한양대학교

OBJECT-ORIENTED SYSTEMS DESIGN (Lab8)

Heejin Park

Hanyang University

8-1 (Display 8.1)

Create a class *Sale* defined as follows.

[*Sale*]

1. Create two instance variables as follows.

   private String *name*

   private double *price*

2. Create a constructor **public Sale()** :

   Initialize *name* to “No name yet.” and *price* to 0.

3. Create a constructor **public Sale(String theName, double thePrice)** :

   Invoke **setName(theName)**.

   Invoke **setPrice(thePrice)**.

4. Create a constructor **public Sale(Sale originalObject)** :

   If **originalObject** is null, print "Error: null Sale object." and exit. Otherwise, copy *originalObject.name* to *name* and *originalObject.price* to *price.*

5. Create a method **public Sale clone()** : It returns copy of the object by using the constructor. (textbook 539p.)

6. Create a method **public static void announcement()**:

   Print “This is the Sale class.”.

7. Create a method **public double getPrice()**: It returns *price*.

8. Create a method **public void setPrice(double newPrice)**:

   If **newPrice** is greater than or equal to 0, copy **newPrice** to *price*.
   Otherwise, print “Error: Negative price.” and exit.

9. Create a method **public String getName()**: It returns *name*.

10. Create a method **public void setName(String newName)**:

    If **newName** is neither null or an empty string, copy **newName** to *name.*

    Otherwise, print “Error: Improper name value.” and exit.

11. Create a method **public String toString ()**:

    It returns *name* + “ Price and total cost = $” + *price*.

12. Create a method **public double bill()**: It returns *price*.

13. Create a method **public boolean equalDeals(Sale otherSale):**

    It returns false if **otherSale** is null. Otherwise, it returns whether or not the instance variables *name* are the same and the **bill** for the calling object is equal to the **bill** for **otherSale**. Use the method **equals()** on the next page if needed.

14. Create a method **public boolean lessThan(Sale otherSale):**

    Print "Error: null Sale object." and exit if **otherSale** is null. Otherwise, it returns whether or not the **bill** for the calling object is less than the **bill** for **otherSale**.

15. Create a method **public boolean equals(Object otherObject)**:

It returns false if **otherObject** is null. It returns false if the **getClass** for the calling object is not equal to the **getClass** for **otherObject**. Otherwise, create a variable Sale *otherSale* and store the **otherObject** converted to Sale type in it, and then return whether or not the instance variables *name* are same and *price* are same. Use the method **equals()** if needed.

8-2 (Display 8.2)

Create a class *DiscountSale* as a derived class of the class *Sale*.

[*DiscountSale*]

1. Create a instance variable **private double** *discount*.

2. Create a constructor **public DiscountSale()** :

   Invoke **super()** and initialize *discount* to 0.

3. Create a constructor public **DiscountSale**(String **theName**, double **thePrice**, double **theDiscount**) :

Invoke **super**(**theName**, **thePrice**).

Invoke **setDiscount**(**theDiscount**).

4. Create a constructor public **DiscountSale**(DiscountSale **originalObject**) :

Invoke **super(originalObject**).

Copy *originalObject.discount* to *discount.*

5. Create a method public **DiscountSale clone()** : It returns copy of the object by using the constructor. (textbook 539p.)

6. Create a method **public static void announcement()**:

   Print “This is the DiscountSale class.”.

7. Create a method **public double bill()**:

   Create a variable **double** *fraction* and store *discount/100* in it.

   Return (1- *fraction*) * **getPrice()**.

8. Create a method **public double getDiscount()**: It returns *discount*.

9. Create a method **public void setDiscount(double newDiscount):**

   If **newDiscount** is greater than or equal to 0, copy **newDiscount** to *discount.* Otherwise, print "Error: Negative discount." and exit.

10. Create a method **public String toString ():**

    It returns (**getName()** + " Price = $" + **getPrice()** + " Discount = " + *discount* + "%₩n" + " Total cost = $" + **bill()**).

8-3 (Display 8.3)

Create a class *LateBindingDemo* that prints the output below using the classes *Sale* and *DiscountSale*.

The detailed description of the class is given on the next page.

<output>

```
floor mat Price and total cost = $10.0
floor mat Price = $11.0 Discount = 10.0%
    Total cost = $9.9
Discounted item is cheaper.
cup holder Price and total cost = $9.9
cup holder Price = $11.0 Discount = 10.0%
     Total cost = $9.9
Deals are equal
```

8-3

[*LateBindingDemo*]

Write a method **main**:

1. Create **Sale** ***simple*** by calling **Sale** with 2 parameters “floor mat” and 10.00.
2. Create **DiscountSale** ***discount*** by calling **DiscountSale** with 3 parameters “floor mat”, 11.00, and 10.
3. Print the ***simple*** and the ***discount***.
4. Print the output below by using the method **lessThan**.

<output>

Discounted item is cheaper.

5. Create **Sale** *regularPrice* by calling **Sale** with 2 parameters "cup holder" and 9.90.

6. Create **DiscountSale** *specialPrice* by calling **DiscountSale** with 3 parameters "cup holder", 11.00, and 10.

7. Print the *regularPrice* and the *specialPrice*.

8. Print the output below by using the method **equalDeals**.

<output>

Deals are equal

8-4 (Display 8.4)

Create a class *StaticMethodDemo* that prints the output below using the classes *Sale* and *DiscountSale*.

The detailed description of the class is given on the next page.

<output>

```
This is the Sale class.
This is the DiscountSale class.
That showed that you can override a static method
definition.
This is the Sale class.
This is the DiscountSale class.
No surprises so far, but wait.
discount2 is a DiscountSale object in a Sale variable.
Which definition of announcement() will it use?
This is the Sale class.
It used the Sale version of announcement()!
```

8-4

[*StaticMethodDemo*]

Write a method **main**:

1. Print the output below by using the method **announcement**() preceded by class names.

<output>
This is the Sale class.
This is the DiscountSale class.

2. Print “That showed that you can override a static method definition.”.

3. Create **Sale** ***s*** by calling **Sale** with no parameter.

4. Create **DiscountSale** ***discount*** by calling **DiscountSale** with no parameter.

5. Print the output below by using the method **announcement()** preceded by object names.

<output>
This is the Sale class.
This is the DiscountSale class.

6. Print “No surprises so far, but wait.”.

7. Create **Sale** ***discount2*** and store ***discount*** in it.

8. Print “discount2 is a DiscountSale object in a Sale variable.” and “Which definition of announcement() will it use?”.

9. Print the output below by using the method **discount2.announcement()**.

<output>
This is the Sale class.

10. Print “It used the Sale version of announcement()!”.

8-5 (Display 8.5)

Create a class *CopyingDemo* that prints the output.

The detailed description of the class is given on the next page.

<output>

```
With copy constructors:
a[0] = atomic coffee mug Price and total cost = $130.0
b[0] = atomic coffee mug Price and total cost = $130.0

a[1] = invisible paint Price = $5.0 Discount 10.0%
 Total cost = $4.5
b[1] = invisible paint Price and total cost = $5.0

With clone method:
a[0] = atomic coffee mug Price and total cost = $130.0
b[0] = atomic coffee mug Price and total cost = $130.0

a[1] = invisible paint Price = $5.0 Discount 10.0%
 Total cost = $4.5
b[1] = invisible paint Price = $5.0 Discount 10.0%
 Total cost = $4.5
```

[*CopyingDemo*]

1. Write a method **main**:
    1. Create a **Sale**-typed array *a* of length 2.
    2. Store *a[0]* by calling **Sale** with 2 parameters “atomic coffee mug”, 130.00.
    3. Store *a[1]* by calling **DiscountSale** with 3 parameters “invisible paint”, 5.00, 10.

5. Create a **Sale**-typed array b and store **badCopy(a)** in it.
6. Print the output below.

<output>

With copy constructors:
a[0] = atomic coffee mug Price and total cost = $130.0
b[0] = atomic coffee mug Price and total cost = $130.0

a[1] = invisible paint Price = $5.0 Discount 10.0%
    Total cost = $4.5
b[1] = invisible paint Price and total cost = $5.0

7. Store **goodCopy(a)** in *b*.
8. Print the output below.

<output>

With clone method:
a[0] = atomic coffee mug Price and total cost = $130.0
b[0] = atomic coffee mug Price and total cost = $130.0

a[1] = invisible paint Price = $5.0 Discount 10.0%
    Total cost = $4.5
b[1] = invisible paint Price = $5.0 Discount 10.0%
    Total cost = $4.5

2. Create a method **Sale[] badCopy (Sale[] a)**:

Create a **Sale**-typed array b with length **a**.

