- **Arguments**
  - Data items you use in a call to a method
- **Parameters**
  - When the method receives the data items
- **Implementation hiding**
  - Encapsulation of method details within class
  - Calling method needs to understand only interface to called method
- Include within method declaration parentheses
  - Parameter type
  - Local name for parameter

Adding Parameters to Methods

- Creating a Method (Single Parameter)

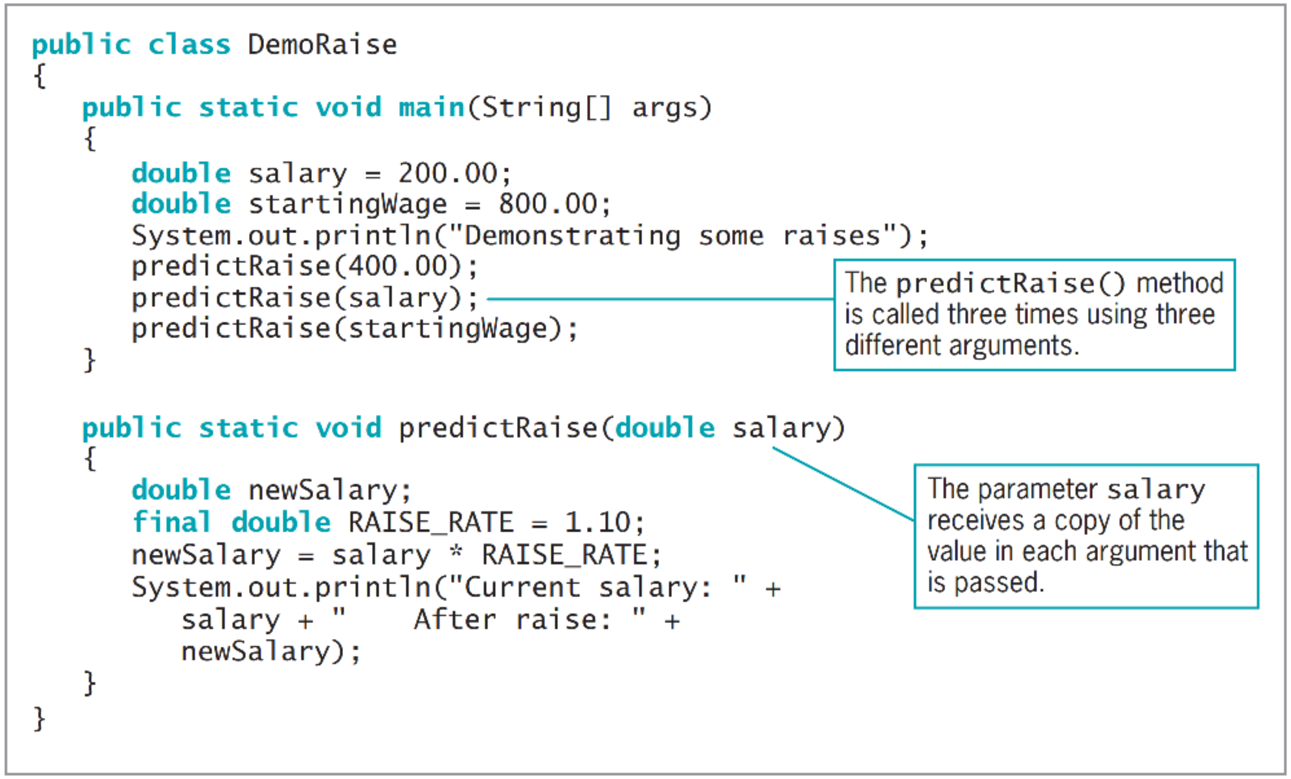


```
public static void predictRaise(double salary)
{
   double newSalary;
   final double RAISE_RATE = 1.10;
   newSalary = salary * RAISE_RATE;
   System.out.println("Current salary: " +
      salary + "     After raise: " +
      newSalary);
}
```

Figure 3-13 The `predictRaise()` method

Adding Parameters to Methods

- **Creating a Method (Single Parameter)**



```
public class DemoRaise
{
   public static void main(String[] args)
   {
      double salary = 200.00;
      double startingWage = 800.00;
      System.out.println("Demonstrating some raises");
      predictRaise(400.00);
      predictRaise(salary);
      predictRaise(startingWage);
   }

   public static void predictRaise(double salary)
   {
      double newSalary;
      final double RAISE_RATE = 1.10;
      newSalary = salary * RAISE_RATE;
      System.out.println("Current salary: " +
         salary + "    After raise: " +
         newSalary);
   }
}
```

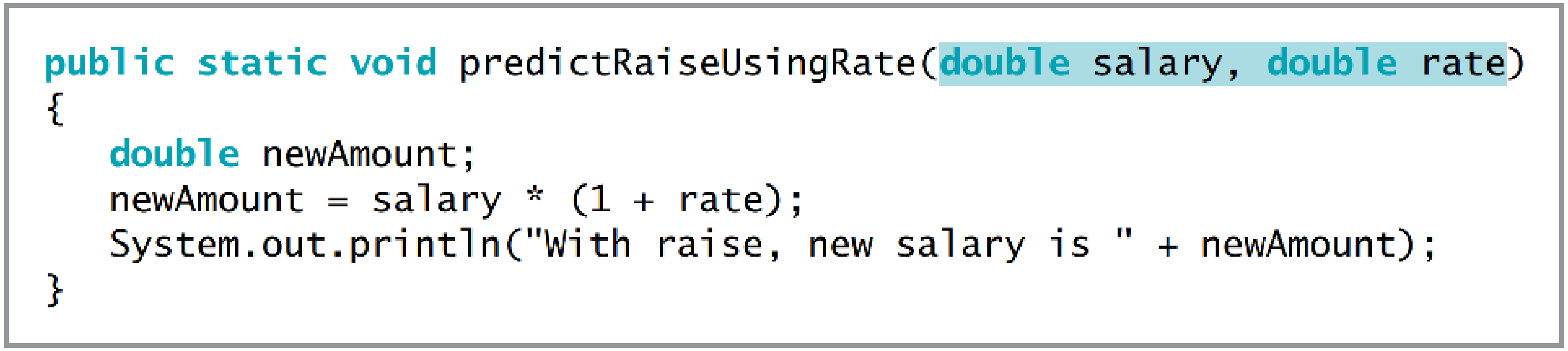
The `predictRaise()` method is called three times using three different arguments.

The parameter `salary` receives a copy of the value in each argument that is passed.

Figure 3-14 The `DemoRaise` class with a `main()` method that uses the `predictRaise()` method three times

Adding Parameters to Methods

- **Creating a Method (Multiple Parameter)**

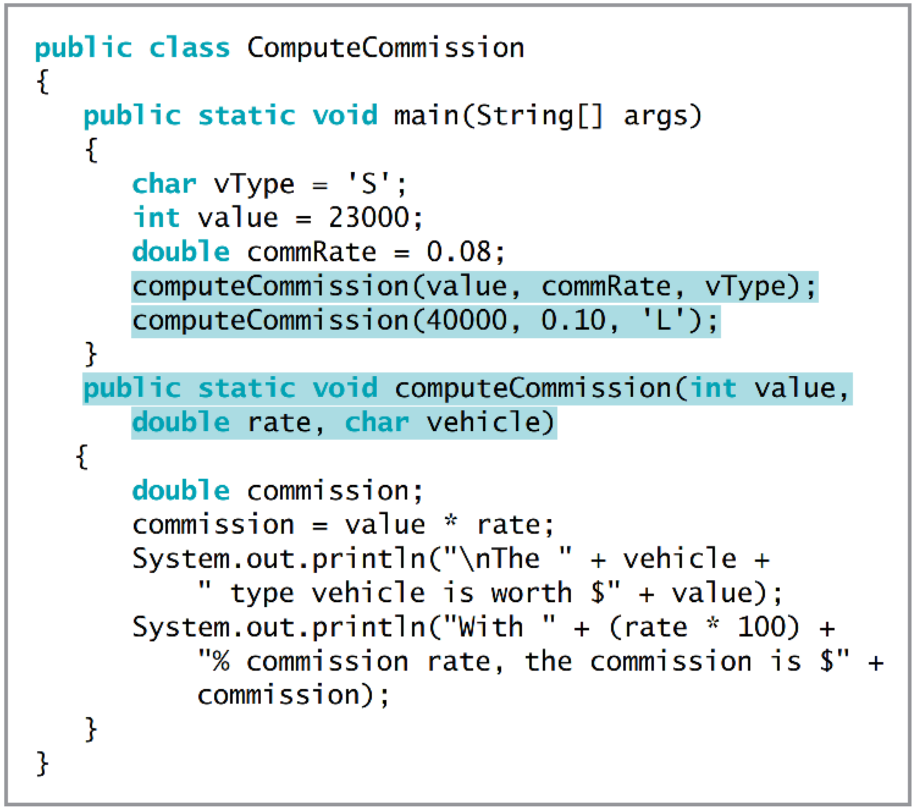


```
public static void predictRaiseUsingRate(double salary, double rate)
{
   double newAmount;
   newAmount = salary * (1 + rate);
   System.out.println("With raise, new salary is " + newAmount);
}
```

Figure 3-16 The `predictRaiseUsingRate()` method that accepts two parameters

Adding Parameters to Methods

- Creating a Method (Multiple Parameter)



```
public class ComputeCommission
{
   public static void main(String[] args)
   {
      char vType = 'S';
      int value = 23000;
      double commRate = 0.08;
      computeCommission(value, commRate, vType);
      computeCommission(40000, 0.10, 'L');
   }
   public static void computeCommission(int value,
      double rate, char vehicle)
   {
      double commission;
      commission = value * rate;
      System.out.println("\nThe " + vehicle +
         " type vehicle is worth $" + value);
      System.out.println("With " + (rate * 100) +
         "% commission rate, the commission is $" +
         commission);
   }
}
```

Figure 3-17 The ComputeCommission class

Adding Parameters to Methods

- **Local variable**
  - Known only within boundaries of method
  - Each time method executes
    - Variable redeclared
    - New memory location large enough to hold type set up and named

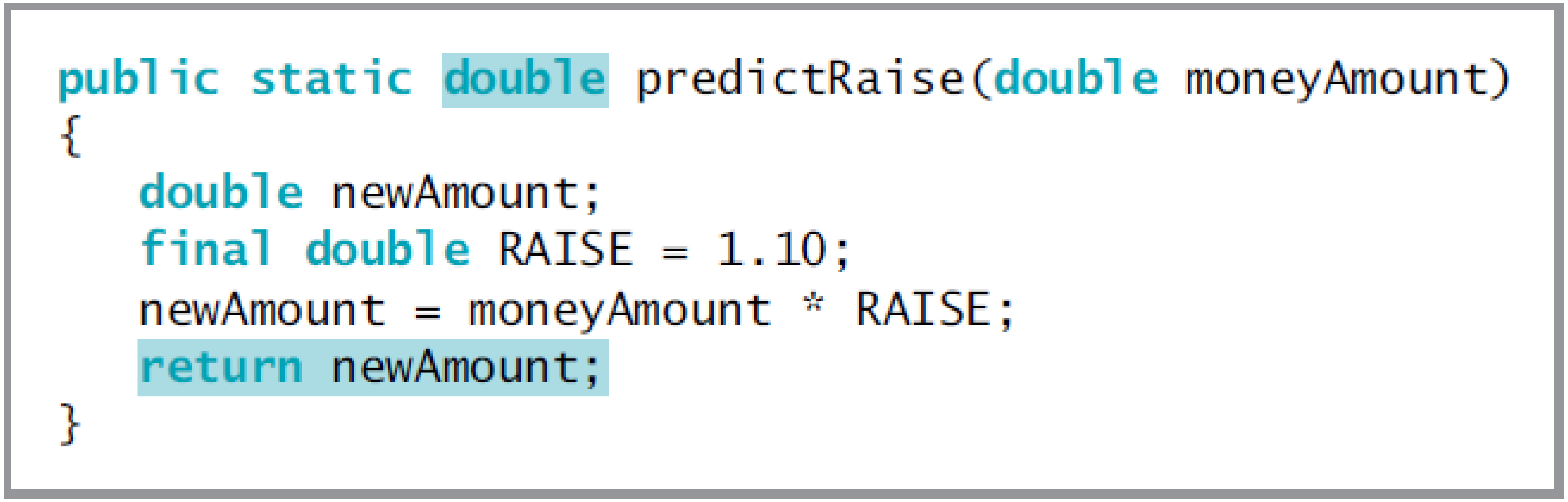
TWO TRUTHS AND A LIE

1. A class can contain any number of methods, and each method can be called any number of times.
2. Arguments are used in method calls; they are passed to parameters in method headers.
3. A method header always contains a return type, an identifier, and a parameter list within parentheses.

