**Flow Control**

**Selection Statement / Decision Making statements**

* if else
* switch

# **Iterative Statement / Loop Statement**

* while
* do while
* for
* for each

# **Transfer Statement**

* break
* continue
* return
* try catch finally
* assert

# if statement

1. **Simple if statement**

if (booleanExpression) {

System.out.println("Inside if statement");

}

1. **if-else statement**

if(condition) {

statement 1; //executes when condition is true

}

else{

statement 2; //executes when condition is false

}

1. **if-else-if ladder**

if(condition 1) {

statement 1; //executes when condition 1 is true

}

if(condition) {

statement 1; //executes when condition is true

}

else{

statement 2; //executes when condition is false

}

1. **Nested if-statement**

if(condition 1) {

statement 1; //executes when condition 1 is true

}

else if(condition 2) {

statement 2; //executes when condition 2 is true

} else{

statement 2; //executes when condition 2 is false

}

|  |  |
| --- | --- |
| int value =10;  if(value){  System.out.println(“ten”);  } | Not allowed, compilation error  Value is not Boolean expression |
| int value =10;  if(value=10){  System.out.println(“ten”);  } | Not allowed, compilation error  value=10 is an assignment, is not a Boolean expression. |
| int value =10;  if(value==10){  System.out.println(“ten”);  } | Allowed  value==10 is a boolean expression. |
| if(true){  System.out.println(“true”);  } | valid |
| if(false){  System.out.println(“false”);  } | Valid but block will not be executed |
| Curly braces are optional but only one statement is allowed to execute.  if(condition)  statement-1  statement-2 | int i=5;  if(i==5)  System.out.println(“yes it is 5”);  System.out.println(“no it is not 5”); |
| You can have zero or one else for a given if, and it must come after any else ifs.. | else{  } |
| You can have zero to many else ifs for a given if and they must come before the (optional) else. | if(condition-1){}  else if(condition-2){}  else if(condition-3){}  else{} |
| Once an else if succeeds, none of the remaining else ifs or elses will be tested. | if(condition-1){}  else if(condition-2){}  else if(condition-3){}  else{}  once condition-1 founds true then remaining condition will not be checked.  int value=10;  if(value==10){  System.out.println("ten");  }else if(value==10){  System.out.println("ten");  }else if(value==10){  System.out.println("ten");  } |
| int i=10;  if(i==10){  System.out.println("");  }  System.out.println("");  else{  System.out.println("");  } | Else without if.  No statement is allowed between if and else statement. |
| int value=10;  if (value < 1)  if (value< 5)  System.out.println("Try again.");  else  System.out.println("Java master!"); | which if belongs to else?  If there are not curly braces then else block will be match to just prior if statement. |
| You can write multiple conditions in if statement. The result should be Boolean value.  if (condition -1 && condition-2){  }  If( condition-1 || condition-1){  } | int y = 5;  int x = 2;  boolean doStuff=true;  if (((x > 3) && (y < 2)) || doStuff) {  System.out.println("true");  } |
| boolean boo = false;  if (boo = true) {  //statement will be exucted and boo will have true value.  }  Sout.(check the boo value here. | This is a valid syntax.  The code compiles and runs fine and the if test succeeds because boo is SET to true (rather than TESTED for true) in the if argument!  Task: try for boo=false condition. |

# switch Statement

switch (expression) {

case constant1: code block

case constant2: code block

default: code block

}

case, break, default is optional.

A switch's expression must evaluate to a char, byte, short, int, enum

Wrapper classes are allowed like Character, Byte, Short, Integer

enum type was introduced in Java 5 and has been available in the switch statement since then.

String type is available in the switch statement starting with Java 7.

A case constant must evaluate to the same type as the switch expression can use,

switch argument and case values should be of the same type.

|  |  |
| --- | --- |
| Below case is in if else block can be written in switch case  int x = 3;  if(x == 1) {  System.out.println("x equals 1");  }  else if(x == 2) {  System.out.println("x equals 2");  }  else if(x == 3) {  System.out.println("x equals 3");  }  else {  System.out.println("No idea what x is");  } | Switch case example:  int x = 3;  switch (x) {  case 1:  System.out.println("x is equal to 1");  break;  case 2:  System.out.println("x is equal to 2");  break;  case 3:  System.out.println("x is equal to 3");  break;  default:  System.out.println("Still no idea what x is");  } |
| the case constant must be a **compile time constant**! | int value =2;    final int firstValue=1;  final int secondValue=2;    switch(value){  case firstValue: System.out.println("one");  break;  case secondValue: System.out.println("two");  break;  } |
| byte g = 2;  switch(g)  {  case 23:  case 128:  } | Won’t compile, byte size is -128 to +127 only. |
| It's also illegal to have more than one case label using the same value. For example, the following block of code won't compile because it uses two cases with the same value of 80: | int temp = 90;  switch(temp)  { case **80** : System.out.println("80");  case **80** : System.out.println("80"); // won't compile!  case 90 : System.out.println("90"); default : System.out.println("default");  } |
| It is legal to leverage the power of boxing in a switch expression. For instance, the following is legal:  Wrapper classes are allowed.  Integer, Byte, Short, Character. | switch(new Integer(4))  { case 4: System.out.println("boxing is OK"); } |
| Break and **Fall-Through** in switch Blocks:  case constants are evaluated from the top down, and the first case constant that matches the switch's expression is the execution entry point. | int x = 1;  switch(x) { case 1: System.out.println("x is one");  case 2: System.out.println("x is two");  case 3: System.out.println("x is three"); } |
| Curly braces are allowed. | int x = 1; switch(x) { case 1: { System.out.println("x is one"); break; } case 2: { System.out.println("x is two"); break; } case 3: { System.out.println("x is two"); break; } } System.out.println("out of the switch"); |
| // fall through when you use fall-through logic | int x = someNumberBetweenOneAndTen;  switch (x) {  case 2:  case 4:  case 6:  case 8:  case 10: { System.out.println("x is an even number");  break; } } |
| The Default Case | int x = someNumberBetweenOneAndTen;  switch (x)  { case 2:  case 4:  case 6:  case 8:  case 10: { System.out.println("x is an even number"); break; }  default: System.out.println("x is an odd number");  } |
| The default case doesn’t have to come at the end of the switch. Look for it in strange places such as the following | int x = 2; switch (x) {  case 2: System.out.println("2"); default:System.out.println("default"); case 3: System.out.println("3");  case 4: System.out.println("4");  } |
| Switch case with String. Value must match with case statement. | String day = "Tuesday"; switch (day) { case "Monday": System.out.println("Week day"); break; case "Tuesday": System.out.println("Week day"); break; case "Wednesday": System.out.println("Week day"); break; case "Thursday": System.out.println("Week day"); 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"); } |
| *Enum Switch Case* | enum DAYS {  *MONDAY*, *TUESDAY*, *WEDNESDAY*, *THURSDAY*, *FRIDAY*, *SATURDAY*, *SUNDAY* }DAYS days = DAYS.*SUNDAY*; switch (days) {  case *MONDAY*:  System.*out*.println("Weekdays");  break;  case *TUESDAY*:  System.*out*.println("Weekdays");  break;  case *WEDNESDAY*:  System.*out*.println("Weekdays");  break;  case *THURSDAY*:  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 14+ introduced arrow operators. You are allowed to return switch expression. See the example. | public class IfTest {  public static void main(String[] args) {   System.*out*.println(*test*("Tuesday"));  }    public static String test(String day){  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";  };  } }  public class IfTest {  public static void main(String[] args) {   System.*out*.println(*test*("Tuesday"));  }    public static String test(String day){  return switch (day) {  case "Monday","Tuesday","Wednesday","Thursday","Friday"  -> "Week day";  case "Saturday", "Sunday" -> "Weekend";  default->"Unknown";  };   } } |
| The *yield*Keyword | public class IfTest {  enum Month {*JANUARY*,*JUNE*,*FEBRUARY*,*SEPTEMBER*,*MARCH*,*APRIL*,*MAY*, *DECEMBER*};   public static void main(String[] args) {  System.*out*.println(*test*(Month.*JUNE*));  }    public static int test(Month month){  return switch (month) {  case *JANUARY*, *JUNE* -> 3;  case *FEBRUARY*, *SEPTEMBER*-> 1;  case *MARCH*, *MAY*, *APRIL* -> {  int monthLength = month.toString().length();  yield monthLength \* 4;  }  default -> 0;  };  } } |

## **Java 17 : Switch Statement / Expression :**

NOTE: below are preview feature, not enabled by default.

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

1. Pattern matching
2. Gaurded pattern
3. Null cases

# Pattern Matching :

It has introduced a new feature for switch i.e called *pattern matching.*

You can match patterns in a case label.

In other words you can pass objects in switch condition and this object can be checked for different types in switch case labels.

public class IfTest {  
 public static void main(String[] args) {  
  
 System.*out*.println(*test*(new String("Java")));  
 }  
  
  
 public static String test(Object obj){  
 return switch (obj) {  
 case Integer i -> "It is an integer";  
 case String s -> "It is a string";  
 case Employee e -> "It is a Employee";  
 default -> "It is none of the known data types";  
 };  
 }  
}  
  
class Employee{}

In the java: patterns in switch statements are a **preview feature** and are disabled by default. (use --enable-preview to enable patterns in switch statements)

above example I am passing an object to the switch condition. This was not possible until Java 17. And then this object can be checked for a particular data type and assigned to a variable as well.

For example consider the case :

case Integer i- > "It is an integer";

The passed object is checked for the type “Integer” and then assigned to the variable “i” if it is an integer. And through the arrow operator the string “It is an integer ” is returned .

# **Gaurded Patterns :**

Let’s take this use case.

Inside the case label where I have checked for a “Employee” instance , I want to do an additional check.

Thinking traditionally , you could be doing this after the case statement.

Something like this:

case Employee emp:

if(emp.getDept().equals("IT")) {

yield "This is IT Employee";

}

But Java 17 has introduced “Guarded Patterns” . You can do this check in the case label itself like below

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 :**

You could 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";

If you have the above switch expression you will never get Null Pointer exception if the object you pass is null.