public class IfStatementExample {

public static void main(String args[]){

int num=70;

if( num < 100 ){

/\* This println statement will only execute,

\* if the above condition is true

\*/

System.out.println("number is less than 100");

}

}

}

**Output:**

number is less than 100

**Nested if statement in Java**

When there is an if statement inside another if statement then it is called the **nested if statement**.  
The structure of nested if looks like this:

if(condition\_1) {

Statement1(s);

if(condition\_2) {

Statement2(s);

}

}

Statement1 would execute if the condition\_1 is true. Statement2 would only execute if both the conditions( condition\_1 and condition\_2) are true.

**Example of Nested if statement**

public class NestedIfExample {

public static void main(String args[]){

int num=70;

if( num < 100 ){

System.out.println("number is less than 100");

if(num > 50){

System.out.println("number is greater than 50");

}

}

}

}

**Output:**

number is less than 100

number is greater than 50

**If else statement in Java**

This is how an if-else statement looks:

if(condition) {

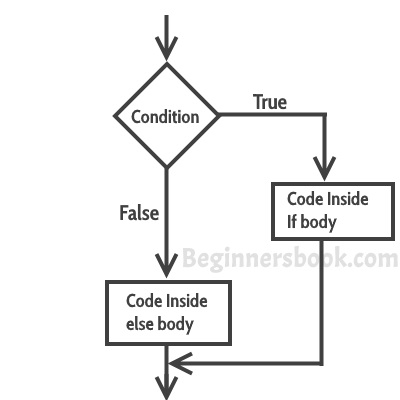
Statement(s);

}

else {

Statement(s);

}

The statements inside “if” would execute if the condition is true, and the statements inside “else” would execute if the condition is false.  


**Example of if-else statement**

public class IfElseExample {

public static void main(String args[]){

int num=120;

if( num < 50 ){

System.out.println("num is less than 50");

}

else {

System.out.println("num is greater than or equal 50");

}

}

}

**Output:**

num is greater than or equal 50

**if-else-if Statement**

if-else-if statement is used when we need to check multiple conditions. In this statement we have only one “if” and one “else”, however we can have multiple “else if”. It is also known as **if else if ladder**. This is how it looks:

if(condition\_1) {

/\*if condition\_1 is true execute this\*/

statement(s);

}

else if(condition\_2) {

/\* execute this if condition\_1 is not met and

\* condition\_2 is met

\*/

statement(s);

}

else if(condition\_3) {

/\* execute this if condition\_1 & condition\_2 are

\* not met and condition\_3 is met

\*/

statement(s);

}

.

.

.

else {

/\* if none of the condition is true

\* then these statements gets executed

\*/

statement(s);

}

**Note:** The most important point to note here is that in if-else-if statement, as soon as the condition is met, the corresponding set of statements get executed, rest gets ignored. If none of the condition is met then the statements inside “else” gets executed.

**Example of if-else-if**

public class IfElseIfExample {

public static void main(String args[]){

int num=1234;

if(num <100 && num>=1) {

System.out.println("Its a two digit number");

}

else if(num <1000 && num>=100) {

System.out.println("Its a three digit number");

}

else if(num <10000 && num>=1000) {

System.out.println("Its a four digit number");

}

else if(num <100000 && num>=10000) {

System.out.println("Its a five digit number");

}

else {

System.out.println("number is not between 1 & 99999");

}

}

}

**Output:**

Its a four digit number

public class SwitchCaseExample1 {

public static void main(String args[]){

int num=2;

switch(num+2)

{

case 1:

System.out.println("Case1: Value is: "+num);

case 2:

System.out.println("Case2: Value is: "+num);

case 3:

System.out.println("Case3: Value is: "+num);

default:

System.out.println("Default: Value is: "+num);

}

}

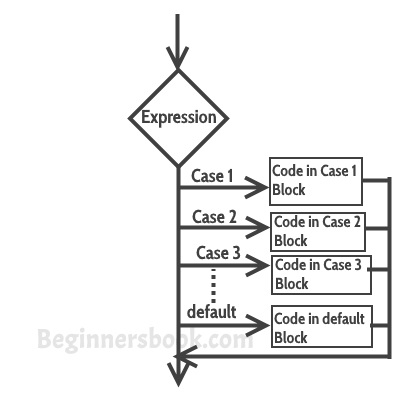
}

**Output:**

Default: Value is: 2

**Explanation:** In switch I gave an expression, you can give variable also. I gave num+2, where num value is 2 and after addition the expression resulted 4. Since there is no case defined with value 4 the default case got executed. This is why we should use default in switch case, so that if there is no catch that matches the condition, the default block gets executed.

## Switch Case Flow Diagram

First the variable, value or expression which is provided in the switch parenthesis is evaluated and then based on the result, the corresponding case block is executed that matches the result.  


## Break statement in Switch Case

Break statement is optional in switch case but you would use it almost every time you deal with switch case. Before we discuss about break statement, Let’s have a look at the example below where I am not using the break statement:

public class SwitchCaseExample2 {

public static void main(String args[]){

int i=2;

switch(i)

{

case 1:

System.out.println("Case1 ");

case 2:

System.out.println("Case2 ");

case 3:

System.out.println("Case3 ");

case 4:

System.out.println("Case4 ");

default:

System.out.println("Default ");

}

}

}

**Output:**

Case2

Case3

Case4

Default

In the above program, we have passed integer value 2 to the switch, so the control switched to the case 2, however we don’t have break statement after the case 2 that caused the flow to pass to the subsequent cases till the end. The solution to this problem is break statement

Break statements are used when you want your program-flow to come out of the switch body. Whenever a break statement is encountered in the switch body, the execution flow would directly come out of the switch, ignoring rest of the cases

Let’s take the same example but this time with break statement.

## Example with break statement

public class SwitchCaseExample2 {

public static void main(String args[]){

int i=2;

switch(i)

{

case 1:

System.out.println("Case1 ");

break;

case 2:

System.out.println("Case2 ");

break;

case 3:

System.out.println("Case3 ");

break;

case 4:

System.out.println("Case4 ");

break;

default:

System.out.println("Default ");

}

}

}

**Output:**

Case2

Now you can see that only case 2 had been executed, rest of the cases were ignored.

**Why didn’t I use break statement after default?**  
The control would itself come out of the switch after default so I didn’t use it, however if you still want to use the break after default then you can use it, there is no harm in doing that.

## Few points about Switch Case

1) Case doesn’t always need to have order 1, 2, 3 and so on. It can have any integer value after case keyword. Also, case doesn’t need to be in an ascending order always, you can specify them in any order based on the requirement.

2) You can also use characters in switch case. for example –

public class SwitchCaseExample2 {

public static void main(String args[]){

char ch='b';

switch(ch)

{

case 'd':

System.out.println("Case1 ");

break;

case 'b':

System.out.println("Case2 ");

break;

case 'x':

System.out.println("Case3 ");

break;

case 'y':

System.out.println("Case4 ");

break;

default:

System.out.println("Default ");

}

}

}

class JavaExample{

static int num;

static String mystr;

static{

num = 97;

mystr = "Static keyword in Java";

}

public static void main(String args[])

{

System.out.println("Value of num: "+num);

System.out.println("Value of mystr: "+mystr);

}

}

### Example 2: Multiple Static blocks

Lets see how multiple static blocks work in Java. They execute in the given order which means the first static block executes before second static block. That’s the reason, values initialized by first block are overwritten by second block.

class JavaExample2{

static int num;

static String mystr;

//First Static block

static{

System.out.println("Static Block 1");

num = 68;

mystr = "Block1";

}

//Second static block

static{

System.out.println("Static Block 2");

num = 98;

mystr = "Block2";

}

public static void main(String args[])

{

System.out.println("Value of num: "+num);

System.out.println("Value of mystr: "+mystr);

}

}

**Output:**

Static Block 1

Static Block 2

Value of num: 98

Value of mystr: Block2

## Java Static Variables

A static variable is common to all the instances (or objects) of the class because it is a class level variable. In other words you can say that only a single copy of static variable is created and shared among all the instances of the class. Memory allocation for such variables only happens once when the class is loaded in the memory.  
Few Important Points:

* Static variables are also known as Class Variables.
* Unlike **non-static variables**, such variables can be accessed directly in static and non-static methods.

### Example 1: Static variables can be accessed directly in Static method

Here we have a static method disp() and two static variables var1 and var2. Both the variables are accessed directly in the static method.

class JavaExample3{

static int var1;

static String var2;

//This is a Static Method

static void disp(){

System.out.println("Var1 is: "+var1);

System.out.println("Var2 is: "+var2);

}

public static void main(String args[])

{

disp();

}

}

**Output:**

Var1 is: 0

Var2 is: null

### Example 2: Static variables are shared among all the instances of class

In this example, String variable is non-static and integer variable is Static. As you can see in the output that the non-static variable is different for both the objects but the static variable is shared among them, thats the reason the changes made to the static variable by object ob2 reflects in both the objects.

class JavaExample{

//Static integer variable

static int var1=77;

//non-static string variable

String var2;

public static void main(String args[])

{

JavaExample ob1 = new JavaExample();

JavaExample ob2 = new JavaExample();

/\* static variables can be accessed directly without

\* any instances. Just to demonstrate that static variables

\* are shared, I am accessing them using objects so that

\* we can check that the changes made to static variables

\* by one object, reflects when we access them using other

\* objects

\*/

//Assigning the value to static variable using object ob1

ob1.var1=88;

ob1.var2="I'm Object1";

/\* This will overwrite the value of var1 because var1 has a single

\* copy shared among both the objects.

\*/

ob2.var1=99;

ob2.var2="I'm Object2";

System.out.println("ob1 integer:"+ob1.var1);

System.out.println("ob1 String:"+ob1.var2);

System.out.println("ob2 integer:"+ob2.var1);

System.out.println("ob2 STring:"+ob2.var2);

}

}

Output:

ob1 integer:99

ob1 String:I'm Object1

ob2 integer:99

ob2 STring:I'm Object2

For more details on refer: [Java – static variable](https://beginnersbook.com/2013/05/static-variable/)

## Java Static Methods

Static Methods can access class variables(static variables) without using object(instance) of the class, however non-static methods and non-static variables can only be accessed using objects.  
Static methods can be accessed directly in static and non-static methods.  
**Syntax:**  
Static keyword followed by return type, followed by method name.

static return\_type method\_name();

### Example 1: static method main is accessing static variables without object

class JavaExample{

static int i = 10;

static String s = "Beginnersbook";

//This is a static method

public static void main(String args[])

{

System.out.println("i:"+i);

System.out.println("s:"+s);

}

}

**Output:**

i:10

s:Beginnersbook

### Example 2: Static method accessed directly in static and non-static method

class JavaExample{

static int i = 100;

static String s = "Beginnersbook";

//Static method

static void display()

{

System.out.println("i:"+i);

System.out.println("i:"+s);

}

//non-static method

void funcn()

{

//Static method called in non-static method

display();

}

//static method

public static void main(String args[])

{

JavaExample obj = new JavaExample();

//You need to have object to call this non-static method

obj.funcn();

//Static method called in another static method

display();

}

}

Output:

i:100

i:Beginnersbook

i:100

i:Beginnersbook

Read more: [Static Method vs non-static Method in Java](https://beginnersbook.com/2013/05/static-vs-non-static-methods/)

## Static Class

A class can be made **static** only if it is a nested class.

1. Nested static class doesn’t need reference of Outer class
2. A static class cannot access non-static members of the Outer class

We will see these two points with the help of an example:

### Static class Example

class JavaExample{

private static String str = "BeginnersBook";

//Static class

static class MyNestedClass{

//non-static method

public void disp() {

/\* If you make the str variable of outer class

\* non-static then you will get compilation error

\* because: a nested static class cannot access non-

\* static members of the outer class.

\*/

System.out.println(str);

}

}

public static void main(String args[])

{

/\* To create instance of nested class we didn't need the outer

\* class instance but for a regular nested class you would need

\* to create an instance of outer class first

\*/

JavaExample.MyNestedClass obj = new JavaExample.MyNestedClass();

obj.disp();

}

}

Output:

BeginnersBook