CSE3040 Java Language

Lecture 08: Object-Oriented Programming (2)

Dept. of Computer Engineering,
Sogang University

Creating an Array of Objects

- It is possible to define an array of objects.
- In that case, each element of the array must be separately created.
  - new Employee[3] only creates an array space for three objects.

```
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[3];
           m[0] = new Employee("Mario");
           m[1] = new Employee("Luigi");
           m[2] = new Employee("Toad");
           System.out.println(m[0].getName());
     }
}
```

Instance Variable Instantiation

- You can assign initial value to an instance variable.
  - String name = "Joe";
  - If a variable is not initialized, they are assigned default values.
    - For example, a default initial value for a String object is null.

```
// Lecture.java

class Employee {
    private String name = "Joe";
    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();
           System.out.println(m.getName());
     }
}
```

Default Initialization

- If an instance variable is not initialized, it is assigned a default value.
  - numbers: 0, boolean: false, objects: null
- Local variables must be initialized.
  - Otherwise, an error occurs.

```
class Employee {
    private int salary;
    public int getSalary() {
        return this.salary;
    }
}
public class Lecture {
     public static void main(String[] args) {
           Employee m = new Employee();
           System.out.println(m.getSalary());
           int local_var;
           System.out.println(local_var);    // Error: local variable not initialized!
     }
}
```

Final Instance Variable

- If an instance variable is declared using keyword final, the value of the variable can only be set in a constructor.

```
class Employee {
      private final String name;
      public Employee() {
            this.name = "Donald";
      }
      public void setName(String name) {
            this.name = name;       // error: cannot assign a value to a final instance variable.
      }
}
```

Static Variable

- When defining a class, a variable can be defined as a static variable.
  - A static variable belongs to the class, and not the instances of the class.
  - In contrast, an instance variable is independent for each instance.
  - A static variable is "shared" among all instances.

```
class Employee {
    private static int lastId = 0;
    private int id;
    public Employee() { id = ++lastId; }
    public int getId() { return this.id; }
    public int getLastId() { return this.lastId; }
}
public class Lecture {
    public static void main(String[] args) {
        Employee m = new Employee();
        Employee n = new Employee();
        System.out.println(m.getId());
        System.out.println(n.getId());
        System.out.println(m.getLastId());
        System.out.println(n.getLastId());
    }
}
```

Static Variable

- A static variable is typically initialized with variable declaration.
- Also, a static initialization block can be used to initialize static variables.

```
class Employee {
    private static int lastId;
    static {
        lastId = 0;
    }
    private int id;
    public Employee() { id = ++lastId; }
    public int getId() { return this.id; }
    public int getLastId() { return this.lastId; }
}
public class Lecture {
    public static void main(String[] args) {
        Employee m = new Employee();
        Employee n = new Employee();
        System.out.println(m.getId());
        System.out.println(n.getId());
        System.out.println(m.getLastId());
        System.out.println(n.getLastId());
    }
}
```

Static final Variable

- Typically, constants that are related to a class is defined as static variables.
- The library class Math has a static variable called PI.
  - If PI is not a static variable, an object of class Math must be created in order to use the constant.

```
// defined in library
public class Math {
    public static final double PI = 3.14159265358979323846;
}
public class Lecture {
    public static void main(String[] args) {
        System.out.println(Math.PI);
    }
}
```

- System.out is a static final variable
  - Defined in the definition of class System.

```
// defined in library
public class System {
    public static final PrintStream out;
    …
}
```

Static Method

- A static method is a method that does not belong to specific instances but belongs to the class itself.
  - You do not need to create an instance to call a static method.