Creating Methods that Return Values

- **`return`** statement
  - Causes value to be sent from called method back to calling method
- Return type can be any type used in Java
  - Primitive types
  - Class types
  - **`void`**
    - Returns nothing
- Method's type
  - Method's return type

Creating Methods that Return Values



```
public static double predictRaise(double moneyAmount)
{
   double newAmount;
   final double RAISE = 1.10;
   newAmount = moneyAmount * RAISE;
   return newAmount;
}
```

Figure 3-12 The predictRaise() method returning a double

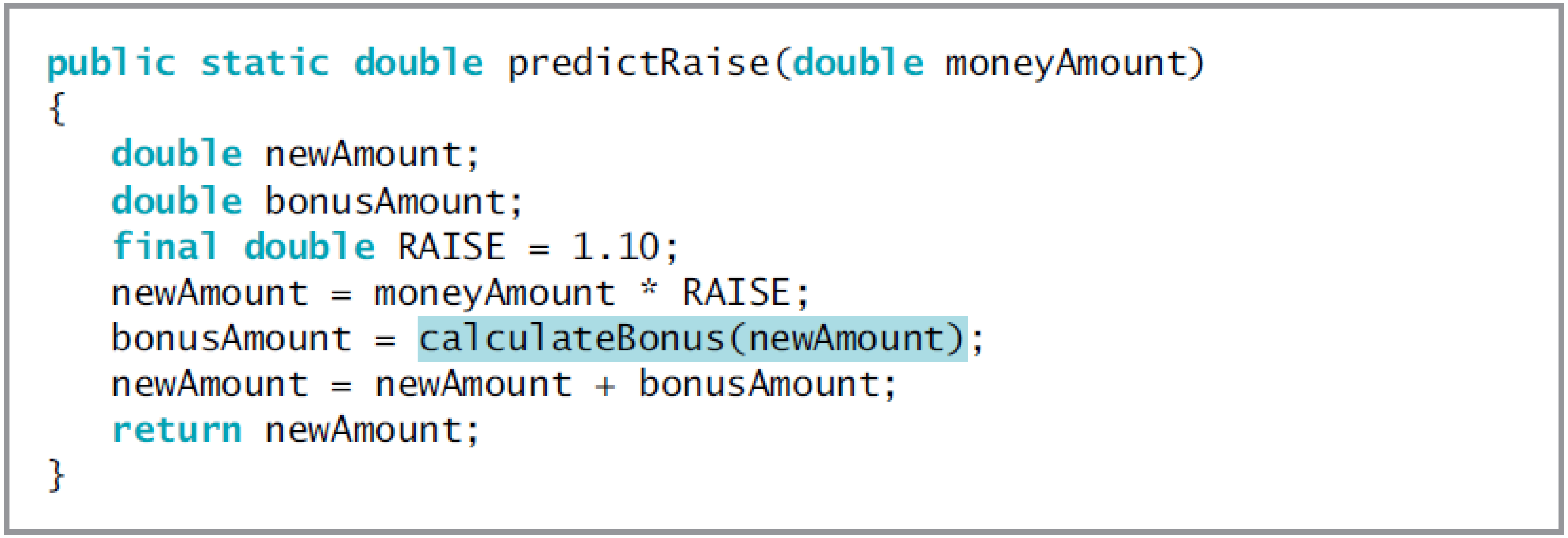
Creating Methods that Return Values

- Unreachable statements (dead code)
  - Statements after a method's return statement.
  - Logical flow leaves method at `return` statement
  - Can never execute
    - Causes compiler error

Chaining Method Calls

- Any method might call any number of other methods
- Method acts as a black box
  - Do not need to know how it works
  - Just call and use result

Chaining Method Calls



```
public static double predictRaise(double moneyAmount)
{
   double newAmount;
   double bonusAmount;
   final double RAISE = 1.10;
   newAmount = moneyAmount * RAISE;
   bonusAmount = calculateBonus(newAmount);
   newAmount = newAmount + bonusAmount;
   return newAmount;
}
```

Figure 3-13 The `predictRaise()` method calling the `calculateBonus()` method

TWO TRUTHS AND A LIE

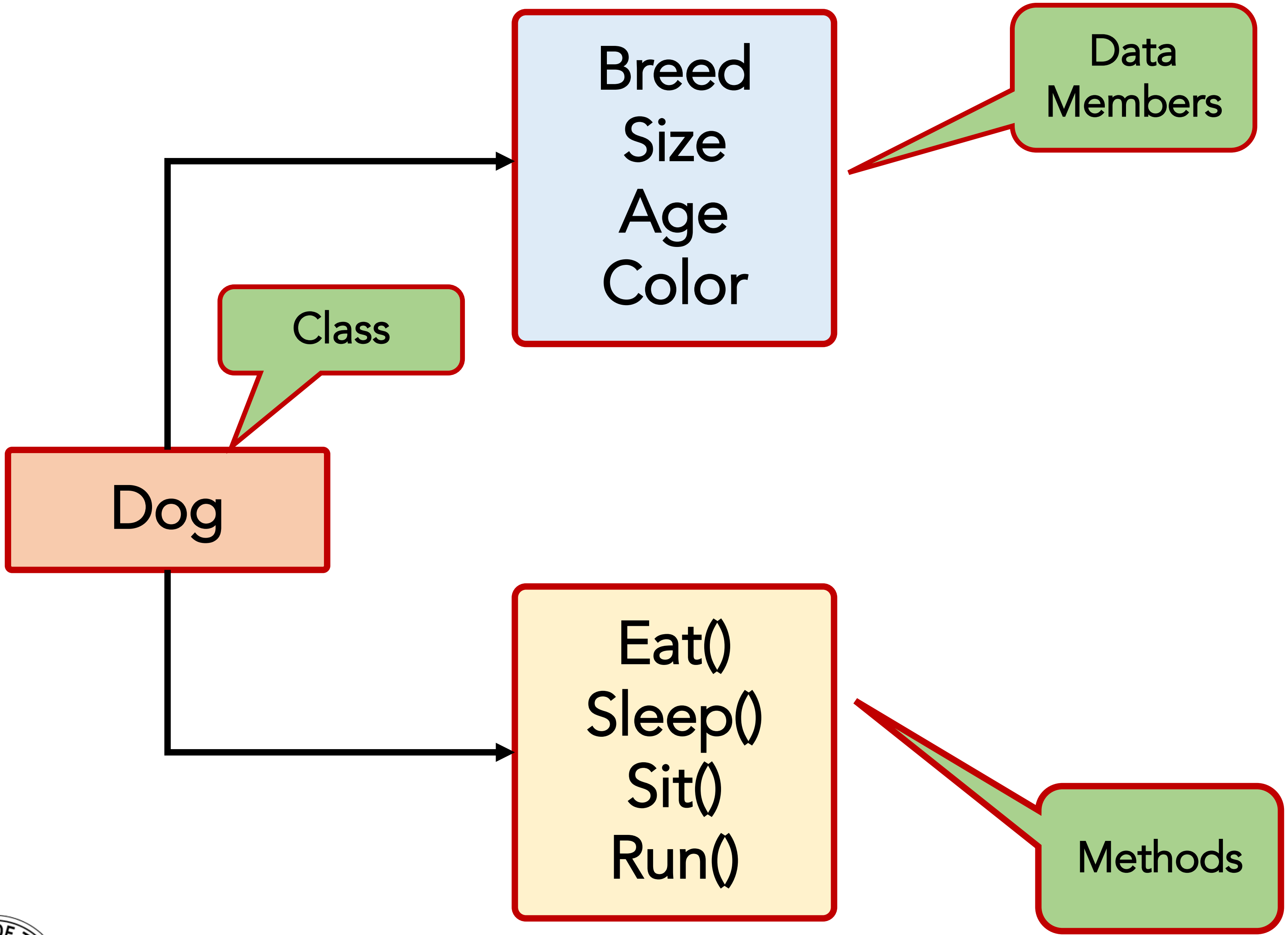
1. The return type for a method can be any type used in Java, including int, double, and void.
2. A method's declared return type must match the type of the value used in the parameter list.
3. You cannot place a method within another method, but you can call a method from within another method.

