Object Oriented Programming with Java (Subject Code: BCS-403)

Unit 3

Lecture 26

Lecture 26

- Local Variable Type Inference
- Switch Expressions
- Yield Keyword

Local Variable Type Inference

- Local variable type inference is a feature in Java 10 that allows the developer to skip the type declaration associated with local variables (those defined inside method definitions, initialization blocks, for-loops, and other blocks like if-else), and the type is inferred by the JDK.
- It will, then, be the job of the compiler to figure out the datatype of the variable.

// declaration using LVTI

```
// Java code for local variable
import java.util.ArrayList;
import java.util.List;
class A {
    public static void main(String ap[])
    {
        var data = new ArrayList<>();
    }
}
```

Cases where you can declare variables using LVTI

```
// block using LVTI in Java 10
class A {
    static
    {
        var x = "Hi there";
        System.out.println(x)'
    }
    public static void main(String[] ax)
    {
    }
}
```

As iteration variable in enhanced for-loop

```
public class MyMain {
public static void main(String[] args) {
int[] arr ={ 1, 2, 3 };
for (var x : arr)
System.out.println(x);
}
}
```

As looping index in for-loop

```
public class MyMain {
public static void main(String[] args)
{
int[] arr = { 1, 2, 3 };
for (var x = 0; x < 3; x++)
System.out.println(arr[x]);
}
}
```

As a return value from another method

```
public class MyMain {
int ret()
{
return 1;
}
public static void main(String a[])
{
var x = new MyMain().ret();
System.out.println(x);
}
}
```

As a return value in a method

```
public class MyMain {
int ret()
{
var x = 1;
return x;
}
public static void main(String a[])
{
System.out.println(new MyMain().ret());
}
}
```

There are cases where declaration of local variables using the keyword ‘var’ produces an error.

1. **Not permitted in class fields**

```
class A {
    var x;
}
```

2. **Not permitted for uninitialized local variables**
3. **Not allowed as parameter for any methods**
4. **Not permitted in method return type.**

```
public var show()
```

5. **Not permitted with variable initialized with ‘NULL’**

Switch Expressions

- Until Java 7 only integers could be used in switch case and this had been the standard for a long time.
- In Java 8 strings & enum were introduced in case values and switch statements started to evolve.

```
public class MyMain {
public static void main(String a[])
{
String day = "Tuesday";
switch (day) {
case "Monday":
System.out.println("Week day");
break;
case "Tuesday":
.
.
break;
case "Friday":
System.out.println("Week day");
break;
case "Saturday":
System.out.println("Weekend");
break;
case "Sunday":
System.out.println("Weekend");
break;
default:
System.out.println("Unknown");
}}}
```

```
public class MyMain {
public static void main(String a[])
{
enum DAYS {
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}
DAYS days = DAYS.MONDAY;
switch (days) {
case MONDAY:
System.out.println("Weekdays");
..
break;
case FRIDAY:
System.out.println("Weekdays");
break;
case SATURDAY:
System.out.println("Weekends");
break;
case SUNDAY:
System.out.println("Weekends");
break;
default:
System.out.println("Unknown");
}}}
```

Java 12 : Switch Statement

- Java 12 further enhanced the switch statement and introduced switch expressions as a *preview* feature.

It introduced a flurry of new features:

- We can return values from a switch block and hence switch statements became ***switch expressions***
- We can have multiple values in a case label
- We can return value from a switch expression through the arrow operator or through the “break” keyword

switch expressions Example

```
public class MyMain {
public static void main(String[] args) {
String day = "Wednesday";
String category = getDayCategory(day);
System.out.println(day + " is a " + category + ".");
}
public static String getDayCategory(String day) {
return switch (day) {
case "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday" -> "Weekday";
case "Saturday", "Sunday" -> "Weekend";
default -> "Unknown";
};
}}
```

Output : Wednesday is a Weekday.

Return value through break keyword

```
return  switch (day) {
    case "Monday":
        break "Weekday";
    case "Tuesday":
        break "Weekday";
     case "Friday":
        break "Weekday";
    case "Saturday":
        break "Weekend";
    case "Sunday":
        break "Weekend";
    default:
        break "Unknown";
};
```

The word break was replaced by “yield” later in Java 13.

```
return  switch (day) {
    case "Monday":
        yield  "Weekday";
    case "Tuesday":
        yield "Weekday";
    case "Sunday":
        yield "Weekend";
    default:
        yield "Unknown";
};
```

Return value through arrow operator :

```
return  switch (day)
{   case "Monday"-> "Week day";
    case "Tuesday"-> "Week day";
    case "Wednesday"->"Week day";
    case "Thursday"->"Week day";
    case "Friday"->"Week day";
    case "Saturday"-> "Weekend";
    case "Sunday"-> "Weekend";
    default->"Unknown";
};
```

Multiple case labels :

```
return  switch (day) {
            case
"Monday","Tuesday","Wednesday","Thursday","Friday"
-> "Week day";
            case "Saturday", "Sunday" -> "Weekend";
            default->"Unknown";
        };
```

Java 17 : Switch Statement / Expression :

Java 17 LTS is the latest long-term support release for the Java SE platform and it was released on September 15, 2021.

Switch expression features

- Pattern matching
- Gaurded pattern
- Null cases

Pattern Matching :

We can pass objects in switch condition and this object can be checked for different types in switch case labels.

```
return switch (obj) {
    case Integer i -> "It is an integer";
    case String s -> "It is a string";
    case Employee s -> "It is a Employee";
     default -> "It is none of the known data types";
};
```

Gaurded Patterns :

```
case Employee emp:
if(emp.getDept().equals("IT")) {
yield "This is IT Employee";
}

return switch (obj) {
case Integer i -> "It is an integer";
case String s -> "It is a string";
case Employee employee &&
employee.getDept().equals("IT") -> "IT Employee";
default -> "It is none of the known data types";
};
```

Null Cases

We can never pass a null value to switch statements prior to Java 17 without a Null pointer exception being thrown.

Java 17 allows you to handle it this way

```
case null -> "It is a null object";
```