Week 4, Lab B – Switch Statements and Enumerated Types (Enums)

# Lab Intro & Prep

## Learning Objectives

* Develop programs with multiple-selection switch statements
* Produce more readable code by combining switch statements with enumerated types

# Exercise 1 – Calculator

Create a class named **Calculator** with a main method. Prompt the user for two numbers (these can be stored as doubles to account for fractional parts). Next, prompt the user for an operation to perform (plus, subtract, multiply, divide), using the appropriate character symbols (+, -, \*, /) – this operation can be stored as a character (see hints below).

Using a switch statement, perform the appropriate operation in the relevant switch case, and output the result to the console (Fig 1).

**Hints:**

* You will want to store the selected operation and collect the input using the **char** data type
* To fetch a character from the input stream, we can use the Scanner object’s **next()** method to return the next string, and immediately call **charAt(0)** to fetch the first character. Note that Strings also have zero-based indexing
* E.g. if your Scanner variable is named input, the next character can be fetched via:

input.next().charAt(0);

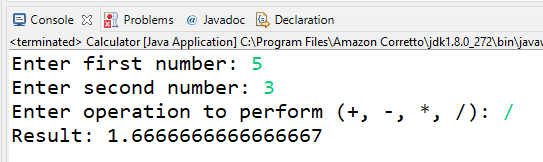


Fig 1 - Exercise 1 Example Output

# Exercise 2 – Calendar

Create a class named **Calendar** with a main method. Prompt the user to enter an integer which corresponds to a month and store this in an integer. **Using a switch statement**, print out the number of days in the month the user entered (Fig 2). You can assume February has 28 days for this exercise.

If the user enters a number which does not fall between the valid range (1-12), you should inform them their input is incorrect (Fig 3).

**Hints:**

* Do you require 12 individual case statements? Months with identical number of days can be ‘chained’ together (omitting the break keyword)

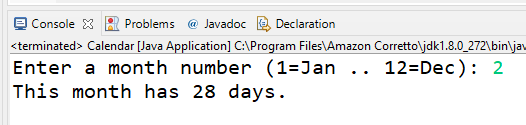


Fig 2 - Exercise 2 Example Output (Valid Month)

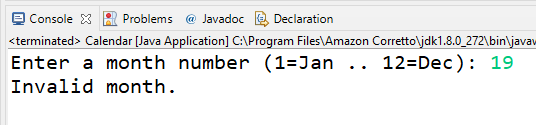


Fig 3 - Exercise 2 Example Output (Invalid Month)

# Exercise 3 – Weather

Create a new class named **Weather**. The weather can be in one of three states **[SUNNY, OVERCAST, RAINING]**, create an enumerated set (call the type State) and create a variable to store the current weather. You may initialise this variable to an initial value e.g. Sunny:

State currentState = State.***SUNNY***;

Allow a user to indicate if the weather is improving or deteriorating, a suitable message should be output each time the state changes. The application should allow user input (a char will do) until an unrecognised char is entered (Fig 4).

E.g. If the current state of the weather is Sunny, improving this state (entering ‘i’) will not result in a change (as Sunny is the ‘best’ weather). If the current state is Sunny and we deteriorate the state (entering ‘d’), the state should change to Overcast.

See if you can develop the program from scratch – if you are struggling, there is some start code provided in Listing 1.

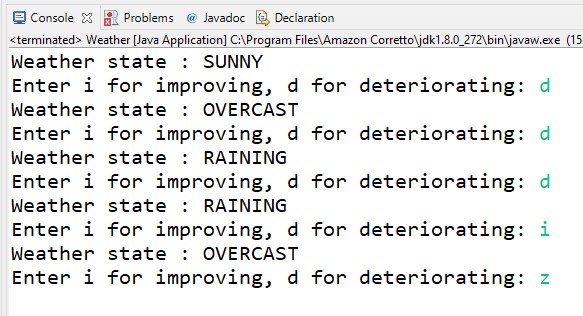


Fig 4 - Exercise 3 Example Output

Listing 1 - Weather Starter Code

|  |
| --- |
| **import** java.util.Scanner;  **enum** State {  ***SUNNY***, ***OVERCAST***, ***RAINING***;  }  **public** **class** Weather {  **public** **static** **void** main(String[] args) {  Scanner input = **new** Scanner(System.***in***);  State currentState = State.***SUNNY***;  **boolean** finished = **false**;  **char** command; // will store current command (i/d)  **while** (!finished) {  System.***out***.println("Weather state: " + currentState);  System.***out***.print("Enter i for improving, d for deteriotating: ");  command = input.next().charAt(0); // fetch 1st character at index 0  **switch** (command) {  **case** 'i':  currentState = *improveState*(currentState);  **break**;  **case** 'd':  currentState = *worsenState*(currentState);  **break**;  **default**:  finished = **true**; // program ends if i or d not entered  }  }  input.close();  }  **public** **static** State improveState(State currentState) {  // Finish implementing this method - it should return the next improved  // state using a switch statement  **return** currentState; // this is here so the code compiles  }  **public** **static** State worsenState(State currentState) {  // Finish implementing this method - it should return the next worse state  // using a switch statement  **return** currentState; // this is here so the code compiles  }  } |