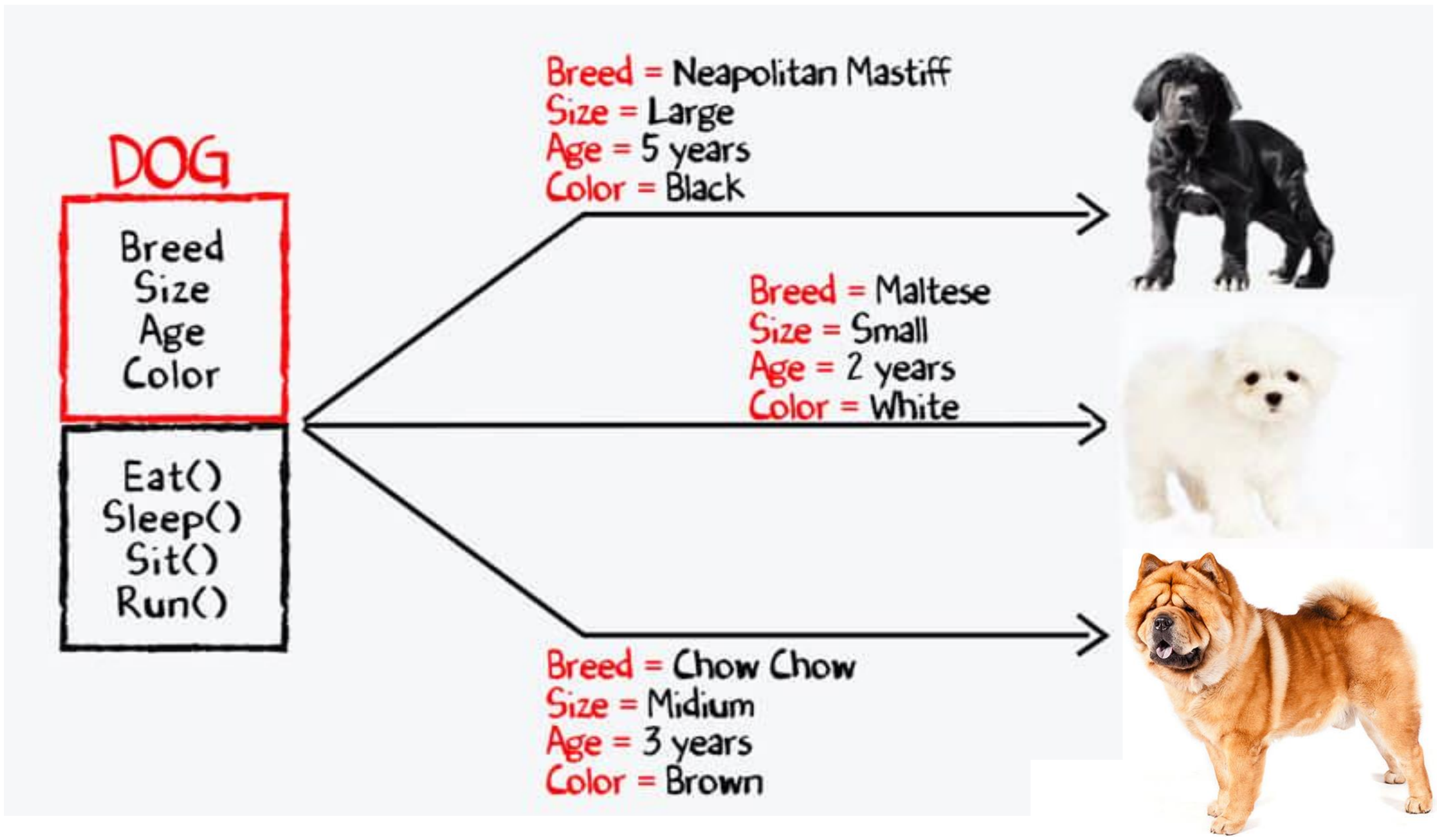
V: Learning About Class Concepts

- **Every object is a member of a class** and every object is a member of a more general class.
- **Is-a relationships**
  - Object "is a" concrete example of the class
- **Instantiation**
  - An object is an instantiation of a class, or one tangible example of a class.
- **Reusability**
  - The concept of a class is useful because of its reusability.

Learning About Class Concepts

- Methods often called upon to return piece of information to source of request
- Class client or class user
  - Application or class that instantiates objects of another prewritten class


Breed
Size
Age
Color
Data Members
Class
Dog
Eat()
Sleep()
Sit()
Run()
Methods


DOG
Breed
Size
Age
Color
Eat()
Sleep()
Sit()
Run()
Breed = Neapolitan Mastiff
Size = Large
Age = 5 years
Color = Black
Breed = Maltese
Size = Small
Age = 2 years
Color = White
Breed = Chow Chow
Size = Midium
Age = 3 years
Color = Brown