- The method pow of class Math is a static method.
  - When calling a static method, class name is used instead of object name
    - Math.pow(2, 5);

- The main method is also defined as a static method.

```
public class Lecture {
	public static void main(String[] args) {
		System.out.println("Hello World!");
	}
}
```

Static Variable and Static Method

- It is possible that you can access a static variable from an instance method.
- However, in principle you should access static variables using static methods.
- You cannot use this in a static method.
  - this indicates the object, but static methods are not related to objects.
- A static method cannot access an instance variable.

```
class Employee {
    private static int lastId = 0;
    private int id;
    public Employee() { id = ++lastId; }
    public int getId() { return this.id; }              // getId cannot be a static method
    public static int getLastId() { return lastId; }    // cannot use this.lastId
}
public class Lecture {
    public static void main(String[] args) {
        System.out.println(Employee.getLastId());
        Employee m = new Employee();
        System.out.println(Employee.getLastId());
    }
}
```

Java: Call-by-Value

- Java uses Call-by-Value principle when calling methods.

- Call-by-Value
  - When calling a method, the values of arguments are copied to the parameters in the method.
  - In the example below, changing values of a and b does not affect values of a1 and a2.

```
public class Lecture {

    public static void swap(int a, int b) {
        int temp = a;
        a = b;
        b = temp;
    }
    public static void main(String[] args) {
        int a1 = 10;
        int a2 = 20;
        swap(a1, a2);
        System.out.println(a1 + " " + a2);
    }
}
```

Java: Call-by-Value

- You must be careful about the results when arrays and objects are passed to methods.
  - When an array is passed as an argument, the reference to the array (memory address of the array) is passed using call-by-value.

```
public class Lecture {
    public static void swap(int [] x) {
        int temp = x[0];
        x[0] = x[1];
        x[1] = temp;
    }
    public static void main(String[] args) {
        int [] a = new int[2];
        a[0] = 10;
        a[1] = 20;
        swap(a);
        System.out.println(a[0] + " " + a[1]);
    }
}
```

Java: Call-by-Value

- Passing an object as an argument
  - A reference to the object is passed using call-by-value.

```
class Employee {
    String name;
    public Employee(String name) {
        this.name = name;
    }
}

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

Programming Lab #08

08-01. Arrays of Objects

- Write a Java program that satisfies the following requirements.
  - Use the following class Employee.
  - Create an array of Employees that has five elements.
  - Name of the five employees is: Kim, Lee, Park, Choi, Chung
  - Print the names of five employees on the screen.

```
class Employee {
  private String name;

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

08-02. Static Variables and Static Methods

- What will be printed on the display when you execute this program?
- What is a more proper way to use static variables?

```
class Employee {
  private static int lastId = 0;
  private int id;
  public Employee() { id = ++lastId; }
  public int getId() { return this.id; }
  public int getLastId() { return this.lastId; }
}

public class Ex08_02 {
  public static void main(String[] args) {
    Employee m = new Employee();
    Employee n = new Employee();
    System.out.println(m.getId());
    System.out.println(n.getId());
    System.out.println(m.getLastId());
    System.out.println(n.getLastId());
  }
}
```

08-03. Call-by-Value: Primitive Types and Arrays

- What will be printed on the display when you execute this program?

```
public class Ex08_03 {
  public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
  }
  public static void swap(int[] x) {
    int temp = x[0];
    x[0] = x[1];
    x[1] = temp;
  }
  public static void main(String[] args) {
    int x = 10;
    int y = 20;
    swap(x, y);
    System.out.println(x + " " + y);
    int[] z = new int[2];
    z[0] = 10;
    z[1] = 20;
    swap(z);
    System.out.println(z[0] + " " + z[1]);
  }
}
```

08-04. Call-by-Value: Objects

- What will be printed on the display when you execute this program?

```
class Employee {
  String name;
  public Employee(String name) {
    this.name = name;
  }
}

public class Ex08_04 {
  public static void changeName(Employee e, String newname) {
    e.name = newName;
  }
  public static void setName(String oldName, String newName) {
    oldName = newName;
  }
  public static void main(String[] args) {
    Employee m = new Employee("Peter");
    System.out.println(m.name);
    changeName(m, "John");
    System.out.println(m.name);
    setName(m.name, "James");
    System.out.println(m.name);
  }
}
```

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr