Objects and Classes

Part 3 – User-Defined Classes

Chapter 4, Core Java, Volume I

Contents

Defining Your Own Classes

- Role of Classes
  - Code template to create objects (instances)
  - A type for objects - user-defined types (cf. structures in C language)

```
String s = new String("Hello");  // String is the type of s
```

- Kinds of Classes
  - User-defined Classes
  - Library Classes (e.g. String, Math, Scanner, etc)

  - Workhorse classes without main methods (Card, Player, Course, etc.)
  - A special (application) class with main method (CardGame, CourseEnrollment)

Defining Your Own Classes

- General structure

```
class ClassName
{
    instance_variable-1 declaration    // also called instance fields  or just fields
    instance_variable-1 declaration
    …
    constructor-1 definition
    constructor-2 definition
    …
    method-1 definition
    method-2 definition
    …
 }
```

Example : Employee Test

```
// EmployeeTest.java
import java.time.*;
public class EmployeeTest
{
   public static void main(String[] args)
   {
      // fill the staff array with three Employee objects
      Employee[] staff = new Employee[3];

      staff[0] =
             new Employee("Carl Cracker", 75000, 1987, 12, 15);
      staff[1] =
             new Employee("Harry Hacker", 50000, 1989, 10, 1);
      staff[2] =
             new Employee("Tony Tester", 40000, 1990, 3, 15);

      // raise everyone's salary by 5%
      for (Employee e : staff)
         e.raiseSalary(5);

      // print out information about all Employee objects
      for (Employee e : staff)
         System.out.println("name=" + e.getName() + ",salary="
            + e.getSalary() + ",hireDay=“ + e.getHireDay());
   } // end of main
} // end of EmployeeTest
```

Example : an Employee Class

```
class Employee
{
   // Instance Fields
   private String name;
   private double salary;
   private LocalDate hireDay;
   // Constructors
   public Employee(String n, double s, int year, int month,
      int day)
   {
      name = n;
      salary = s;
      hireDay = LocalDate.of(year, month, day);
   }
  // Methods
   public String getName()
   {
       return name;
   }
   public double getSalary()
   {
      return salary;
   }
   public void setSalary(double s)
   {
      salary = s;
   }
   public LocalDate getHireDay()
   {
      return hireDay;
   }
   public void raiseSalary(double byPercent)
   {
      double raise = salary * byPercent / 100;
      salary += raise;
   }
} // end of Employee
```

Java Program Structure using Multiple Files

```
// EmployeeTest.java
import java.time.*;
public class EmployeeTest
{
   public static void main(String[] args)
   {
      // fill the staff array with three Employee objects
      Employee[] staff = new Employee[3];

      staff[0] =
             new Employee("Carl Cracker", 75000, 1987, 12, 15);
      staff[1] =
             new Employee("Harry Hacker", 50000, 1989, 10, 1);
      staff[2] =
             new Employee("Tony Tester", 40000, 1990, 3, 15);
        …
    }
}  // end of EmployeeTest
```

```
// Empolyee.java
pubic class Employee
{
   private String name;
   private double salary;
   private LocalDate hireDay;

   public Employee(String n, double s, int year, int month,
      int day)
   {
      name = n;
      salary = s;
      hireDay = LocalDate.of(year, month, day);
   }
   public String getName()
   {
      return name;
   }
   …
} // end of Employee
```

Instance Variables

- Instance variables (or instance fields)
  - Are variables which are declared in a class but outside the methods
  - Are shared by the methods defined in the class
  - All instances of the class have their own copies of instance variables

- Initializing instance variables
  - Can be initialized in the declaration, within the constructor methods, etc.
    - private int salary = 5000.0; // in Employee class
    - private int top = -1; // in Stack class
  - Without initializing, they have default initial values
    - `Reference type : null`
    - `Numerica types : 0 or 0.0`
    - `Boolean : false`
- Scope: throughout the class except in static methods
- Lifetime: is until the object stays in memory

Constructors and Creating Objects

- A constructor method initializes instance variables:

