Lecture 18

Switch Statement

- Switch statement is an alternative to an if-else-if ladder statement.
- We use switch statement when we want to execute one of many code blocks based on a condition but this condition should be simple. e.g:- if the condition involes a single variable and equality operator then we can use switch instead of if statements.

it is also known as Selector expression

Syntax:-

```
switch (expression) {
    case value1:
        // Code will be executed if expression matches value1
        break;
    case value2:
        // Code will be executed if expression matches value2
        break;
    .
    .
    .
    case valuen:
        // Code will be executed if expression matches valuen.
        break;
    default:
        // Code to be executed if no case value matches the
        // expression.
        break;
}
```

Working of Switch:-

① The switch expression is evaluated once. & this expression value should be of compatible type (like byte, short, int, char, enum, , String). [String was allowed in Java Switch from Java 7]
& Wrapper class also allowed (like Byte, Short, Integer, Character)
→ Expression can also have any object reference – Pattern matching for switch (Java 21)

② Here case is a keyword and the value1 is known as Case labels. Each case represents a possible value of expression. ~~The value of expression is compared with the value of each~~
. Code block of the matched case would be executed.

③ break :- break stops further execution & exits the switch block. Without break the next case will also execute. (this is known as fall-through)

④ default :- The block is executed if no case matches. It is optional but useful.
→ It is not compulsory to write default block at the end. We can write it anywhere with the switch block.

→ **fall-through in switch** :-
. Any code after the matched case, will be executed, until a break statement or the end of switch statement occurs.
Because of this Java Switch Statement is known a fall through statement.

Switch expression can not be null (till Java 20)

IMP Points about Switch :-

①
```
int x=1, b=2;
Switch (x+b){
```
Output = 3
```
    Case 3 :
        System.out.println ("x+b = ", +(x+b));
        break;
    .
    .
}
```

(x+b) will be evaluated to int value only. So it is allowed in switch expression. but float, double, long & boolean types are not allowed in expression.

② Case labels must be constant expression in context of switch.

eg:- in above example if I write :-

(i) int temp = 3;
Case temp : → error variables are not allowed because variables will be evaluated at run time so the value of variable will not be known at compile time unless it is a constant variable
System.out.println ("x+b = ", +(x+b));

But :- (ii) final int temp = 3; → No error because temp is a constant here
Case temp :
System.out.println("x+b = ", +(x+b));

If we write
final int temp;
temp = 3;
it will give error.

(iii) Case 1+2 : → allowed it is a constant expression

(iv) Case "1+2" : → it is string as it is in " "(double quotes)
String type is allowed in case labels.
but in this example in switch expression we are writing switch (x+b) which is int expression. So in this case it will give error.

NOTE:- Switch expression type & case label type must be same.

So we can say case labels/values must be constants or literals and of the same type as the switch expression.

(3) Duplicate case labels/values are not allowed

(4) Sometimes if you need to have multiple cases without "break" then it is allowed.

e.g:-

```
byte day = 2;
switch (day) {
    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:
        System.out.println("Invalid day type");
}
```

imp — Case labels should be in the range of Data type (byte)

(5) So in many cases when we have to choose from different choices based on some condition (simple condition having only one variable & equality operator) then we use switch because it is more readable, concise & it can also be efficient than if. statements. Because switch is designed to be efficient. The code becomes much more readable, compact & concise if we use switch. And if we use arrow labels & switch expression then code would be much more compact & concise.

(6) After break we can't write any other statement within the same case.

→ Switch is a fundamental control flow statement. So it's been there from Java 1 & the syntax has been constant from Java 1 & till Java 13. But there are some limitations in this basic syntax (fall through). So the language designers have extended the syntax to include

Arrow label & switch expressions

↳ these features were added in Java 12 as a preview feature & became a standard feature in Java 14.

Why switch is more efficient?

↳ It's because of that fact that case values must be constants expression meaning the case values must be known at compile time only. Hence the case values can be found in constant time O(1).

e.g:-

```
int day = 7;
switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        System.out.println("Week day"); break;
    case 6: case 7:
        System.out.println("Weekend"); break;
}
```

→ In the case control will directly jump to the case label 7. (not guaranteed). So we are not going to match day with 1, 2, 3, 4, 5, 6, 8. So this is constant time **O(1)**. But if we use if then all the if, else if, conditions would be checked until the true case/condition is found. **O(n)** [in the worst case if we enter day = 8 then in if-else all the conditions would be checked & then the else block would be executed having time complexity O(n).

? How enum types can be used in Switch expression ?

```
enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY };

Day day = Day.FRIDAY;
switch (day) {
    case MONDAY: case TUESDAY: case WEDNESDAY:
    case THURSDAY: case FRIDAY:
        System.out.println("Weekday");
    case SATURDAY: case SUNDAY:
        System.out.println("Weekend");
    default:
        System.out.println("Invalid day choice");
}
```

Here, enum in Java is a special class that represents a group of predefined constants. So in this example we have 7 enum constants.

→ So here a class called Day would be created and it has 7 different instances / variables of type Day.