TWO TRUTHS AND A LIE

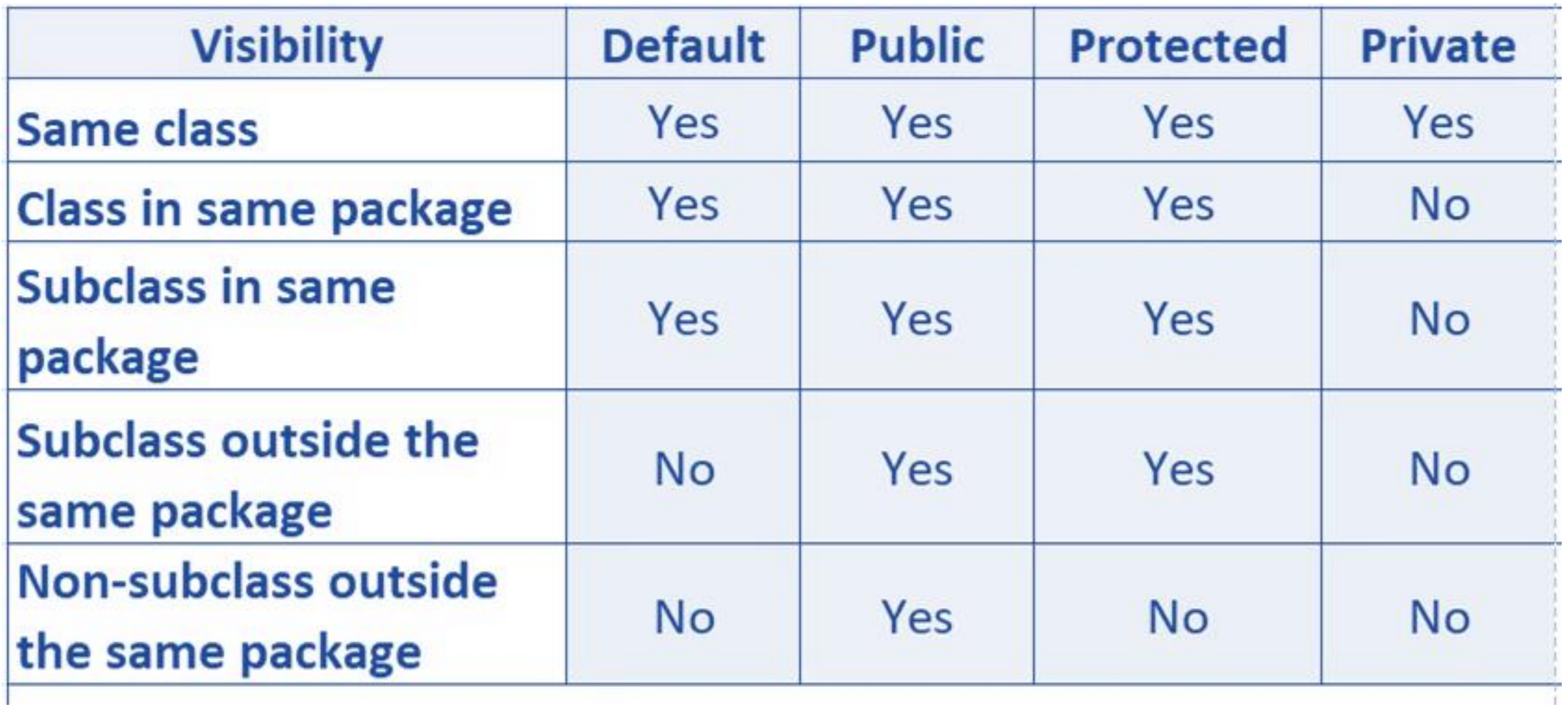
1. A class is an instantiation of many objects.
2. Objects gain their attributes and methods from their classes.
3. An application or class that instantiates objects of another prewritten class is a class client.

VI: Creating a Class

- Assign name to class
- Determine what data and methods will be part of class
- Class header
  - Optional access modifier
  - Keyword **`class`**
  - Any legal identifier for the name of class
- **`public`** class
  - Accessible by all objects

Creating a Class

- Access Modifiers

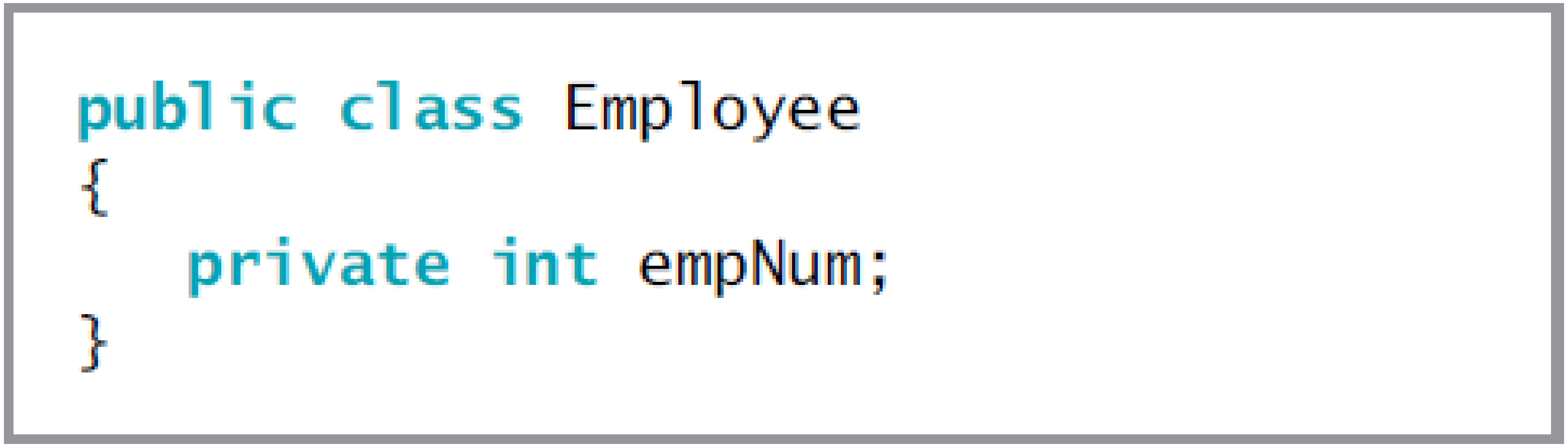


Visibility	Default	Public	Protected	Private
Same class	Yes	Yes	Yes	Yes
Class in same package	Yes	Yes	Yes	No
Subclass in same package	Yes	Yes	Yes	No
Subclass outside the same package	No	Yes	Yes	No
Non-subclass outside the same package	No	Yes	No	No

Creating a Class

See sample source files.

Creating a Class



```
public class Employee
{
   private int empNum;
}
```

Figure 3-15 The Employee class with one field

Creating a Class

- `extends` **keyword**
  - Use as a basis for any other class
  - Inheritance
  - *(This will be discussed on the next modules)*
- **Data fields**
  - Variables declared within class
    - But outside of any method

Creating a Class

- **`private`** access for fields
  - No other classes can access field's values
  - Only methods of same class allowed to use **`private`** variables
  - The principle used in creating private access is sometimes called information hiding
- Most class methods `public`

TWO TRUTHS AND A LIE

1. A class header contains an optional access specifier, the keyword class, and an identifier.
2. When you instantiate objects, each has its own copy of each static data field in the class.
3. Most fields in a class are private, and most methods are `public`.

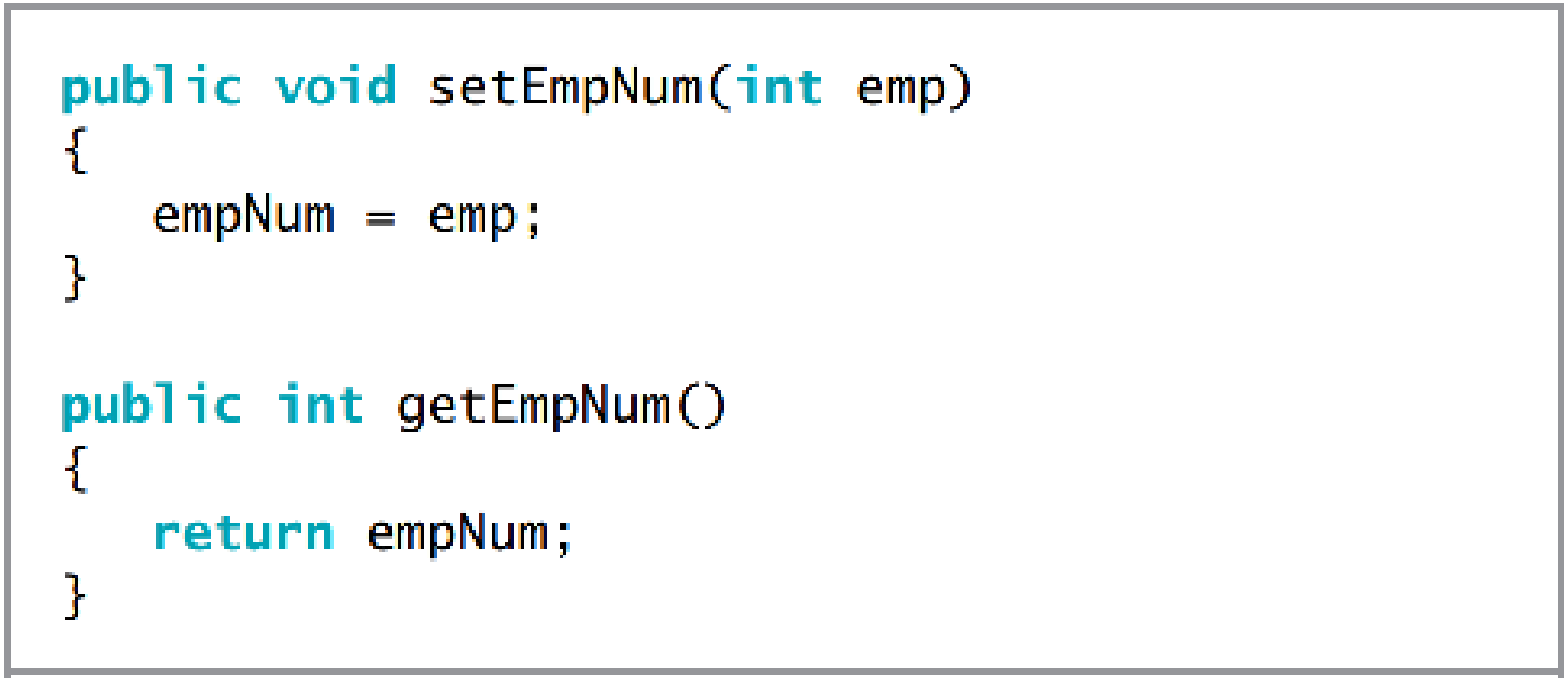
VII: Creating Instance Methods in a Class

- Classes contain methods
- Methods that set or change field values are called **mutator methods.**
  - Starts with prefix “**set**”
  - Sometimes called as **setters**
- Methods that retrieve values are called **accessor methods**.
  - Starts with prefix “**get**”
  - Sometimes called as **getters**

Understanding Data Hiding

- Data hiding using **encapsulation**
  - Data fields are usually `private`
  - Client application accesses them only through `public` interfaces
- **Set method**
  - Controls data values used to set variable
- **Get method**
  - Controls how value is retrieved

Creating Instance Methods in a Class



```
public void setEmpNum(int emp)
{
    empNum = emp;
}

public int getEmpNum()
{
    return empNum;
}
```

Figure 3-24 The `setEmpNum()` and `getEmpNum()` methods

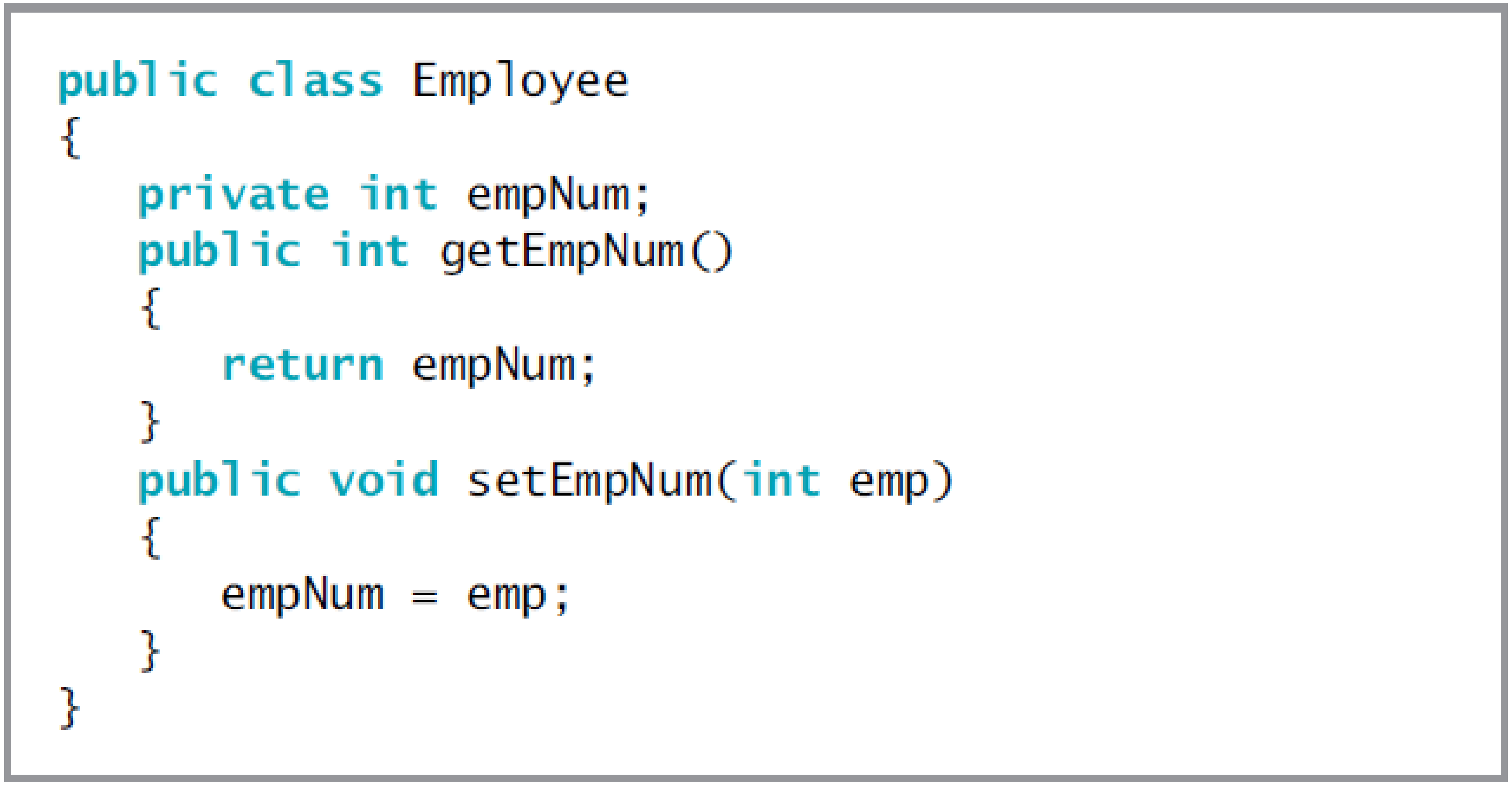
Creating Instance Methods in a Class

- **Non-static methods**
  - Instance methods
  - "Belong" to objects
- Typically declare non-static data fields
- **`static`** class variables
- If it occurs **once per class**, it is **`static`**, but if it occurs **once per object**, it is not **`static`**.

Static	Nonstatic
In Java, `static` is a keyword. It also can be used as an adjective.	There is no keyword for nonstatic items. When you do not explicitly declare a field or method to be static, then it is nonstatic by default.
Static fields in a class are called class fields.	Nonstatic fields in a class are called instance variables.
Static methods in a class are called class methods.	Nonstatic methods in a class are called instance methods.
When you use a static field or method, you do not use an object; for example: `JOptionPane.showDialog();`	When you use a nonstatic field or method, you must use an object; for example: `System.out.println();`
When you create a class with a static field and instantiate 100 objects, only one copy of that field exists in memory.	When you create a class with a nonstatic field and instantiate 100 objects, then 100 copies of that field exist in memory.
When you create a static method in a class and instantiate 100 objects, only one copy of the method exists in memory and the method does not receive a `this` reference.	When you create a nonstatic method in a class and instantiate 100 objects, only one copy of the method exists in memory, but the method receives a `this` reference that contains the address of the object currently using it.
Static class variables are not instance variables. The system allocates memory to hold class variables once per class, no matter how many instances of the class you instantiate. The system allocates memory for class variables the first time it encounters a class, and every instance of a class shares the same copy of any static class variables.	Instance fields and methods are nonstatic. The system allocates a separate memory location for each nonstatic field in each instance.

Table 3-1 Comparison of static and nonstatic

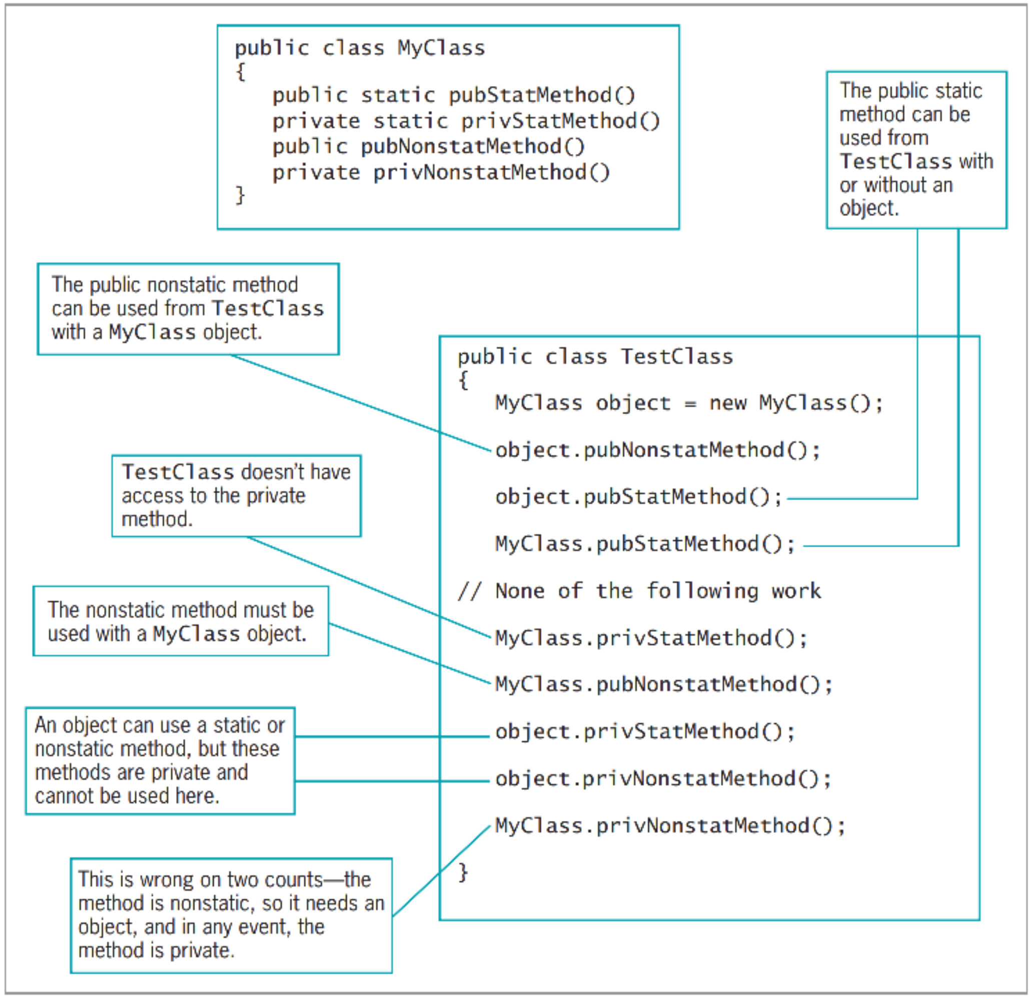
Creating Instance Methods in a Class



```java
public class Employee
{
   private int empNum;
   public int getEmpNum()
   {
      return empNum;
   }
   public void setEmpNum(int emp)
   {
      empNum = emp;
   }
}
```

Figure 3-18 The Employee class with one field and two methods

Creating Instance Methods in a Class



```
public class MyClass
{
   public static pubStatMethod()
   private static privStatMethod()
   public pubNonstatMethod()
   private privNonstatMethod()
}
```

```
public class TestClass
{
   MyClass object = new MyClass();
   object.pubNonstatMethod();
   object.pubStatMethod();
   MyClass.pubStatMethod();
// None of the following work
   MyClass.privStatMethod();
   MyClass.pubNonstatMethod();
   object.privStatMethod();
   object.privNonstatMethod();
   MyClass.privNonstatMethod();
}
```

The public static method can be used from TestClass with or without an object.

