

Object – Oriented Programming

Lab #6

● Contents

- Arrays
- Multidimensional Arrays
- Enumerated Types

CSLAB

● Arrays

• Background

- Programmers often need the ability to represent a group of values as a list
 - List may be one-dimensional or multidimensional
- An ***array*** is a data structure used to process a collection of data that is all of the same type

CSLAB

● Basic terminology

- An array is composed of *elements*
- Elements in an array have a *common name*
 - Example: $a[3] = 5$;
 - The common name is 'a'
- The *entire array* is referenced through the common name
- Array elements are of the same type – the base type
- Elements of an array are referenced by *subscribing* (indexing) the common name

CSLAB

● Arrays (Contd)

• Defining arrays

– *char[] c;*

c 

Defines the Array identifier **c**

Array object variable **c** is
un-initialized

– *int[] value = new int[10];*

value  →

0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

Defines and initializes the Array

Array object variable **value**
references a new ten element
list of integers

Each of the integers is default
initialized to 0

• Generally declare Arrays in java as follows

BaseType[] ArrayName = new BaseType[size];

CSLAB

● Arrays (Contd)

- **Creating an array:**

```
int[] foo = new int[10];
```

- **Accessing an array:**

```
foo[3] = 7;  
System.out.println(foo[1]);
```

- **Creating an array:**

```
String[] bar = new String[10];
```

- **Accessing an array:**

```
bar[3] = "qux";  
System.out.println(bar[1]);
```

CSLAB

● An array example

```
int[] v = new int[10];
```

```
int i = 7;
```

```
int j = 2;
```

```
int k = 4;
```

```
v[0] = 1;
```

```
v[i] = 5;
```

```
v[j] = v[i] + 3;
```

```
v[j+1] = v[i] + v[0];
```

```
v[j+2] = 3;
```

```
v[8] = 12;
```

v	1	0	8	6	3	0	0	5	12	0
	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

CSLAB

● Java array features

- Base (element) type can be any type
- Size of array can be specified at run time
double[] score = new double[count];
- Index type is integer and the index range must be 0 to n-1
 - Where n is the *number of elements*
 - Just like Strings indexing!
- Automatic bounds checking
 - Ensures any reference to an array element is valid
- Array is an object
 - Has features common to all other objects

CSLAB

● Explicit initialization

- An array can be initialized when it is declared
- Example

```
String[] puppy = {"pika", "mila", "arlo", "mikki"};
```

```
int[] unit = { 1 };
```

- Equivalent to

```
String[] puppy = new String[4];  
puppy[0] = "pika";   puppy[1] = "mila";  
puppy[2] = "arlo";   puppy[3] = "nikki";
```

```
int[] unit = new int[1];  
unit[0] = 1;
```

CSLAB

● Variable-size Declaration

- In Java, we are not limited to fixed-size array declaration
- The following code prompts the user for the size of an array and declares an array of designated size:

```
Scanner scanner = new Scanner(System.in);  
int size;  
int[] number;  
  
System.out.print("Size of an array:");  
size= scanner.nextInt( );  
  
number = new int[size];
```

CSLAB


● Array length

- Member *length*

- Size of the array

```
for(int i = 0; i < puppy.length; ++i) {  
    System.out.println(puppy[i]);  
}
```

- Note that **length** is a *field*, not a method!

- i.e., It is **not** puppy.length(); 

CSLAB

● Array manipulation

- The most natural way to manipulate items in an array is to use the *for loop*

- Basic for loop

```
for(int index = 0; index < arr.length; index++) {  
    //perform array manipulation tasks here  
    System.out.println(arr[index]);  
}
```

- The *for loop* uses an index from 0 to the end of the array
- This is useful initializing your array with default values
- For example 0's

0	0	0	0	0	0
---	---	---	---	---	---

CSLAB

● Array manipulation (Contd)

- You can use other loops with array

- While loop

```
int i = 0;  
while(i < arr.length) {  
    System.out.println(arr[i]);  
}
```

- Do While loop

```
int j = 0;  
do {  
    System.out.println(arr[j]);  
    j++;  
} while (j < arr.length);
```

CSLAB

● An array example

```
import java.util.Scanner;

public class ArrayOfScores2
{
    /**
     Reads in 5 scores and shows how much each
     score differs from the highest score.
     */
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        double[] score = new double[5];
        int index;
        double max;

        System.out.println("Enter " + score.length + " scores:");
        score[0] = keyboard.nextDouble();
        max = score[0];
        for (index = 1; index < score.length; index++)
        {
            score[index] = keyboard.nextDouble();
            if (score[index] > max)
                max = score[index];
            //max is the largest of the values score[0],..., score[index].
        }

        System.out.println("The highest score is " + max);
        System.out.println("The scores are:");
        for (index = 0; index < score.length; index++)
            System.out.println(score[index] + " differs from max by "
                               + (max - score[index]));
    }
}
```

Enter 5 scores:

34

56

78

35

68

The highest score is 78.0

The scores are:

34.0 differs from max by 44.0

56.0 differs from max by 22.0

78.0 differs from max by 0.0

35.0 differs from max by 43.0

68.0 differs from max by 10.0

● Arrays and References

- Like class types, a variable of an array type holds a *reference*
 - Arrays are objects
 - A variable of an array type holds the address of where the array object is stored in memory
 - Array types are (usually) considered to be class types

CSLAB

● Use of = and == with Arrays

- Because an array variable contains the memory address of the array it names, the assignment operator (=) only copies this memory address
 - It does not copy the values of each indexed variable
 - Using the assignment operator will make two array variables be different names for the same array
 - $\mathbf{b = a;}$
 - The memory address in \mathbf{a} is now the same as the memory address in \mathbf{b} : They reference the same array

CSLAB

● Use of = and == with Arrays (Contd)

- A *for loop* is usually used to make two different arrays have the same values in each indexed position:

```
int i;
```

```
for (i = 0; (i < a.length) && (i < b.length); i++)
```

```
    b[i] = a[i];
```

- Note that the above code will not make *b* an exact copy of *a*, unless *a* and *b* have the same length

CSLAB

• Arrays with a Class Base Type

- The base type of an array can be a class type

Date[] holidayList = new Date[20];

- The above example creates 20 indexed variables of type *Date*
 - It does not create 20 objects of the class *Date*
 - Each of these indexed variables are automatically initialized to *null*
 - Any attempt to reference any them at this point would result in a "null pointer exception" error message

CSLAB

● Arrays with a Class Base Type (Contd)

- Like any other object, each of the indexed variables requires a separate invocation of a constructor using *new* (singly, or perhaps using a *for loop*) to create an object to reference

```
holidayList[0] = new Date();
```

...

```
holidayList[19] = new Date();
```

OR

```
for(int i = 0; i < holidayList.length; i++)
```

```
    holidayList[i] = new Date();
```

- Each of the indexed variables can now be referenced since each holds the memory address of a *Date* object

CSLAB

Arrays (Contd)

```
package classTest;

public class Test {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        ID[] T1 = new ID[10];
        //T1[0] = new ID();
        T1[0].setId(3);

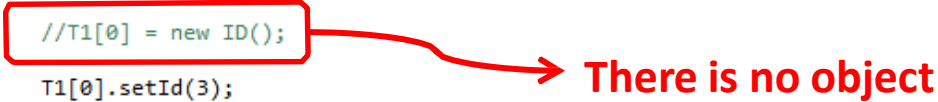
        System.out.println(T1[0].getId());
    }
}

class ID{

    private int id;

    public void setId(int id){
        this.id = id;
    }

    public int getId(){
        return this.id;
    }
}
```



Problems @ Javadoc Console

<terminated> Test (1) [Java Application] C:\Program Files\Java\jre7\bin\w
Exception in thread "main" [java.lang.NullPointerException](#)
at classTest.Test.main([Test.java:15](#))

There is no object

CSLAB

Arrays (Contd)

```
package classTest;

public class Test {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        ID[] T1 = new ID[10];
        T1[0] = new ID();
        T1[0].setId(3);

        System.out.println(T1[0].getId());
    }
}

class ID{

    private int id;

    public void setId(int id){
        this.id = id;
    }

    public int getId(){
        return this.id;
    }
}
```

```
<terminated> Test (1) [Java Application] C:\#Pro
3
```

Creates an object

CSLAB

● Passing an Array as a parameter

```
package classTest;

public class Test {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        int[] arr = new int[10];

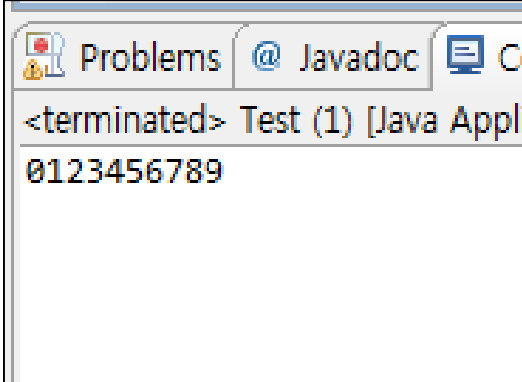
        insertToArr(arr);
        for(int i=0; i<arr.length; i++)
            System.out.print(arr[i]);

    }
    public static void insertToArr(int[] parameterArr){

        for(int i=0; i<parameterArr.length; i++)
            parameterArr[i]=i;

    }

}
```



<terminated> Test (1) [Java Appli
0123456789

To pass an array as a parameter
simply declare the identifier.

You do not need to enter the size
of the array

CSLAB

● Methods That Return an Array

- In Java, a method may also return an array
 - The return type is specified in the same way that an array parameter is specified

```
public static int[] incrementArray(int[] a, int increment) {  
    int[] temp = new int[a.length];  
    int i;  
    for (i=0; i < a.length; i++) {  
        temp[i] = a[i] + increment;  
    }  
    return temp;  
}
```

CSLAB

● Self-Test (1)

- 시험 점수의 평균값과 각 점수의 평균값과의 차이값을 출력하는 TestScores 클래스를 구현할 것
- scores 배열에 입력값을 입력 받는 fillArray 메소드를 작성할 것
 - 점수 배열을 인자로 받음
 - 음수값을 입력 받거나, 입력값의 개수가 배열 최대 크기를 초과할 경우 종료
 - 입력값의 개수를 반환함
- 각 점수와 평균값의 차이값을 반환하는 showDifference 메소드를 작성할 것
 - 점수 배열과 입력값의 개수를 인자로 받음
 - 메소드 내에서 평균값을 computeAverage 메소드를 호출함
 - 이를 이용하여 각 점수와 평균값과의 차이값을 출력함

CSLAB

• Self-Test (1) (Contd)

- scores의 평균값을 반환하는 computeAverage

- 점수 배열과 입력값의 개수를 인자로 받음
- 점수 배열내 원소의 총합과 인자로 받은 입력값의 개수를 나누어 평균값을 반환함

```
This program reads test scores and shows
how much each differs from the average.
Enter test scores:
Mark the end of the list with a negative number.
100
45
75.5
10
2
99
-100
Average of the 6 scores = 55.25
The scores are:
100.0 differs from average by 44.75
45.0 differs from average by -10.25
75.5 differs from average by 20.25
10.0 differs from average by -45.25
2.0 differs from average by -53.25
99.0 differs from average by 43.75
```

LAB

• Multidimensional arrays

- Many problems require information be organized as a two-dimensional or multidimensional list
- Examples
 - Matrices
 - Graphical animation
 - Economic forecast models
 - Map representation
 - Time studies of population change
 - Microprocessor design

CSLAB

● Multidimensional arrays (Contd)

- A multidimensional array is basically an array of arrays
- In a multidimensional array each array item represents the pointer to another array

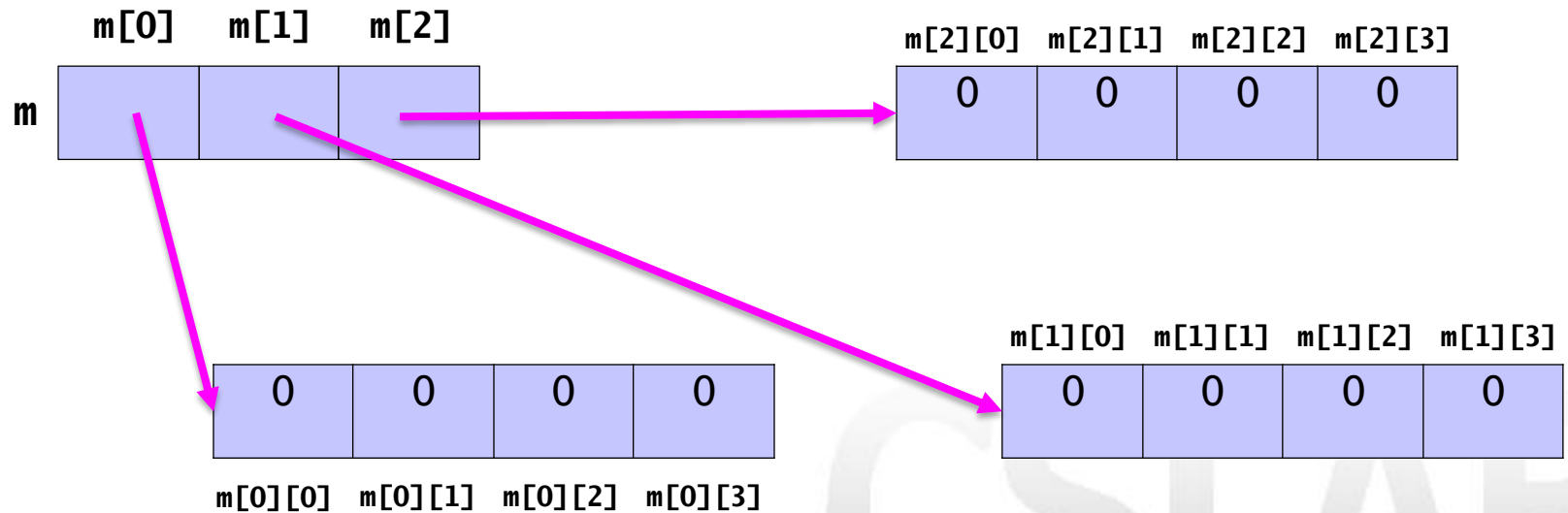
CSLAB

● Example

• Segment

int[][] m = new int[3][4];

• Products



● Multidimensional array visualization

- A multi-dimensional array declaration

declaring the array

```
int[][] m = new int[3][4];
```

accessing the array

```
m[2][1] = 4;
```

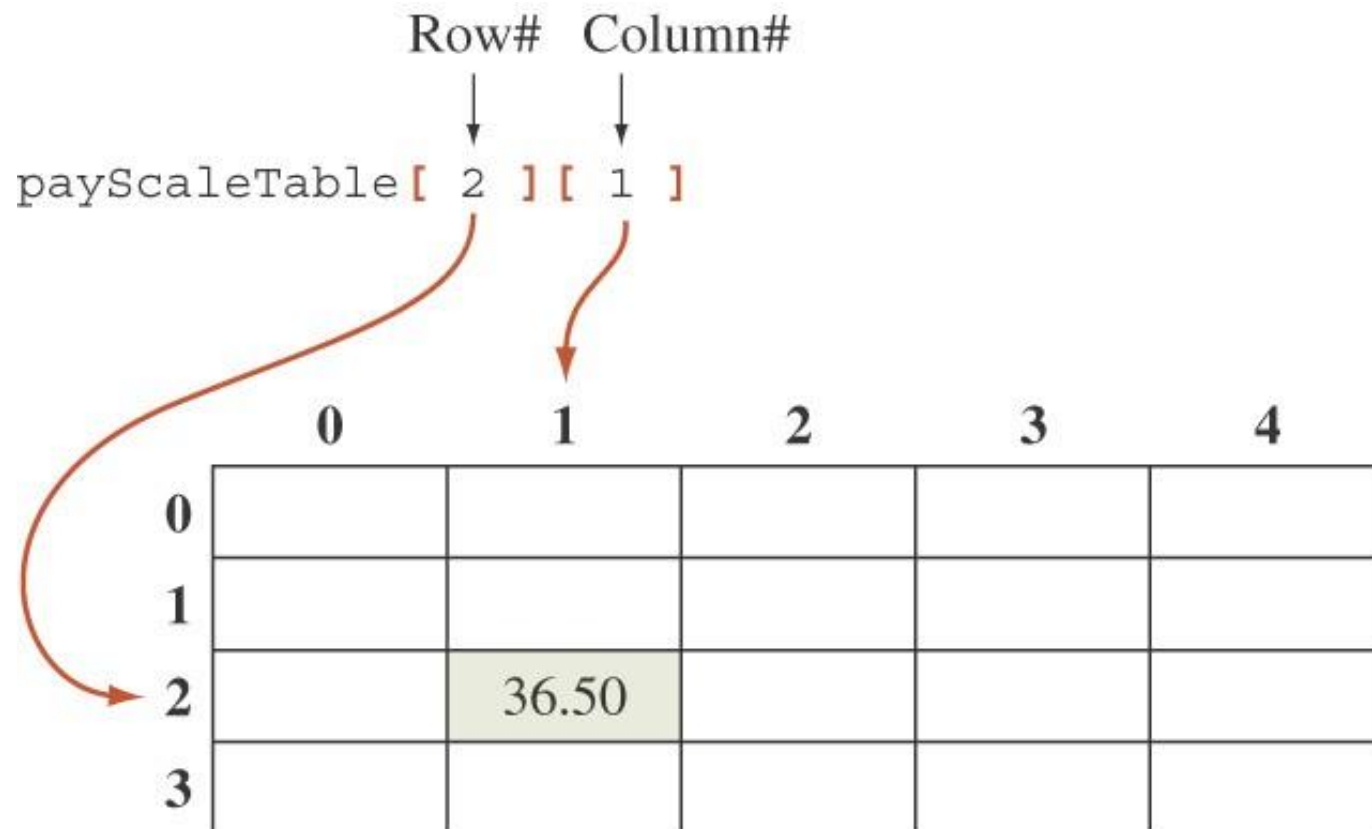
- How we visualize it:

0	0	0	0
0	0	0	0
0	4	0	0

LAB

● Accessing an Element

- An element in a two-dimensional array is accessed by its row and column index



● Sample 2-Dimensional Array Processing

- Multidimensional arrays are usually processed using nested loops
- The outer loop iterates the rows
- The inner loop iterates the items of each row (*i.e. column*)

```
double[ ] average = { 0.0, 0.0, 0.0, 0.0 };

for (int i = 0; i < payScaleTable.length; i++) {
    for (int j = 0; j < payScaleTable[i].length; j++) {
        average[i] += payScaleTable[i][j];
    }

    average[i] = average[i] / payScaleTable[i].length;
}
```

CSLAB

● Sample 2-Dimensional Array Processing (Contd)

- The basic structure of 2D array processing code

```
arrayType[][] arrayName = new arrayType[numRows][numCols]

for (int i = 0; i < arrayName.length; i++) {

    for (int j = 0; j < arrayName[i].length; j++) {

        //Perform actions in here
        // e.g. System.out.print(arrayName[i][j])
    }
}
```

CSLAB

• Enumerated Types

- Starting with version 5.0, Java permits enumerated types
 - An enumerated type is a type in which **all the values are given** in a (typically) short list
- The definition of an enumerated type is normally placed outside of all methods in the same place that named constants are defined:

enum TypeName (VALUE_1, VALUE_2,, VALUE_N);

- Note that a value of an enumerated type is a **kind of named constant** and so, by convention, is spelled with all uppercase letters
- As with any other type, variables can be declared of an enumerated type

CSLAB

• Enumerated Types Example

- Given the following definition of an enumerated type:

```
enum WorkDay {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY};
```

- A variable of this type can be declared as follows:

```
WorkDay meetingDay, availableDay;
```

- The value of a variable of this type can be set to one of the values listed in the definition of the type, or else to the special value *null*:

```
meetingDay = WorkDay.THURSDAY;  
availableDay = null;
```

CSLAB

• Enumerated Types Usage

- Just like other types, variable of this type can be declared and initialized at the same time:
WorkDay meetingDay = WorkDay.THURSDAY;
 - Note that the value of an enumerated type must be prefaced with the name of the type
- The value of a variable or constant of an enumerated type can be output using *println*
 - The code
System.out.println(meetingDay);
 - Will produce the following output:
THURSDAY
 - As will the code:
System.out.println(WorkDay.THURSDAY);
 - Note that the type name *WorkDay* is not output

CSLAB

• Enumerated Types Usage (Contd)

- Although they may look like String values, values of an enumerated type are not String values
- However, they can be used for tasks which could be done by String values and, in some cases, work better
 - Using a String variable allows the possibility of setting the variable to a nonsense value
 - Using an enumerated type variable **constrains the possible values** for that variable
 - An error message will result if an attempt is made to give an enumerated type variable a value that is not defined for its type

CSLAB

• Enumerated Types Usage (Contd)

- Two variables or constants of an enumerated type can be compared using the *equals* method or the `==` operator
- However, the `==` operator has a nicer syntax

```
if (meetingDay == availableDay)
```

```
    System.out.println("Meeting will be on schedule.");
```

```
if (meetingDay == WorkDay.THURSDAY)
```

```
    System.out.println("Long weekend!");
```

CSLAB

● An Enumerated Type Example

Display 6.13 An Enumerated Type

```
1  public class EnumDemo
2  {
3      enum WorkDay {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY};
4
5      public static void main(String[] args)
6      {
7          WorkDay startDay = WorkDay.MONDAY;
8          WorkDay endDay = WorkDay.FRIDAY;
9
10         System.out.println("Work starts on " + startDay);
11         System.out.println("Work ends on " + endDay);
12     }
13 }
```

SAMPLE DIALOGUE

Work starts on MONDAY
Work ends on FRIDAY

Some Methods included with Every Enumerated Type

Display 6.14 Some Methods Included with Every Enumerated Type

```
public boolean equals(Any_Value_Of_An_Enumerated_Type)
```

Returns true if its argument is the same value as the calling value. While it is perfectly legal to use `equals`, it is easier and more common to use `==`.

EXAMPLE

For enumerated types, `(Value1.equals(Value2))` is equivalent to `(Value1 == Value2)`.

```
public String toString()
```

Returns the calling value as a string. This is often invoked automatically. For example, this method is invoked automatically when you output a value of the enumerated type using `System.out.println` or when you concatenate a value of the enumerated type to a string. See Display 6.15 for an example of this automatic invocation.

EXAMPLE

`WorkDay.MONDAY.toString()` returns "MONDAY".
The enumerated type `WorkDay` is defined in Display 6.13.

(continued)

Some Methods included with Every Enumerated Type (Contd)

Display 6.14 Some Methods Included with Every Enumerated Type

```
public int ordinal()
```

Returns the position of the calling value in the list of enumerated type values. The first position is 0.

EXAMPLE

`WorkDay.MONDAY.ordinal()` returns 0, `WorkDay.TUESDAY.ordinal()` returns 1, and so forth. The enumerated type `WorkDay` is defined in Display 6.13.

```
public int compareTo(Any_Value_Of_The_Enumerated_Type)
```

Returns a negative value if the calling object precedes the argument in the list of values, returns 0 if the calling object equals the argument, and returns a positive value if the argument precedes the calling object.

EXAMPLE

`WorkDay.TUESDAY.compareTo(WorkDay.THURSDAY)` returns a negative value. The type `WorkDay` is defined in Display 6.13.

CSLAB

Some Methods included with Every Enumerated Type (Contd)

```
public EnumeratedType[] values()
```

Returns an array whose elements are the values of the enumerated type in the order in which they are listed in the definition of the enumerated type.

EXAMPLE

See Display 6.15.

```
public static EnumeratedType valueOf(String name)
```

Returns the enumerated type value with the specified name. The string name must be an exact match.

EXAMPLE

`WorkDay.valueOf("THURSDAY")` returns `WorkDay.THURSDAY`. The type `WorkDay` is defined in Display 6.13.

CSLAB

Programming Tip: Enumerated Types in *switch* Statements

- Enumerated types can be used to control a *switch* statement
 - The *switch* control expression uses a variable of an enumerated type
 - Case labels are the unqualified values of the same enumerated type
- The enumerated type control variable is set by using the static method *valueOf* to convert an input string to a value of the enumerated type
 - The input string must contain all upper case letters, or be converted to all upper case letters using the *toUpperCase* method

CSLAB

Enumerated Types in *switch* Statements

Example

Display 6.16 Enumerated Type in a switch Statement

```
1  import java.util.Scanner;
2
3  public class EnumSwitchDemo
4  {
5      enum Flavor {VANILLA, CHOCOLATE, STRAWBERRY};
6
7      public static void main(String[] args)
8      {
9          Flavor favorite = null;
9          Scanner keyboard = new Scanner(System.in);
```

(continued)

CSLAB

Enumerated Types in *switch* Statements

Example (Contd)

Display 6.16 Enumerated Type in a switch Statement

```
10      System.out.println("What is your favorite flavor?");
11      String answer = keyboard.next();
12      answer = answer.toUpperCase();
13      favorite = Flavor.valueOf(answer);

14      switch (favorite)
15      {
16          case VANILLA:
17              System.out.println("Classic");
18              break;
19          case CHOCOLATE:
20              System.out.println("Rich");
21              break;
22          default:
23              System.out.println("I bet you said STRAWBERRY.");
24              break;
25      }
26  }
27 }
```

The case labels must have just the name of the value without the type name and dot.

(continued)

Enumerated Types in *switch* Statements

Example (Contd)

Display 6.16 Enumerated Type in a switch Statement

SAMPLE DIALOGUE

What is your favorite flavor?
Vanilla
Classic

SAMPLE DIALOGUE

What is your favorite flavor?
STRAWBERRY
I bet you said STRAWBERRY.

SAMPLE DIALOGUE

What is your favorite flavor?
PISTACHIO

This input causes the program to end and issue an error message.

CSLAB

● Self-Test (2)

- 각 평일마다 근무 시간을 입력 받은 뒤, 한 주 동안 일한 시간을 출력하는 **EnumValuesDemo** 클래스를 작성할 것
 - MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY 값을 가지는 enum type을 정의할 것
 - 정의한 enum type의 배열 변수를 선언할 것
 - 해당 배열 변수를 이용하여 각 평일마다 근무 시간을 입력 받을 것 (총 5번을 입력 받아야 함)
 - 총 근무 시간을 출력할 것

CSLAB