**Control statements (flow control)**

In java we can write any number of statements which are executed in sequence order by default.

But if we want to execute java statements randomly according to our requirement then we have to use control statements

**In java we have following 3 types of control statements**

**1. Conditional statements**

1. if..else

2. switch-case

**2. Iterative statements**

1. for

2. while

3. do..while

4. foreach(not a java keyword)

**3. Transfering statements**

1.break

2.continue

3.return

**Java - Decision Making**

Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

**Following is the general form of a typical decision making structure found in most of the programming languages:**

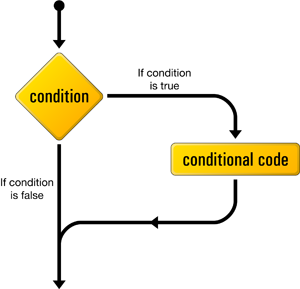


Java programming language provides following types of decision making statements.

|  |  |
| --- | --- |
| **Statement** | **Description** |
| [**if statement**](http://www.tutorialspoint.com/java/if_statement_in_java.htm) | An **if statement** consists of a boolean expression followed by one or more statements. |
| [**if...else statement**](http://www.tutorialspoint.com/java/if_else_statement_in_java.htm) | An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false. |
| [**nested if statements**](http://www.tutorialspoint.com/java/nested_if_statements_in_java.htm) | You can use one **if** or **else if** statement inside another **if** or **else if** statement(s). |
| [**switch statement**](http://www.tutorialspoint.com/java/switch_statement_in_java.htm) | A **switch** statement allows a variable to be tested for equality against a list of values. |

## If Statement in Java Programming : Conditional Selection

1. The if statement is a conditional branch statement.
2. It tells your program to **execute a certain section of code only if a particular test evaluates to true**.
3. If **boolean Expression** evaluates to true, the statements in the block following the if statement are executed.
4. If it evaluates to false, the statements in the **if block are not executed**.

[](http://img.c4learn.com/2012/03/if-statement-in-java-programming-language1.png)

## Syntax : If Statement

***if*** *(booleanExpression) {*

*statement(s)*

*}*

If **boolean Expression evaluates to false** and there is an **else block**, the statements in the else block are executed.

## Syntax : If-Else Statement

***if*** *(booleanExpression) {*

*statement(s)*

*}* ***else*** *{*

*statement(s)*

*}*

## Syntax : If-Else Ladder Statement

1. If we have multiple conditions then we can use if-else ladder –

***if*** *(booleanExpression1) {*

*// statements*

*}* ***else******if*** *(booleanExpression2) {*

*// statements*

*}...****else*** *{*

*// statements*

*}*

## Live Example 1 : Simple If-Else Ladder Statement

*class IfDemo {*

*public* ***static******void*** *main(String[] args) {*

***int*** *marks = 76;*

*String grade;*

***if*** *(marks >= 40) {*

*grade = "Pass";*

*}*

***if*** *(marks < 40) {*

*grade = "Fail";*

*}*

*System.out.println("Grade = " + grade);*

*}*

*}*

## Output :

*Grade = Pass*

## Live Example 2 : Simple If-Else Ladder Statement

*class IfDemo {*

*public* ***static******void*** *main(String[] args) {*

***int*** *marks = 76;*

*String grade;*

***if*** *(marks >= 40) {*

*grade = "Pass";*

*}****else****{*

*grade = "Fail";*

*}*

*System.out.println("Grade = " + grade);*

*}*

*}*

## Output :

*Grade = Pass*

## Live Example 3 : Simple If-Else Ladder Statement

*class IfElseDemo {*

*public* ***static******void*** *main(String[] args) {*

***int*** *marks = 76;*

***char*** *grade;*

***if*** *(marks >= 90) {*

*grade = 'A';*

*}* ***else******if*** *(marks >= 80) {*

*grade = 'B';*

*}* ***else******if*** *(marks >= 70) {*

*grade = 'C';*

*}* ***else******if*** *(marks >= 60) {*

*grade = 'D';*

*}* ***else*** *{*

*grade = 'F';*

*}*

*System.out.println("Grade = " + grade);*

*}*

*}*

## Output :

*Grade = C*

## Live Example 4 : We can have multiple conditions inside If

***if****(marks > 70 || marks < 90)*

*System.out.println("Class A");*

***else***

*System.out.println("Class B");*

1. We can use “&&” , “||” operators in order to refine condition.
2. && operator will check whether both left hand and right hand conditions are true or not.
3. || condition will check

## Must Read Note :

**If we have written two conditions inside “If Condition” then –**

1. **If First Condition is true then || operator will skip checking second condition.**
2. **If First Condition is false then || operator will check second condition if second condition is true then overall if will follow true block otherwise it will follow else block.**

### Proof –

*class Test{*

*public* ***static******void*** *main(String args[]) {*

***int*** *num1 = 10;*

***int*** *num2 = 20;*

***if*** *((++num1 > 0) || (++num2 > 0)) {*

*System.out.println(num1);*

*System.out.println(num2);*

*}*

*}*

*}*

### Output :

*11*

*20*

### Explanation :

1. First condition is true so it skip out checking second condition.

*class Test{*

*public* ***static******void*** *main(String args[]) {*

***int*** *num1 = 10;*

***int*** *num2 = 20;*

***if*** *((++num1 < 0) && (++num2 > 0)) {*

*System.out.println(num1);*

*System.out.println(num2);*

*}*

*System.out.println(num1);*

*System.out.println(num2);*

*}*

*}*

## Output :

*11*

*20*

### Explanation :

1. If **First Condition is true** then && operator will check second condition.
2. If **First Condition is false then** && operator will skip checking second condition.