The public nonstatic method can be used from TestClass with a MyClass object.

TestClass doesn't have access to the private method.

The nonstatic method must be used with a MyClass object.

An object can use a static or nonstatic method, but these methods are private and cannot be used here.

This is wrong on two counts—the method is nonstatic, so it needs an object, and in any event, the method is private.

Figure 3-25 Summary of legal and illegal method calls based on combinations of method modifiers

Organizing Classes

- Place data fields in logical order
  - At beginning of class
  - Fields listed vertically
- May place data fields and methods in any order within a class
  - Common to list all data fields first
  - Can see names and data types before reading methods that use data fields

Organizing Classes

```java
// Employee.java holds employee data
// Programmer: Lynn Greenbrier
// Date: September 24, 2011
public class Employee
{
   // private data fields:
   private int empNum;
   private String empLastName;
   private String empFirstName;
   private double empSalary;

   // public get and set methods:
   public int getEmpNum()
   {
      return empNum;
   }
   public void setEmpNum(int emp)
   {
      empNum = emp;
   }
   // ...and so on
```

Figure 3-23 Start of Employee class with data fields, methods, and comments

TWO TRUTHS AND A LIE

1. The keyword static is used with class-wide methods, but not for methods that “belong” to objects.
2. When you create a class from which objects will be instantiated, most methods are non-static because they are associated with individual objects.
3. Static methods are instance methods.

VIII: Declaring Objects and Using Their Methods

- Declaring class **does not** create any actual objects
- Create instance of class
  - Supply type and identifier
  - Allocate computer memory for object
  - Use **`new`** operator

```
Employee someEmployee;
someEmployee = new Employee();
```

  - or in a shorthand notation like:

```
Employee someEmployee = new Employee();
```

Declaring Objects and Using Their Methods

- Reference to the object
  - Name for a memory address where the object is held
- **Constructor method**
  - Method that creates and initializes class objects
  - Can write own constructor methods
  - Java creates a constructor
    - Name of constructor method always same as name of class

Declaring Objects and Using Their Methods

- After object instantiated
  - Methods accessed using:
    - Object’s identifier
    - Dot
    - Method call

Declaring Objects and Using Their Methods

See sample source files:
OOP Module 3

- *MyClass.java*
- *OtherClass.java*

Declaring Objects and Using Their Methods

```
public class DeclareTwoEmployees
{
   public static void main(String[] args)
   {
      Employee clerk = new Employee();
      Employee driver = new Employee();
      clerk.setEmpNum(345);
      driver.setEmpNum(567);
      System.out.println("The clerk's number is " +
        clerk.getEmpNum() + " and the driver's number is " +
        driver.getEmpNum());
   }
}
```

Figure 3-19 The DeclareTwoEmployees class

Declaring Objects and Using Their Methods

See sample source files:
OOP Module 3

- *Employee.java*
- *DeclaringEmployee.java*

TWO TRUTHS AND A LIE

1. When you declare an object, you give it a name and set aside enough memory for the object to be stored.
2. An object name is a reference; it holds a memory address.
3. When you don’t write a constructor for a class, Java creates one for you; the name of the constructor is always the same as the name of its class.

IX: An Introduction to Using Constructors

- A constructor establishes an object; a default constructor is one that **requires no arguments**.
- A default constructor is **created automatically by the Java compiler** for any class you create
- How is it written?
  - Must have same name as class it constructs
  - Cannot have return type
  - `public` access modifier

An Introduction to Using Constructors

- Default constructor provides specific initial values to object's data fields
  - Numeric fields
    - Set to 0 (zero)
  - Character fields
    - Set to Unicode '\u0000'
  - Boolean fields
    - Set to `false`
  - Nonprimitive object fields
    - Set to `null`

An Introduction to Using Constructors

```
Employee chauffeur = new Employee();
```

- Actually calling method named `Employee()`

- Default constructor
  - Requires no arguments
  - Created automatically by Java compiler
    - For any class
    - Whenever you do not write constructor

An Introduction to Using Constructors

```
public Employee()
{
    empSalary = 300.00;
}
```

Figure 3-24 The Employee class constructor

TWO TRUTHS AND A LIE

1. In Java, you cannot write a default constructor; it must be supplied for you automatically.
2. The automatically supplied default constructor sets all numeric fields to 0, character fields to Unicode '\u0000', Boolean fields to false, and fields that are object references to null.
3. When you write a constructor, it must have the same name as the class it constructs, and it cannot have a return type.

X: Understanding that Classes are Data Types

- Classes you create become data types
- **Abstract Data Type (ADT)**
  - A data type whose implementation is hidden and accessed through its public methods.
- A class that you create can also be called a **programmer-defined data type**; in other words, it is a type that is not built into the language.

You Do It

- Creating a **`static`** method that requires no arguments and returns no values
- Calling a **`static`** method from another class
- Creating a **`static`** method that accepts arguments and returns values
- Creating a class that contains instance fields and methods

You Do It

- Creating a class that instantiates objects of another class
- Adding a constructor to a class
- Creating a more complete class

Don’t Do It

- Don’t place semicolon at end of method header
- Don’t think “default constructor” means only automatically supplied constructor
- Don’t think that class’s methods must:
  - Accept its own fields’ values as parameters
  - Return values to its own fields
- Don’t create class method that has parameter with same identifier as class field

Summary

- Method
  - Series of statements that carry out a task
  - Declaration includes parameter type and local name for parameter
  - Can pass multiple arguments to methods
  - Has return type
- Class objects
  - Have attributes and methods associated with them
- Instantiate objects that are members of class

Summary

- Constructor
  - Method establishes object and provides specific initial values for object's data fields
- Everything is an object
  - Every object is a member of a more general class
- Implementation hiding, or encapsulation
  - `private` data fields
  - `public` access methods

End of Module.

REFERENCE:

Farrell, J. (2016). *Java Programming*. 8th Edition. Course Technology, Cengage Learning.