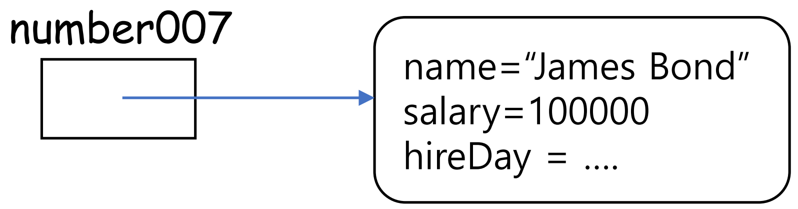
```
public Employee(String n, double s, int year, int month, int day)
{
   name = n;
   salary = s;
   hireDay = LocalDate.of(year, month, day);
}
```

- Creating objects

```
Employee number007 = new Employee("James Bond", 100000, 1950, 1, 1);
```

sets the fields as follows:

```
name = "James Bond";
salary = 100000;
hireDay = LocalDate.of(1950, 1, 1);
```



Constructors and Creating Objects

- The name is the same as the class name.
- Constructor only works with `new` operator.
- Constructors should be public (with some exceptions)
- A class can have more than one constructor (overloading).
- A constructor can take zero, one, or more parameters.
- A constructor has no return value.

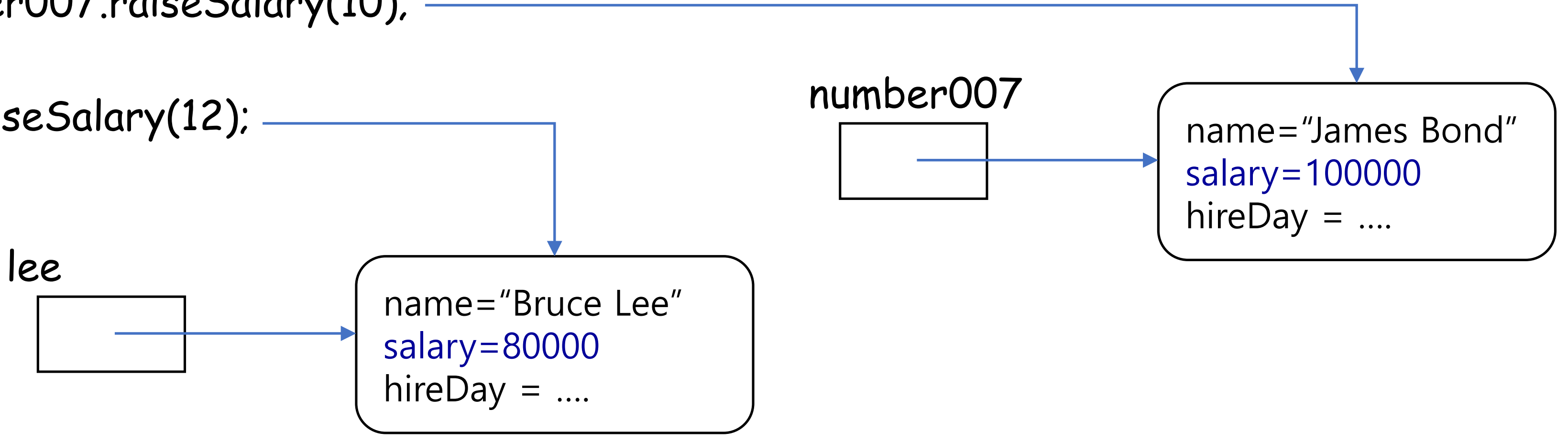
Methods and Method Calls

- Methods access and modify instance fields of the object to which the method applies

```
public void raiseSalary(double byPercent)
{
   double raise = salary * byPercent / 100;
   salary += raise;
}
```

- Method calls are executed on the context of the object to which the method applies.

