Switch Case with Grouping in Java :

In switch statement, It is possible to combine multiple case labels to execute one common block of code. It must be separated by , (comma)

We should use switch case with grouping style, when multiple values (cases) should result in the same action.

```
//Program
Group.java
--------------
void main()
{
 int data = Integer.parseInt(IO.readln("Enter a number :"));

   switch(data)
   {
        case 2, 4, 6, 8 -> IO.println("It is Even number");

        case 1, 3, 5, 7 -> IO.println("It is odd number");
   }
}
```

```
Group1.java
--------------
      void main()
      {
           IO.println("Enter an animal Name :");
           String animal = IO.readln().toLowerCase();

           switch(animal)
           {
                case "dog", "cat", "cow", "sheep" ->
                {
                  IO.println("It is a domestic animal");
                }

                case "lion", "tiger", "elephant", "wolf" ->
                {
                    IO.println("It is a wild animal");
                }

                default ->
                {
                    IO.println("Unknown animal");
                }
           }
      }
```

Switch Case with Return Value (Expression Style) :

* From switch body we can return some value

* When we want to return some value from switch body then ; is compulsory
  example :
```
String str = switch(var)
{
   case 101 -> "xyz";
};
```

* default is compulsory becauae If no cases will match then this default case will return the value.

* Working with switch as an expression, we must return appropriate type of return type. For a single statement where {} braces are not required the statement will itself return a value but If have multiple statements then {} are compulsory and to return the value yield keyword is required.

From JDK 14V onwards, A switch statement can also return some value that means we can use switch statement as switch expression.

This is known as switch case with return value OR expression style switch.

Rules :

a) While writing switch body ; (semicolon is compulsory at the end)
b) Here default -> is compulsory
c) From the switch body, We need to return appropriate value.

```
Expr.java
-----------
//Print the message based om the grade
void main()
{
          IO.println("Enter your Subject grade [A/B/C/D/E]");
          char grade = IO.readln().toUpperCase().charAt(0);

          String result = switch(grade)
         {
               case 'A', 'B' -> "Excellent!!";
               case 'C' -> "Very Good!!";
               case 'D' -> "Good";
               case 'E' -> "Average";

         default -> "No Grade provided";
          };

          IO.println("Student Grade is :"+result);
}
```

From JDK 14v onwards, switch can be directly used as an expression so It can directly return value hence return keyword is not required.

On the other hand, If we use switch expression with {} blocks then compulsory we should use yield (keyword) not return statement as shown in the program below :

```
/* Employee performer of the year
   Grade A -> High-level performer [bonus amount is 20% of the salary]
   Grade B -> Mid-level performer [bonus amount is 15% of the salary]
   Grade C -> Basic performer [bonus amount is 10% of the salary]
   No grade -> 5 % of the salary
*/

void main()
{
       IO.print("Enter your performance grade [A/B/C] : ");
       char grade = IO.readln().toUpperCase().charAt(0);

       double salary = 40000;

      double bonus = switch(grade)
       {
            case 'A' ->
        {
                IO.println("High-level performer");
                yield salary*0.20;
            }
        case 'B' ->
        {
                IO.println("Mid-level performer");
                yield salary*0.15;
            }
        case 'C' ->
        {
                IO.println("Basic performer");
                yield salary*0.10;
            }

        default -> salary * 0.05;
       };

       IO.println("Your Salary is :"+salary);
       IO.println("Your bonus amount is :"+bonus);

}
```

Note :If we have block of statements while working with switch cases then we should compulsory use yield keyword.

return keyword is used to return a value and terminate the flow from a method but switch is not a method (does not contain return type) It is an expression so, yield keyword is compulsory.

```
void main()
{
     long x = 12;

        switch(x)
        {
         case 12L:
        }
}
```

Note : We can't pass long, float and double and boolean value.

```
void main()
{
     byte b = 90;

    switch(b)
     {
         case 90 -> IO.println("Case 90");
         case 128 -> IO.println("Case 128");
     }

}
```

While working with switch case byte and short case value must be within the range only.

---

instanceof operator :

* It is a relational operator
* It returns true OR false
* It is mainly used to verify whether a reference variable is pointing to a particular type of Object or not

```
instanceof.java
-----------------
void main()
{
   String s = "Java";  //s is pointing to String object

   if(s instanceof String)
     {
       IO.println("s is pointing to String object");
     }
}
```

Bitwise left & right shit operator (<< left shift , >> right shift)

It will work with two operands
It is used to work at the binary (bit) level, not on decimal values.
It is used to shift the bits of an integer value to the left (<<) or right (>>).
They are mainly used for fast arithmetic operations at bit level.

Left shift Operator formula: x<<n = x * 2^n
Right shift Operator formula: x>>n = x / 2^n

```
void main()
{
  IO.println(9<<1);   //9 X 2
  IO.println(9<<2);   //9 X 4
  IO.println(9<<3);   //9 X 8

  IO.println("..................");
  IO.println(9>>1);  //9 / 2
  IO.println(9>>2);  //9 / 4
}
```

**What is Method Overloading in Java :

* We cannot write multiple methods with same name.
  Example :
```
void main()
{
   IO.println("MOL");
}

void main()
{
}
```

* We can write multiple methdos with same name but parameter must be different.
```
void main()
{
   IO.println("MOL");
}

void main(int x)
{
}
```

Priority of main method given by java from Java 25V :

Case 1 :
```
void main()   //Java 25V main method without command line argument program
{
}
```

Case 2 :
```
void main(String [] args)   //Java 25V main method with command line argument program
{
}
```

Case 3 :
```
public static void main(String [] args) //Old method
{

}
```

Points to remember :

1) In Java 25v, both the main methods are non static method [JVM is creating the object for calling these methods]

2) Our old main method is static method [JVM is calling with class name, To call this main method object is not required.]

3) Among all these 3 main method the priority is given by using the following order :

```
  a) public static void main(String [] args)   |
      {                                         |
      }                                         |   Priority Order
                                                |   Highest
  b) void main(String [] args)                  |      ⇩
      {                                         |   Lowest
      }                                         |
                                                |
  c) void main()                                |
      {                                         ↓
      }
```

```
Programs :
-------------
void main()
{
    IO.println("Lowest Priority");
}

public static void main(String [] args)
{
    IO.println("Highest Priority");
}
```

Note : Old main method is having highest priority

```
//Program
  void main()
  {
      IO.println("Lowest Priority");
  }

  void main(String [] args)
  {
      IO.println("Highest Priority");
  }
```

Note : The main method which is taking String as a parameter has higher priority than main() method

```
//Program
-------------
public static void main(String [] args)
{
    IO.println("Highest Priority");
}

void main(String [] args)
{
    IO.println("Highest Priority");
}
```

Note : error

```
//Program

public static void main(String x)
{
    IO.println("Highest Priority");
}

void main(int x)
{
    IO.println("Lowest Priority");
}
```

Error : Compilation error , compiler is not finding either main(){} OR main(String []args) in the compact source file