Switch expression

Java 12 introduced a new feature into the Java language called switch expressions, which can be used to simplify many switch statements.

Switch statements are often used to avoid long chains of if and else if statements, generally making your code more readable.

Demerits of earlier switch:

That being said, switch statements can be verbose in their own way, and the strict requirements for placing break statements often make them error-prone.

Switch statements vs switch expressions

The main difference between a switch expression and a switch statement is that while a switch statement can be used to update the value of a predefined variable, a switch expression is assigned to a variable.

This is possible because a switch expression evaluates to a specific value.

Additionally, switch expressions introduced a new **arrow syntax** that condenses the code, makes it more readable, and eliminates the need for break statements.

We begin with an enumeration of various things that are going to be taste-tested.
Our switch statement and switch expressions are going to assign a taste rating as an integer from 1 to 10, with 1 being utterly disgusting and 10 being absolutely delicious.

First, we'll look at how this would commonly be written as a switch statement:
private enum ThingsToTaste {PIZZA, BROCCOLI, STEAK, SUGAR, DIRT, MEATBALLS, CHOCOLATE}

```
int tasteValue = 0;
ThingsToTaste taste = ThingsToTaste.DIRT;

switch (taste) {
    case SUGAR:
    case PIZZA:
    case CHOCOLATE:
        tasteValue = 10;
        break;
    case MEATBALLS:
    case STEAK:
        tasteValue = 7;
```

```
        break;
    case BROCCOLI:
        tasteValue = 4;
        break;
    case DIRT:
        tasteValue = 1;
        break;
    default:
        throw new IllegalStateException("Invalid tastable object: " + taste);
}
System.out.println(taste + ": " + tasteValue);
```

If you have IntelliJ set to use Java 12 or higher, it will convert your switch statement into a switch expression with just one click.

```
int tasteValue = switch (taste) {
    case SUGAR, PIZZA, CHOCOLATE -> 10;
    case MEATBALLS, STEAK -> 7;
    case BROCCOLI -> 4;
    case DIRT -> 1;
    default -> throw new IllegalStateException("Invalid tastable object: " + taste);
};
```

As you can see, this is way shorter. Let's go through what changed! First, the tasteValue variable did not have to be initialized before the switch. Instead, the entire switch expression is assigned to be the value of tasteValue. This works because the switch expression will ultimately yield an integer value.

he next major difference is that when multiple case statements yield the same value, they can all be combined into one line. SUGAR, PIZZA, and CHOCOLATE all yield 10, so we can simply write case SUGAR, PIZZA, CHOCOLATE -> 10;

Next, note that the break statements are gone! The new arrow syntax replaces the need for both the : after case and the break at the end of the case statement.

The arrow signals that once the value is reached it is to be assigned to the tasteValue variable and then stop. We no longer have to explicitly state the full assignment expression; just stating the value is enough. The JVM knows to set the integer value to tasteValue.

We can still have a default case at the end as a fallback option.
Also notice that in this example the default case does not return an integer, but instead throws an exception.
In fact, there are three possibilities for what can come after the arrow:

- a value of the type the switch expression was declared with

- throw a new exception
- a code block that evaluates to a value of the correct type

One very important thing that you must keep in mind is that since switch expressions evaluate to a specific value of a specific type, you need to account for all possible cases.

If the data type is a primitive or an object, then you must provide a default case. The only exception is using an enum because it is easier to account for every possibility.

Variations of switch expressions

You can also use colon case statements in a switch expression.
The only real difference in the code would be that a regular
old switch statement is assigned to a variable and there are no breaks.
While this is a valid option, it is not preferable because it does not take advantage of the newer and more compact arrow syntax.
It also makes it easier to lose track of whether you are looking at
a switch statement or a switch expression because other than the variable assignment at the top and the absence of breaks, there are no visual indicators of that in the code.

```
import java.util.*;
import java.lang.annotation.*;


public class MyClass {

   public static void main(String args[]) {
      String c = "a";
      int val = switch(c){
         case "a" -> 1;
         case "b" -> 2;
      };
   }
}
```

Output:

```
MyClass.java:9: error: the switch expression does not cover all possible input values
      int val = switch(c){
                ^
1 error
```

For enum it will be covered and we do not need default.

Java 13 introduced the yield keyword which can be used inside colon case statements to identify the value the case statement yields. It also replaces the break statement and removes the need to explicitly mention the variable the value is assigned to. If you are going to use colons in your case statements, using the new yield keyword is the best option.

```
int tasteValue = switch (taste) {
    case SUGAR:
    case PIZZA:
    case CHOCOLATE:
        yield 10;
    case MEATBALLS:
    case STEAK:
        yield 7;
    case BROCCOLI:
        yield 4;
    case DIRT:
        yield 1;
    default:
        throw new IllegalStateException("Invalid tastable object: " + taste);
};
```

The yield keyword cannot be used inside a switch statement. Likewise, break cannot be used in a switch expression. Therefore the use of yield helps ensure that the reader of your code doesn't forget which type of switch they are reading.

There is also an in-between option that uses the new arrow syntax but puts a code block with a yield in it to the right of the arrow. This might seem unnecessarily verbose compared to the first example of the arrow case syntax shown above, but there are situations in which this long way has some advantages. The yield statement must be the last line in the code block, but you can call other functions in the lines before it.

```
tasteValue = switch (taste) {
    case SUGAR, PIZZA, CHOCOLATE -> {
        System.out.println(10);
        yield 10;
    }
    case MEATBALLS, STEAK -> {
        System.out.println(7);
        yield 7;
    }
```

```
    case BROCCOLI -> {
        System.out.println(4);
        yield 4;
    }
    case DIRT -> {
        System.out.println(1);
        yield 1;
    }
    default -> {
        throw new IllegalStateException("Invalid tastable object: " + taste);
    }
};
```

Conclusion

A switch expression can be used instead of a switch statement to make the code more concise and less error-prone.
The entire switch expression is assigned to a variable because it yields a value.
Unless you are using an enum in your switch expression, you must include a default case. You can yield a single value, throw an exception, or use a code block that ultimately evaluates to a single value.

The new arrow syntax allows us to put all of our cases that yield the same result on one line; the arrow replaces the colon and the break.
Java 13 introduced the yield keyword, which can be used in switch expressions but not in switch statements.
It can be used both with colon or arrow syntax, but it is typically used at the end of a code block to return a value, often after calling other functions earlier in the block.
If yield is used with colon syntax, it replaces the break, same as the arrow does in arrow syntax.
An easy way to differentiate between switch expressions and switch statements is that switch expressions cannot have breakin them, and switch statements cannot have yield in them.