TheServerSide.com

# New Java 7 Features: Using String in the Switch Statement Tutorial

First, I hate the switch statement. It is so darned procedural, and quite often when I see a junior programmer fiddling around with a switch statement, I'm always convinced that there's probably a much more object oriented way of approaching the problem at hand. But love it or hate it, the switch statement is part of the Java language, and it is part of the Oracle Certified Professional exam, so you need to know how to use it, especially if you want to obtain your OCP designation.

## Java 7 and the Switch Statement

Prior to Java 7, the condition of the switch had to be either a non-long integer type (byte/Byte, short/Short, char/Character and int/Integer), or an enumerated type. So, a basic switch statement might look like this:

```
public class IntSwitch {
  public static void main(String[] args) {

    int numberOfPlayers = 6;
    String sport = null;

    switch (numberOfPlayers) {
      case 1:
        sport = "tennis";
        break;
        case 6:
        sport = "volleyball";
        break;
      case 9:
        sport = "baseball";
        break;
      default:
        sport = "redrover";
      break;
    }
      System.out.println("You're playing " + sport);
  }
}
```

The code above prints out: You're playing volleyball

It is interesting that you can't switch on a long. Attempting a piece of code such as this:

```
long sport = 0;
switch (sport) {}
```

Would generate a compile time error such as this: Cannot switch on a value of type long. Only convertible int values, strings or enum constants are permitted

## Switching on a String

New in Java 7 is the ability for your programs to switch on a String:

```
public class StringSwitch {
  public static void main(String[] args) {

    int numberOfPlayers = 0;
    String sport = "volleyball";

    switch (sport) {
      case "tennis":
        numberOfPlayers = 1;
        break;
      case "volleyball":
        numberOfPlayers = 6;
        break;
      case "baseball":
        numberOfPlayers = 9;
        break;
    }
    System.out.println(numberOfPlayers + " players are needed.");
  }
}
```

In this case, the code switches on the name of the sport, which is of type String, and initializes the int variable named numberOfPlayers depending upon which conditional case in the switch statement is met. When compiled and executed, the code above prints out: 6 players are needed.

By the way, you can group cases together to allow for an initialization to occur under multiple conditions. Take a look at the following example, where the String *sport* is initialized to "*hockey*":

```
public class StringSwitch {
  public static void main(String[] args) {

    int numberOfPlayers = 0;
    String sport = "hockey";

    switch (sport) {
      case "tennis": case "pingpong": case "badminton":
        numberOfPlayers = 1;
        break;
      case "volleyball": case "hockey":
        numberOfPlayers = 6;
        break;
      case "baseball": case "softball":
        numberOfPlayers = 9;
        break;
    }
    System.out.println(numberOfPlayers + " player(s) are needed.");
  }
}
```

In this switch statement, the numberOfPlayers variables gets initialized to the number 6, and the following is printed out to the console: 6 player(s) are needed.

## String and only Strings

And one other thing to note is that the in Java 7 you can switch on a variable of type String, but the variable must be referenced as a String, and not simply initialized as a String, so the following code would fail:

```
public class StringSwitch {
  public static void main(String[] args) {

    int numberOfPlayers = 0;
    Object sport = "hockey";

    switch (sport) {
      case "tennis": case "pingpong": case "badminton":
        numberOfPlayers = 1;
        break;
      case "volleyball": case "hockey":
        numberOfPlayers = 6;
        break;
      case "baseball": case "softball":
        numberOfPlayers = 9;
        break;
    }
    System.out.println(numberOfPlayers + " player(s) are needed.");
  }
}
```

In this case, the code fails with the following error:
**Cannot switch on a value of type Object. Only convertible int values, strings or enum constants are permitted**

## Passing null references to a switch statement

By the way, one thing you need to be careful of is passing a null to a switch statement. It wasn’t a problem when switch statements just worked with basic primitive types, but when you’re working with wrapper classes, there’s always the possibility that a null object has found it’s way into the mix.

There are two key things you need to know about nulls and switch statements. First, if you pass a null to a switch statement, you’ll get a NullPointerException at runtime. Secondly, there is no way to test  a null **case** condition in the body of the switch. Trying to add a **case null:** will generate the following compile time exception: **case expressions must be constant expressions**

So, the following code is a complete and total fail in several different ways:

```
int numberOfPlayers = 0;
String sport = null;
switch (sport) {
  case null:
    numberOfPlayers = -1;
    break;
  case "tennis": case "pingpong": case "badminton":
    numberOfPlayers = 1;
    break;
  case "volleyball": case "hockey":
    numberOfPlayers = 6;
    break;
  case "baseball": case "softball":
    numberOfPlayers = 9;
  break;
}
System.out.println(numberOfPlayers + " player(s) are needed.");
```

The code will fail to compile due to the null case, and even if that was removed, the fact that the code switches on a null String would trigger a NullPointerException at runtime. So let that be a lesson to you: avoid null values, especially when you're working with switch statements.

**Java Certification Books and Learning Resources**

OCP Java SE 6 Programmer Practice Exams (Exam 310-065) (Certification Press)
OCP Java SE 7 Programmer Study Guide (Certification Press)
SCJP Sun Certified Programmer for Java 6 Exam 310-065
A Programmer's Guide to Java SCJP Certification: A Comprehensive Primer (3rd Edition)
SCJA Sun Certified Java Associate Study Guide for Test CX-310-019, 2nd Edition

**Check out these other tutorials from TheServerSide's Sal Pece and Cameron McKenzie covering the new Java 7 features:**

New Java 7 Features: Binary Notation and Literal Variable Initialization
New Java 7 Features: Numeric Underscores with Literals Tutorial
New Java 7 Features: Using String in the Switch Statement Tutorial
New Java 7 Features: The Try-with-resources Language Enhancement Tutorial
New Java 7 Features: Automatic Resource Management (ARM) and the AutoCloseable Interfact Tutorial
New Java 7 Features: Suppressed Exceptions and Try-with-resources Tutorial
Java 7 Mock Certification Exam: A Tricky OCPJP Question about ARM and Try-with-Resources
OCAJP Exam to Debuts in March 2010. OCPJP Released in June?
OCPJP & OCAJP Java 7 Exams: Oracle Drops the Training Requirement
OCAJP and OCPJP Changes for Java 7: New Objectives, a Format Change and a Price Hike

*15 Dec 2011*