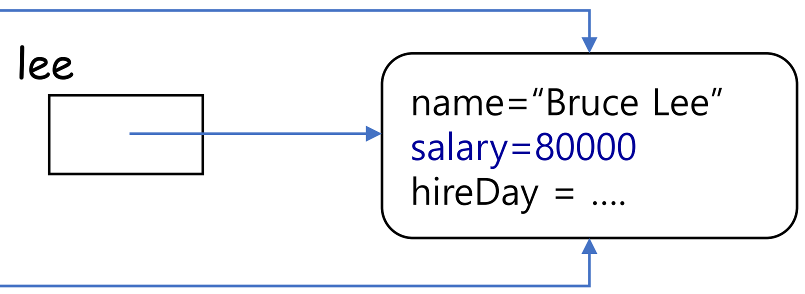
```
…
number007.raiseSalary(10);
…
lee.raiseSalary(12);
```



Methods and Method Calls

- A method can call another method within the same class
  - The called methed is also executed on the same context as the calling method.

```
public void raiseSalary(double byPercent)  // modified version using a method call
{
   double raise = salary * byPercent / 100;
   setSalary(salary + raise);
}
…
lee.raiseSalary(12);
…
```



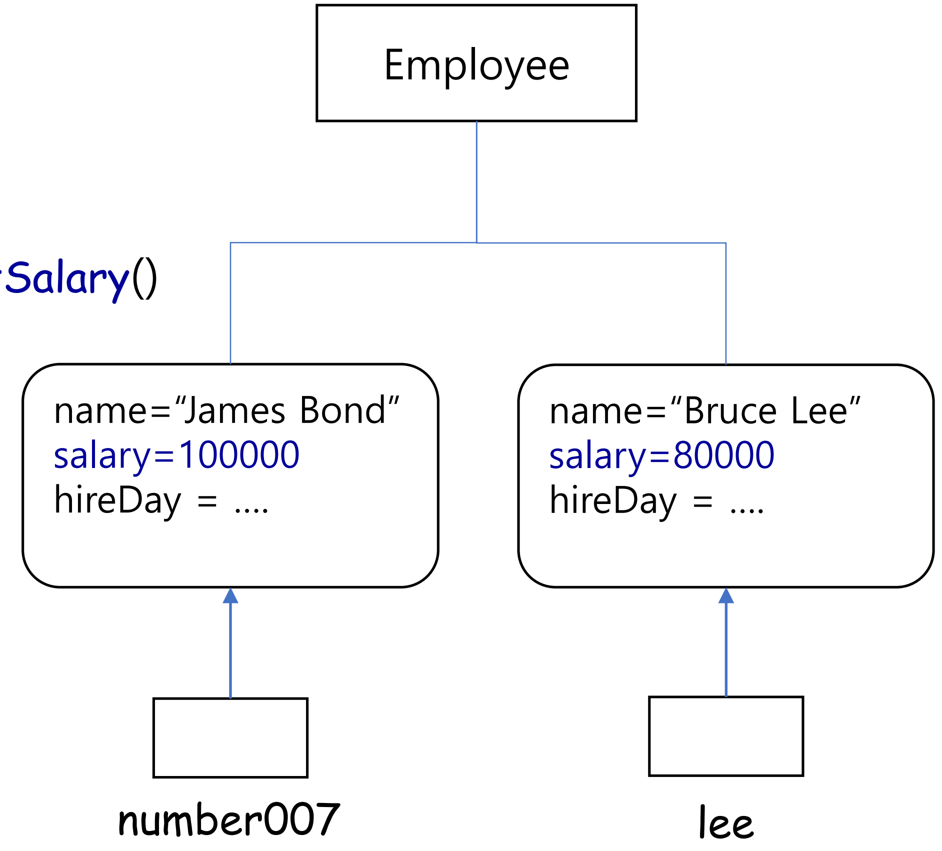
Implicit Parameters : this

- Method code in a class is shared by serveral objects of the class
- How does the method know the appropriate object?

```
number007.raiseSalary(10);  // number007.salary must be accessed
lee.raiseSalary(12);        // lee.salary must be accessed
```

- The *implicit* parameter this (pseudo variable)
  - this refers to the object on which the method is invoked
  - salary in raiseSalary implicitly denotes this.salary
  - call to setSalary() in raiseSalary implicitly denotes this.setSalary()
- Can optionally use this to refer instance variables or to invoke methods:

```
public void raiseSalary(double byPercent)
{
   double raise = this.salary * byPercent / 100;
   this.setSalary(this.salary + raise);
}
```


Employee
name="James Bond"
salary=100000
hireDay = ...
name="Bruce Lee"
salary=80000
hireDay = ...
number007
lee