Copy **a[i]** to **b[i]** using constructor **Sale**().

Return b.

3. Create a method **Sale[] goodCopy (Sale[] a)**:

Create a **Sale**-typed array b with length **a**.

Copy **a[i]** to **b[i]** using **clone**().

Return b.

8-6 (Display 8.7)

Copy the classes *Employee*, *SalariedEmployee* and *HourlyEmployee* from chapter 7 and modify them as follows.

Create a class *Date* by copying the class *Date* from chapter 4.

[*Employee*]

1. Add modifier “abstract” to the class heading.
2. Create a method **public abstract double getPay**();

   It has no method body.

8-6 (Display 8.7)

3. Modify a method **public boolean equals** (Object otherObject);

It returns false if **otherObject** is null. It returns false if the **getClass** for the calling object is not equal to the **getClass** for **otherObject**. Otherwise, create a variable Employee *otherEmployee* and store the **otherObject** converted to Employee type in it, and then return whether or not the instance variables *name* are same and *hireDate* are same. Use the method **equals()** if needed.

4. Create a method **public samePay(Employee other)** :

If other is null, print "Error: null Employee object." and exit. Otherwise, return whether it is the same pay or not. (It returns true if they have the same pay. Otherwise, return false.) Use the method **getPay()** if needed.

[*SalariedEmployee*]

1. Add “extends Employee” to the class heading.
2. Modify a method **public boolean equals** (Object otherObject);

It returns false if **otherObject** is null. It returns false if the **getClass** for the calling object is not equal to the **getClass** for **otherObject**. Otherwise, create a variable SalariedEmployee *otherSalariedEmployee* and store the **otherObject** converted to SalariedEmployee type in it, and then return whether or not the instance variables *name* are the same and *hireDate* are the same by calling **equals(otherSalariedEmployee)** of the **parent class** and *salary* are the same.

[*HourlyEmployee*]

1. Add “extends Employee” to the class heading.

2. Modify a method **public boolean equals** (Object otherObject);

It returns false if **otherObject** is null. It returns false if the **getClass** for the calling object is not equal to the **getClass** for **otherObject**. Otherwise, create a variable HourlyEmployee *otherHourlyEmployee* and store the **otherObject** converted to HourlyEmployee type in it, and then return whether or not the instance variables *name* are the same and *hireDate* are the same by calling **equals(otherHourlyEmployee)** of the **parent class,** and

instance variables *wageRate* are the same and *hours* are the same.

8-7 (additional example)

Create a class *AbstractClassDemo* that prints the output below using the class *Employee*, *SalariedEmployee*, *HourlyEmployee* and *Date*.

The detailed description of the class is given on the next page.

<output>

```
Name : Joe
Hire Date : January 1, 2015
Pay : $ 8333.333333333334
============================
Name : Sam
Hire Date : February 1, 2016
Pay : $ 2020.0
============================
They don't have the same pay.
```

[*AbstractClassDemo*]

1. Create a method **showEmployee (**Employee **employeeObject)**:

   Print the output below. Use the methods **employeeObject.getName**(), **employeeObject.getHireDate**(), and **employeeObject.getPay**().

   <output>

```
Name : Joe
Hire Date : January 1, 2015
Pay : $ 8333.333333333334
==============================
```

2. Write a method **main**:

    1. Declare **SalariedEmployee type** *joe* by calling **SalariedEmployee** with 3 parameters “Joe”, (“January”, 1, 2015) of **Date** type, and 100000.

    2. Declare **HourlyEmployee type** *sam* by calling **HourlyEmployee** with 4 parameters “Sam”, (“February”, 1, 2016)) of **Date** type, 50.50, and 40.

    3. Print the output in the previous page using **showEmployee() .**

    4. If they have the same pay, print “They have the same pay.”. Otherwise, print “They don't have the same pay.” using **samePay().**