## Java 7 Features

1. **Strings in switch Statement**

* String in switch statement feature is available from JDK7
* Before JDK7 we can use only int, char and enum in switch statement
* But now with JDK7 we can directly use string inside switch and case statement
* If we want to use string before JDK7, we need to map those string to final integer constant or char constant to use them inside the switch and case statement
* We can avoid the number of if-else statement using this feature

**Example of String in switch in JDK7:**

public String getTypeOfDayWithSwitchStatement(String dayOfWeekArg) {

String typeOfDay;

switch (dayOfWeekArg) {

case "Monday":

typeOfDay = "Start of work week";

break;

case "Tuesday":

case "Wednesday":

case "Thursday":

typeOfDay = "Midweek";

break;

case "Friday":

typeOfDay = "End of work week";

break;

case "Saturday":

case "Sunday":

typeOfDay = "Weekend";

break;

default: throw new IllegalArgumentException();

}

return typeOfDay;

}

**Example of String in switch prior JDK7:**

public String getTypeOfDayWithSwitchStatement(String dayOfWeekArg) {

String typeOfDay;

if (dayOfWeekArg.equals("Monday":) {

typeOfDay = "Start of work week";

} else if (dayOfWeekArg.equals("Thursday ":) {

typeOfDay = " Midweek ";

} else if(dayOfWeekArg.equals("Friday ":) {

typeOfDay = " End of work week ";

} else if(dayOfWeekArg.equals("Sunday ":) {

typeOfDay = " Weekend ";

} else {

throw new IllegalArgumentException();

}

return typeofDay;

}

**2. Multiple Exception Handling**

* Multiple Exception Handling feature is available from JDK7
* A single catch block can handle more than one type of exception
* The catch clause specifies the types of exceptions that the block can handle, and each exception type is separated with a vertical bar (|)
* This feature can reduce code duplication
* The below example contains duplicate code in each of the catch blocks

catch (IOException ex) {

logger.log(ex);

throw ex;

} catch (SQLException ex) {

logger.log(ex);

throw ex;

}

* The following example, which is valid in JDK7 and later, eliminates the duplicated code

catch (IOException|SQLException ex) {

logger.log(ex);

throw ex;

}

* If a catch block handles more than one exception type, then the catch parameter is implicitly final.
* In the above example, the catch parameter ex is final and therefore you cannot assign any values to it within the catch block.

**3. Try with Resources**

* Try with Resources feature is available from JDK7
* The try-with-resources statement is a try statement that declares one or more resources
* A resource is an object that must be closed after the program is finished with it
* Any object that implements java.lang.AutoCloseable can be used as resource
* The following example reads the first line from a file. BufferedReader is a resource that must be closed after the program is finished with it

static String readFirstLineFromFile(String path) throws IOException {

try (BufferedReader br = new BufferedReader(new FileReader(path))) {

return br.readLine();

}

}

* The resources declared in a try-with-resource statement will be closed regardless of whether the try statement completes normally or abruptly
* Prior to JDK7, we can use a finally block to ensure that a resource is closed regardless of whether the try statement completes normally or abruptly

static String readFirstLineFromFileWithFinallyBlock(String path)

throws IOException {

BufferedReader br = new BufferedReader(new FileReader(path));

try {

return br.readLine();

} finally {

if (br != null) br.close();

}

}

* The exception thrown from the try-with-resources block is suppressed
* The try-with-resources statement can contains more than one declarations that are separated by a semicolon

try (ZipFile zf = new ZipFile(zipFileName);

BufferedWriter writer = new BufferedWriter(outputFilePath, charset)

) {

* The close methods of resources are called in the opposite order of their creation. In the above example first it will close the BufferedWriter then ZipFile resource

**4. Underscores in Numeric Literals**

* From JDK7 we can use underscore in a numerical literal.
* Underscore in a numerical literal will improve the readability of the code
* You can place underscores only between digits
* You cannot place underscores in the following places
* At the beginning or end of a number
* Adjacent to a decimal point in a floating point literal
* Prior to an F or L suffix
* In positions where a string of digits is expected
* The following examples demonstrate valid and invalid underscore placements in numeric literals:

float pi1 = 3\_.1415F; // Invalid; cannot put underscores adjacent to a decimal point

float pi2 = 3.\_1415F; // Invalid; cannot put underscores adjacent to a decimal point

long socialSecurityNumber1 = 999\_99\_9999\_L; // Invalid; cannot put underscores prior to an L suffix

int x1 = \_52; // This is an identifier, not a numeric literal

int x2 = 5\_2; // OK (decimal literal)

int x3 = 52\_; // Invalid; cannot put underscores at the end of a literal

int x4 = 5\_\_\_\_\_\_\_2; // OK (decimal literal)

int x5 = 0\_x52; // Invalid; cannot put underscores in the 0x radix prefix

int x6 = 0x\_52; // Invalid; cannot put underscores at the beginning of a number

int x7 = 0x5\_2; // OK (hexadecimal literal)

int x8 = 0x52\_; // Invalid; cannot put underscores at the end of a number

int x9 = 0\_52; // OK (octal literal)

int x10 = 05\_2; // OK (octal literal)

int x11 = 052\_; // Invalid; cannot put underscores at the end of a number

* The following example shows other ways you can use the underscore in numeric literals:

**long creditCardNumber = 1234\_5678\_9012\_3456L;**

**long socialSecurityNumber = 999\_99\_9999L;**

**float pi = 3.14\_15F;**

**long hexBytes = 0xFF\_EC\_DE\_5E;**

**long hexWords = 0xCAFE\_BABE;**

**long maxLong = 0x7fff\_ffff\_ffff\_ffffL;**

**byte nybbles = 0b0010\_0101;**

**long bytes = 0b11010010\_01101001\_10010100\_10010010;**

**5. Type Inference for Generic Instance Creation**

* We can replace the type arguments required to invoke the constructor of a generic class with an empty set of type parameters (<>)
* This pair of angle brackets is informally called the diamond
* For example, consider the following variable declaration

Map<String, List<String>> myMap = new HashMap<String, List<String>>();

* In Java SE 7, we can substitute the parameterized type of the constructor with an empty set of type parameters (<>)

Map<String, List<String>> myMap = new HashMap<>();

**6. Binary Literals**

* From JDK7, the integral types (byte, short, int, and long) can also be expressed using the binary number system.
* To specify a binary literal, add the prefix 0b or 0B to the number.
* The following examples show binary literals:

// An 8-bit 'byte' value:

byte aByte = (byte)0b00100001;

// A 16-bit 'short' value:

short aShort = (short)0b1010000101000101;

// Some 32-bit 'int' values:

int anInt1 = 0b10100001010001011010000101000101;

int anInt2 = 0b101;

int anInt3 = 0B101; // The B can be upper or lower case.

// A 64-bit 'long' value. Note the "L" suffix:

long aLong = 0b1010000101000101101000010100010110100001010001011010000101000101L;