Week 8 – Switch Statements and Enumerated Types (Enums)

# Learning Objectives

* Restrict the value of a variable to a list of permittable types using an **enumerated type (enum)**
* Implement multi-selection decision-making in applications using a **switch statement**

# Preparation

In this webinar, we will be creating multiple classes in IntelliJ and learning about two new features that can be used in your assessment – **switch statements** and **enumerated types** (enums).

We will be creating a class to model a **Pizza**, and will refactor the code along the way to make use of switch statements and enums.

# Step 1 – Creating our Classes

1. We’ll start by creating a class with a main method, which will be empty to begin with, but we will revisit it a little later.
2. Next, we’ll add a Pizza class which will represent a Pizza, the Pizza class will have the following in it:
   1. variables
      1. String name
      2. int size
      3. ArrayList<String> toppings
      4. boolean isVegetarian
   2. constructor (accepts a String for the name, and an int for the size)

# Step 2 – Creating a Pizza Object

Now that we have a class to present a Pizza data type, we can create an object of it. In the main method, create a Pizza object by calling the constructor (pass in a name for the pizza and an int for the size in inches).

# Step 3 – Pizza methods

Let’s now add some methods to our Pizza class so that we can add a topping and print out all the toppings currently on the pizza object.

1. Provide an **addTopping** method which accepts a String for a topping and then adds that to the pizza object’s ArrayList. The method doesn’t need to return anything (so should be void).
2. Add a few toppings to your pizza that are suitable for the type/name of the pizza. E.g. if the name of your pizza is Chicken & Mushroom pizza object, you can call the **addTopping** method multiple times, once to add Chicken, another time to add Mushroom, and any other toppings you like on your pizza.
3. Provide a **printToppings** method that loops through the toppings ArrayList and prints out each of the toppings currently on the pizza. Call this method in your main method after the lines of code that added the toppings.

# Step 4 – Making use of Enumerated Types (Enums)

The problem with the addTopping method is that accepts *any* String passed to it. This String could be “Chicken”, “Mushroom”, “Pineapple”, or something completely random such as “q212]e”.

This is a good use case for an **enumerated type**, often just called an **enum** for short. An enum allows us to provide a list of permitted values for a type. E.g.

enum Season { *SPRING*, *SUMMER*, *AUTUMN*, *WINTER* }

enum Apple { *GRANNY\_SMITH*, *GOLDEN\_DELICIOUS*, *PIPPIN* }

enum Planet { *MERCURY*, *VENUS*, *EARTH*, *MARS*, *JUPITER*, *SATURN*, *URANUS*, *NEPTUNE* }

enum TrafficLightColour { *RED*, *AMBER*, *GREEN* }

Create an enum named **Topping**. An enum is created in the same way a class is in IntelliJ, except we select **Enum** instead of **Class**:

Graphical user interface, text, application

Description automatically generated

In your enum Toppins body { }, provide valid values for the Topping data type (list as many toppings as you like), e.g.:

public enum Topping { *CHICKEN*, *MUSHROOM*, *GREEN\_PEPPERS*, *ONIONS*}

We can now refactor our code to make use of enums.

1. Modify the **addTopping** method so that it accepts a Topping instead of a String
2. Modify the ArrayList so that it is an ArrayList<Topping> instead of an ArrayList<String>
3. Modify the **printToppings** method too so that it refers to the Topping data type
4. Modify the calls to **addTopping** (so we’re passing in Toppings and not Strings)

# Step 5 – Determining if a Pizza is Suitable for Vegetarians

In your addTopping method, after adding a topping, determine if the pizza is suitable for vegetarians by checking if the topping just added is a meat-type product, if it is, set **isVegetarian** to false;

We can use if statements here with or || operators, or we can make use of another type of multi-selection statement available to us in Java known as the **switch statement**.

An example of a switch statement can be seen below, where a variable of type **int** named **dayNum** is inspected by the switch statement, and depending on the value of dayNum, a different case (set of statements) is executed.

int dayNum = 3;

switch (dayNum) {  
 case 1:  
 System.*out*.println("Monday");  
 break;  
 case 2:  
 System.*out*.println("Tuesday");  
 break;  
 case 3:  
 System.*out*.println("Wednesday");  
 break;  
 case 4:  
 System.*out*.println("Thursday");  
 break;  
 case 5:  
 System.*out*.println("Friday");  
 break;  
 case 6:  
 System.*out*.println("Saturday");  
 break;  
 case 7:  
 System.*out*.println("Sunday");  
 break;  
 default: *// default case executes if no other case did* System.*out*.println("Invalid number for day");  
} *// end of switch statement*

If certain cases need to execute the same code, we can **combine** those cases together by omitting the break keyword. This is called fall-through – meaning we don’t need to have duplicate logic in all of the different cases if those cases result in the same thing happening.

int dayNum = 6;  
  
switch (dayNum) {  
 case 1:  
 case 2:  
 case 3:  
 case 4:  
 case 5:  
 System.*out*.println("Weekday");  
 break;  
 case 6:  
 case 7:  
 System.*out*.println("Weekend");  
 break;  
 default: *// default case executes if no other case did* System.*out*.println("Invalid day number");  
} *// end of switch statement*

A switch statement can operate on other data types, such as **String**, **char**, **short**, and even an **enum.**

Let’s make use of a switch statement in our **addTopping** method to inspect the topping just added, and in the case that the topping is meat, set the **isVegetarian** boolean to false;

