CSE3040 Java Language

Lecture 07: Object-Oriented Programming (1)

Dept. of Computer Engineering,

Sogang University

This material is based on lecture notes by Prof. Juho Kim. Do not post it on the Internet.

Java: An Object-Oriented Language

- In Java, most variables and literals are objects.
- An object is an instance of a class.
  - A class can be viewed as a **type** of an object.

```
// Lecture.java

class Employee {
    String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}

public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           m.setName("Peter");
           System.out.println(m.getName());
     }
}
```

Class Definition

- In Lecture.java, two classes are defined.
  - class Employee
  - public class Lecture

- A single .java file may contain multiple classes, but only one of the classes can be a public class.
  - The name of the public class should match the file name.
    - public class Lecture → Lecture.java

- When Lecture.java is compiled using **javac**, a .class file is created for each defined class.

- The public class contains the main method, which is the starting point of the program.

  $ javac Lecture.java

  - Lecture.class and Employee.class are created.

  $ java Lecture

  - The main method of class Lecture is executed.

Class Definition

- A class contains variables and methods.
  - String name; → a **variable** of class Employee
  - public void SetName(…) { … } → a **method** of class Employee

```
// Lecture.java

class Employee {
    String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}

public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           m.setName("Peter");
           System.out.println(m.getName());
     }
}
```

Instance Variable

- An instance variable is a variable defined in a class without the keyword static.
  - String name; → an instance variable
  - An instance variable belongs to an object instance.
    - If there are two instances, their instance variables are independent of each other.

- An access modifier defines who can access the variable.
  - String name: visible to the classes in the same package
  - public String name: visible to any class
  - protected String name: visible to subclasses and classes in the same package
  - private String name: visible to the class only

  - Access modifiers are also applied to **methods**.

Instance Variable

- Instance variables of two different objects are independent.

```
class Employee {
    String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}

public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           m.setName("Peter");
           System.out.println(m.getName());
           Employee n = new Employee();
           n.setName("John");
           System.out.println(n.getName());
     }
}
```

Access Modifiers

- In the example below, class Lecture can directly access variable name because Lecture and Employee are in the same package.
  - m.name = "Peter";
- If the variable name is defined as private, the code will cause compile error.
  - private String name;

```
// Lecture.java
package sogangcse;

class Employee {
     String name;
}

public class Lecture {
     public static void main(String[] args) {
            Employee m = new Employee();
            m.name = "Peter";
            System.out.println(m.name);
     }
}
```

Methods

- Method Header
  - access modifier: public, protected, private, (no keyword)
  - return type
    - If the method has no return value, then return type is void.
  - method name
    - naming convention: use uppercase letters for the first letter in a word.
  - arguments
    - *argument_class argument_name*
    - multiple arguments are separated by comma.

```
class Employee {
    private String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
```

Methods

- Method Body
  - If the method has a return value, use keyword return to end the method and return the value.
- this
  - When a method of an object is called, the object itself is referred using this.
    - this could be omitted if there is no ambiguity, but its use is suggested.

```
class Employee {
    private String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           m.setName("Peter");
           System.out.println(m.getName());
     }
}
```

Methods

- Calling a method
  - format: *object.method(args)*

```
class Employee {
    private String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           m.setName("Peter");
           System.out.println(m.getName());
     }
}
```

Encapsulation

- Encapsulation is one of the major principles of object-oriented programming.
  - Make visible only what is necessary, and hide everything else.
  - Only the methods that must be called from outside the class are defined public.
  - Other methods and variables are defined as private.

```
class Employee {
    private String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           m.setName("Peter");
           System.out.println(m.getName());
     }
}
```

Creating an Object

- An object can be created using the keyword new.
- When an object is created, the constructor of the class is called.
- If no constructor is defined, a default constructor is called.
  - No arguments, does nothing.
  - Even if you don’t define a constructor, every class has a constructor.

```
class Employee {
    private String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           m.setName("Peter");
           System.out.println(m.getName());
     }
}
```

Deleting an Object?

- It is not necessary to delete objects in Java, because Java provides automatic garbage collection.

- Automatic garbage collection
  - When an object is created, memory is allocated for that object.
  - While the program is running, it is possible that a certain object is not referenced any more.
  - In that case, the garbage collector will automatically deallocate memory so that it could be used for other objects.

  - Without automatic garbage collection, the programmer needs to explicitly delete objects. Otherwise, it will lead to memory leak.

Constructor

- When defining a class, a constructor can be defined.
  - Typically a constructor initializes instance variables.
- A constructor has the same name as the class, and does not have a return type.

```
class Employee {
    private String name;
    public Employee() {
        this.name = "NoName";
    }
    public String getName() {
        return this.name;
    }
}
public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           System.out.println(m.getName());
     }
}
```

Constructor

- You can define constructors with arguments.
  - The constructor should be called with matching number of arguments.

```
class Employee {
    private String name;
    public Employee(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee("Harry Potter");
           System.out.println(m.getName());
     }
}
```

Constructor

- You can define multiple constructors with different arguments.
  - Which constructor is called depends on the arguments given by the caller.
- This is called method overloading, and is applicable to any methods.
  - Multiple methods of the same name with different arguments.

```
class Employee {
    private String name;
    public Employee() {
        this.name = "NoName";
    }
    public Employee(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee("Harry Potter");
           System.out.println(m.getName());
     }
}
```

Constructor

- You can call an overloaded constructor inside a constructor.
- In that case, you use this instead of the class name.

```
class Employee {
    private String name;
    public Employee() {
        this("NoName");
    }
    public Employee(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}
public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           System.out.println(m.getName());
     }
}
```

Programming Lab #07

07-01. Defining and Using class Employee

- Write a Java program that satisfies the following requirements.
  - Define class Employee.
  - The class should have one String type instance variable named name.
  - The class should have two overloading constructors:
    - A constructor with no argument sets variable name to be "anonymous".
    - A constructor with a single argument sets variable name with the argument.
  - The class should have two instance methods, setName and getName.
    - The setName method takes one argument and sets name with the argument.
    - The getName method returns the name.
  - All instance variables must be defined as private variables.
  - The following code should run without error. (You must not modify the following code.)

```java
public class Ex07_01 {
  public static void main(String[] args) {
    Employee e1 = new Employee("Harry");
    Employee e2 = new Employee();
    System.out.println(e1.getName());
    System.out.println(e2.getName());
    e2.setName(e1.getName() + " Potter");
    System.out.println(e2.getName());
  }
}
```

07-02. Defining and Using class Employee

- Write a Java program that satisfies the following requirements.
  - Define and implement class Employee so that the following code should run and produce the correct result.
  - All instance variables must be declared as private variables.

```
public class Ex07_02 {
  public static void main(String[] args) {
    Employee e1 = new Employee("John", 100000);
    Employee e2 = new Employee("Peter");
    Employee e3 = new Employee();
    System.out.println("Salary of " + e1.getName() + " is " + e1.getSalary());
    System.out.println("Salary of " + e2.getName() + " is " + e2.getSalary());
    System.out.println("Salary of " + e3.getName() + " is " + e3.getSalary());
    e1.setSalary(150000);
    System.out.println("Salary of " + e1.getName() + " is " + e1.getSalary());
  }
}
```

```
Salary of John is 100000
Salary of Peter is 50000
Salary of Employee is 50000
Salary of John is 150000
```

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr