OBJECT-ORIENTED SYSTEMS DESIGN
[Exercise]: Defining Classes I

2022/04/06

Department of Computer Science,
Hanyang University

Lecturer: Taeuk Kim
TA: Taejun Yoon, Seong Hoon Lim, Young Hyun Yoo

Today's Plan

1. **Chapter Review: 25 min.**

2. **Practice: 35 min.**

Class Definition

A Class Is a Type

- **A class is a special kind of programmer-defined type, and variables can be declared of a class type.**
  - A value of a class type is called an object or *an instance of the class*.
  - A class determines the types of data that an object can contain, as well as the actions it can perform.

- **Primitive Type Values vs. Class Type Values**
  - A primitive type value is a single piece of data.
  - A class type value or object can have multiple pieces of data, as well as actions called *methods*.

The Contents of a Class Definition

- **A class definition specifies the data items and methods that all of its objects will have.**
  - These data items methods are sometimes called *members* of the object.
  - Data items are called *fields* or *instance variables*.

```
class Classname {
    [field declarations]
    [method declarations]
  }
```

```
public class DateFirstTry {
    public String Month;
    public int day;
    public int year;

    public void writeOutPut(){
        System.out.println("month"+" "+day+" "+year);
    }
}
```

Instance variables

Methods

An Example of a Class Definition (1)

```
1   import java.util.Scanner;

2   public class Date
3   {
        private String month;
        private int day;
        private int year; //a four digit number.

        public Date()
        {
            month = "January";
            day = 1;
            year = 1000;
        }

        public Date(int monthInt, int day, int year)
        {
            setDate(monthInt, day, year);
        }

        public Date(String monthString, int day, int year)
        {
            setDate(monthString, day, year);
        }
```

public vs. **private** modifiers: The modifier **private** means that an instance variable or method cannot be accessed by name outside of the class.

Instance variables

Constructor: a special kind of method designed to intialize the instance variables for an object.

No-argument constructor

You can invoke another method inside a constructor definition.

An Example of a Class Definition (2)

Overloading: two or more methods *in the same class* have the same method name.

```
21      public Date(int year)
22      {
23          setDate(1, 1, year);
24      }

25      public Date(Date aDate)
26      {
27          if (aDate == null) //Not a real date.
28          {
29              System.out.println("Fatal Error.");
30              System.exit(0);
31          }

32          month = aDate.month;
33          day = aDate.day;
34          year = aDate.year;
35      }
```

A constructor usually initializes all instance variables, even if there is not a corresponding parameter.

Class type variables can be initialized as **`null`** to represent the fact that they do not refer to any real obeject.

An Example of a Class Definition (3)

```
36      public void setDate(int monthInt, int day, int year)
37      {
38          if (dateOK(monthInt, day, year))
39          {
40              this.month = monthString(monthInt);
41              this.day = day;
42              this.year = year;
43          }
44          else
45          {
46              System.out.println("Fatal Error");
47              System.exit(0);
48          }
49      }

50      public void setDate(String monthString, int day, int year)
51      {
52          if (dateOK(monthString, day, year))
53          {
54              this.month = monthString;
55              this.day = day;
56              this.year = year;
57          }
58          else
59          {
60              System.out.println("Fatal Error");
61              System.exit(0);
62          }
63      }
```

The mutator methods, whose names begin with **set**, *are used to reset the data in an object after the object has been created using* **new** *and a constructor.*

A **void** method does not return a value.

Methods

this

The **this** parameter is hidden in the list of the parameters of every method invocation (**the calling object** is automatically plugged in for).

An Example of a Class Definition (4)

```
64      public void setDate(int year)
65      {
66          setDate(1, 1, year);
67      }

68      public void setYear(int year)
69      {
70          if ( (year < 1000) || (year > 9999) )
71          {
72              System.out.println("Fatal Error");
73              System.exit(0);
74          }
75          else
76              this.year = year;
77      }

78      public void setMonth(int monthNumber)
79      {
80          if ((monthNumber <= 0) || (monthNumber > 12))
81          {
82              System.out.println("Fatal Error");
83              System.exit(0);
84          }
85          else
86              month = monthString(monthNumber);
87      }
```

An Example of a Class Definition (5)

```
88      public void setDay(int day)
89      {
90          if ((day <= 0) || (day > 31))
91          {
92              System.out.println("Fatal Error");
93              System.exit(0);
94          }
95          else
96              this.day = day;
97      }

98      public int getMonth()
99      {
100         if (month.equals("January"))
101             return 1;
102         else if (month.equals("February"))
103             return 2;
104         else if (month.equals("March"))
105             return 3;
                . . .
```

<The omitted cases are obvious, but if need be, you can see all the cases in Display 4.2.>

```
                . . .
106         else if (month.equals("November"))
107             return 11;
108         else if (month.equals("December"))
109             return 12;
110         else
111         {
112             System.out.println("Fatal Error");
113             System.exit(0);
114             return 0; //Needed to keep the compiler happy
115         }
116     }
```

Accessor (`getXXX`) and **mutator** (`setXXX`) methods:

- **Accessor** methods allow the programmer to obtain the value of an object's (**private**) instance variables.
- **Mutator** methods allow the programmer to change the value of an object's instance variables in a controlled manner.

An Example of a Class Definition (6)

```
117     public int getDay()
118     {
119         return day;
120     }

121     public int getYear()
122     {
123         return year;
124     }

125     public String toString()
126     {
127         return (month + " " + day + ", " + year);
128     }

129     public boolean equals(Date otherDate)
130     {
131         return ( (month.equals(otherDate.month))
132                  && (day == otherDate.day)
                     && (year == otherDate.year) );
133     }

134     public Boolean precedes(Date otherDate)
135     {
136         return ( (year < otherDate.year) ||
137            (year == otherDate.year && getMonth() <
               otherDate.getMonth()) ||
138            (year == otherDate.year && month.equals(otherDate.month)
139                                    && day < otherDate.day) );
140     }
```

A method that returns a value must specify the **type** of that value in its heading.

The method **`equals`** *of the class* **`String`**

In Java, the **`toString`** and **`equals`** methods are encouraged to be implemented for every class.

Wrapper classes provide a class type corresponding to each of the primitive types. (**Automatic boxing:** **`boolean`** → **`Boolean`**)

An Example of a Class Definition (7)

```
141     public void readInput()
142     {
143         boolean tryAgain = true;
144         Scanner keyboard = new Scanner(System.in);
145         while (tryAgain)
146         {
147             System.out.println("Enter month, day, and year.");
148             System.out.println("Do not use a comma.");
149             String monthInput = keyboard.next();
150             int dayInput = keyboard.nextInt();
151             int yearInput = keyboard.nextInt();
152             if (dateOK(monthInput, dayInput, yearInput) )
153             {
154                 setDate(monthInput, dayInput, yearInput);
155                 tryAgain = false;
156             }

157         else
158             System.out.println("Illegal date. Reenter input.");
159         }
160     }

161     private boolean dateOK(int monthInt, int dayInt, int yearInt)
162     {
163         return ( (monthInt >= 1) && (monthInt <= 12) &&
164                  (dayInt >= 1) && (dayInt <= 31) &&
165                  (yearInt >= 1000) && (yearInt <= 9999) );
166     }

167     private boolean dateOK(String monthString, int dayInt, int
                               yearInt)
168     {
169         return ( monthOK(monthString) &&
170                  (dayInt >= 1) && (dayInt <= 31) &&
171                  (yearInt >= 1000) && (yearInt <= 9999) );
172     }
```

```
public static void foo(int x) {
    if (x == 1){
        System.out.println("x is 1");
        return;
    }
    System.out.println("x is not 1");
}
public static void main(String[] args) {
    foo( x: 1);
}
```

```
x is 1
```

A **`return`** statement **specifies** the value returned and **ends** the method invocation.

An Example of a Class Definition (8)

```
173         private boolean monthOK(String month)
174         {
175             return (month.equals("January") || month.equals("February") ||
176                     month.equals("March") || month.equals("April") ||
177                     month.equals("May") || month.equals("June") ||
178                     month.equals("July") || month.equals("August") ||
179                     month.equals("September") || month.equals("October") ||
180                     month.equals("November") || month.equals("December") );
181         }

182         private String monthString(int monthNumber)
183         {
184             switch (monthNumber)
185             {
186             case 1:
187                 return "January";
                  . . .
```

<The omitted cases are obvious, but if need be, you can see all the cases in Display 4.9.>

```
                  . . .
188             default:
189                 System.out.println("Fatal Error");
190                 System.exit(0);
191                 return "Error"; //to keep the compiler happy
192             }
193         }
194 }
```

"private methods"

The private methods need not be last, but that's as good a place as any.

Creating an Object

A variable of a class type

The new Operator

- **The new Operator**
  - It is used to create an object of a class and associate the object with a variable name.
  - It is combined with a constructor of the class.

- **Syntax**
  - `Classname variable = new Classname();`
  - `Classname variable;`
    `Variable = new Classname();`

- **Example**
  - `Date date1 = new Date();`
  - `Date date1;`
    `date1 = new Date();`

Use of Constructors

```
 1  public class ConstructorsDemo
 2  {
 3      public static void main(String[] args)
 4      {
 5          Date date1 = new Date("December", 16, 1770),
 6               date2 = new Date(1, 27, 1756),
 7               date3 = new Date(1882),
 8               date4 = new Date();
 9          System.out.println("Whose birthday is " + date1 + "?");
10          System.out.println("Whose birthday is " + date2 + "?");
11          System.out.println("Whose birthday is " + date3 + "?");
12          System.out.println("The default date is " + date4 + ".");
13      }
14  }
```

Sample Dialogue

```
Whose birthday is December 16, 1770?
Whose birthday is January 27, 1756?
Whose birthday is January 1, 1882?
The default date is January 1, 1000.
```

The `StringTokenizer` Class

The `StringTokenizer` Class

- **The `StringTokenizer` class is used to recover the words or *tokens* in a multi-word `String`.**
  - You can use whitespace characters to separate each token, or you can specify the characters you wish to use as separators.
  - In order to use the **`StringTokenizer`** class, be sure to include the following at the start of the file:
    - **`import java.util.StringTokenizer;`**

- **There are three constructors:**
  - **`StringTokenizer(String str)`**
    - Delimiters are whitespace characters; any sequence of non-whitespace characters is returned as a token.
  - **`StringTokenizer(String str, String delim)`**
    - Same as above, except you get to specify which characters are delimiters.
  - **`StringTokenizer(String str, String delim, boolean returnDelims)`**
    - Same as above, except you can decide whether you also want the delimiters to be returned as tokens.

The `StringTokenizer` Class

Display 4.17 Some Methods in the Class `StringTokenizer`

The class `StringTokenizer` is in the `java.util` package.

`public StringTokenizer(String theString)`

Constructor for a tokenizer that will use whitespace characters as separators when finding tokens in `theString`.

`public StringTokenizer(String theString, String delimiters)`

Constructor for a tokenizer that will use the characters in the string `delimiters` as separators when finding tokens in `theString`.

`public boolean hasMoreTokens()`

Tests whether there are more tokens available from this tokenizer's string. When used in conjunction with `nextToken`, it returns true as long as `nextToken` has not yet returned all the tokens in the string; returns `false` otherwise.

`public String nextToken()`

Returns the next token from this tokenizer's string. (Throws `NoSuchElementException` if there are no more tokens to return.)[5]

`public String nextToken(String delimiters)`

First changes the delimiter characters to those in the string `delimiters`. Then returns the next token from this tokenizer's string. After the invocation is completed, the delimiter characters are those in the string `delimiters`. (Throws `NoSuchElementException` if there are no more tokens to return. Throws `NullPointer-Exception` if `delimiters` is `null`.)[5]

`public int countTokens()`

Returns the number of tokens remaining to be returned by `nextToken`.

The `StringTokenizer` Class

- **`StringTokenizer` example**

```
public class StkDemo {
    public static void main(String[] args){
        String str = "this is my string";
        StringTokenizer stk = new StringTokenizer(str);
        System.out.println(stk.countTokens());

        while(stk.hasMoreElements()){
            System.out.println(stk.nextToken());
        }
        System.out.println(stk.countTokens());
    }
}
```

```
4
this
is
my
string
0
```

```
public class StkDemo {
    public static void main(String[] args){
        String str = "this%%is%%my%%string";
        StringTokenizer stk = new StringTokenizer(str, delim: "%%");
        System.out.println(stk.countTokens());

        while(stk.hasMoreElements()){
            System.out.println(stk.nextToken());
        }
        System.out.println(stk.countTokens());
    }
}
```

```
4
this
is
my
string
0
```

Practice

Construct a separate class for each problem!

Exercise/WeekN/Practice.java,
/Student.java

Practice for Today

- **Define a `Student` class**
  - It stores the name of a student and his/her date of birth, calculates his/her age, and provides a comparison between the ages of two different students.
  - **Hint: `java.util.Calendar`**

- **Specification: instance variables**
  - **`Student`** class should have 4 private instance variables; **`String name`**, **`int year`**, **`int month`**, and **`int day`**.

- **Specification: constructors**
  - **`Student(String name, int year, int month, int day):`**
    - A constructor that initializes by the name and date of birth.
  - **`Student(String name, int year):`**
    - A constructor that initializes the name and sets the date of birth as a random valid date of the given **`year`**.

Practice for Today

- **Specification: methods**
  - `String getName(),int getYear(), int getMonth(), int getDay():`
    - Accessors that allow access to the corresponding instance variables.
  - `boolean checkDate(parameters of your choice):`
    - Check if the date of birth entered by a user is a valid one. For example, 2008.02.30 is an invalid date.
  - `int calAge(parameters of your choice)`:
    - A method that returns the age of the student based on his/her date of birth.
    - Assume that one becomes 1 year old on his/her first birthday (i.e., we follow the **international rule**, not the Korean rule).
  - `boolean isOlder(Student stu):`
    - Return `true` if the birth date of the calling object is **earlier** than that of the student given as a parameter. Otherwise, return `false`.

Practice for Today

- **Define a class `Practice`**
  - **`public static void main(String args[]):`**
    - Accept the information of two students as input.
      - Use the **`StringTokenizer`** to parse the input.
    - Each student should be represented as **`name year.month.day`** or **`name.`**
      - For the first student, enter both name and date of birth. ex) **Yun 2000.10.10.**
      - For the second student, the program only receives his/her name. We assume that the second student was born in the same year as the first student and that his/her exact date of birth has been randomly determined.
      - Every date of birth must be valid.
    - Print the information of each student including his/her age.
    - Compare the birth dates of the two students to print out which student is older.

- **Exemplar**

```
kim 2001.03.05
yun
kim 2001/3/5 age :21
yun 2001/4/28 age :20
kim is older than yun
```

```
kim 2000.02.30
yun
invalid input
```

Time for Practice

Get it started, and ask TAs if you are in a trouble.