## Alternate ways to write the above *if* statement

There are lots of ways to write the above if statement. Here are some.

1. Reverse the condition. Which to put first?

**There are two practices:**

Either put the normal case first, or the case that makes the boolean condition easiest to read.

***String comment; // Message to the user.***

*if (score < 60) {*

*comment = "This is terrible";*

*} else {*

*comment = "Not so bad";*

*}*

1. Initialize the variable to a default value, and only change it if necessary. This is often used when the condition is only rarely true.

*String comment = "Not so bad; // Message to the user.*

*if (score < 60) {*

*comment = "This is terrible";*

*}*

1. **BAD**:

Two if’s. This is almost always a bad way to write an if-else. It's confusing to read, border values can slip through, and both conditions must be evaluated (inefficiency).

***// BAD BAD BAD BAD BAD BAD BAD BAD BAD***

*String comment; // Message to the user.*

*if (score < 60) {*

*comment = "This is terrible";*

*}*

*if (score >= 60) {*

*comment = "Not so bad";*

*}*

## Brace style

**Always write braces.** It is good programming style to always write the curly braces, {}, although they are not needed if the clause contains only a single statement. There are two reasons this is good.

* **Reliability**. When code is modified, the indentation is such a strong indicator of structure that the programmer may not notice that the addition of a statement at the "correct" indentation level really isn't included in the scope of the *if* statement. This is a surprisingly common error.
* **Readability**. It is faster to read code with the braces because the reader doesn't have to keep in mind whether they are dealing with an un-braced single statement or a braced block.
* Java doesn't care about your indentation -- it is for humans (including yourself!).

## Example 1 - No indentation - BAD BAD BAD

Here is the paintComponent() method from a previous page without indentation. This is small, so it's easy to see which statements are in the true and false parts. If the if statement is much larger, it will be unreadable without indentation.

*public void paintComponent(Graphics g) {*

*super.paintComponent(g);*

*if (marks < 50)*

*g.setColor(Color.red);*

*else*

*g.setColor(Color.black);*

*g.drawString("Score = " + marks, 10, 50);*

*}*

## Example 2 - No indentation and no line breaks

Even a very short method is almost unreadable when you take out the line breaks and spaces. Here is the same method:

*public void paintComponent(Graphics g) {super.paintComponent(g);if (marks<50)g.setColor(Color.red);else g.setColor(Color.black);g.drawString("Score = " + marks,10,50);}*

## if inside if

You can put an if statement inside another if statement.

## Nearest 'else'

If you use braces, there is no problem with deciding which else goes with which if For example,

*if (age < 24) {*

*if (height > 200) {*

*c = Color.RED;*

*}*

*} else {*

*c = Color.BLUE;*

*}*

Because the true and false parts are both single statements, you might be tempted to omit braces and write:

*if (age < 24)*

*if (height > 200)*

*c = Color.RED;*

*else // DANGER: which 'if' goes with this 'else'*

*c = Color.BLUE;*

But this is WRONG, because 'else' always goes with the nearest 'if' when there are no braces. This code is the same as:

*if (age < 24) {*

*if (height > 200)*

*c = Color.RED;*

*else*

*c = Color.BLUE;*

*}*

## Advice: Always use braces on if statements

These kinds of errors are very hard to find. This is another good reason to *always use braces*.

## Watch out for semicolons on your if statements

Why does the following code always say it thinks the user is lying?

*String ageStr = JOptionPane.showInputDialog(null, "How old are you?");*

*int age = Integer.parseInt(ageStr);*

*if (age > 120 || age < 0);*

*System.out.println("I think you're lying about your age!");*

It's the semicolon! if you put a semicolon directly after the condition in an if statement, Java thinks it's finished with the body of the statement. The indentation of the next line, which is so important to human readers, is ignored by Java.

This is another error that's harder to make if you always follow the condition by an opening brace.

## Series of tests

It is common to make a series of tests on a value, where the else part contains only another if statement. If you use indentation for the else part, it isn't easy to see that these are really a series of tests which are similar. It is better to write them at the same indentation level by writing the if on the same line as the else.

## Example -- series of tests - cascading ifs

This code is correctly indented, but ugly and hard to read. It also can go very far to the right if there are many tests.

*if (score < 35) {*

*g.setColor(Color.MAGENTA);*

*} else {*

*if (score < 50) {*

*g.setColor(Color.RED);*

*} else {*

*if (score < 60) {*

*g.setColor(Color.ORANGE);*

*} else {*

*if (score < 80) {*

*g.setColor(Color.YELLOW);*

*} else {*

*g.setColor(Color.GREEN);*

*}*

*}*

*}*

*}*

## Example -- using 'else if' style for formatting

Here is the same example, using a style of writing the if immediately after the else. This is a common exception to the indenting rules, because it results in more readable programs. Note that it makes use of the rule that a single statement in one of the Java clauses doesn't need braces.

*if (score < 35) {*

*g.setColor(Color.MAGENTA);*

*} else if (score < 50) {*

*g.setColor(Color.RED);*

*} else if (score < 60) {*

*g.setColor(Color.ORANGE);*

*} else if (score < 80) {*

*g.setColor(Color.YELLOW);*

*} else {*

*g.setColor(Color.GREEN);*

*}*

## Other languages

Some programming languages recognize this common construction with a special *elseif* keyword. Although it is hardly necessary, this kind of small touch can make a language a little nicer to use. The Java language designers are very conservative about adding keywords to the language, so don't expect it.

## Java IF-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

**Syntax:**

***if****(condition1){*

*//code to be executed if condition1 is true*

*}****else******if****(condition2){*

*//code to be executed if condition2 is true*

*}*

***else******if****(condition3){*

*//code to be executed if condition3 is true*

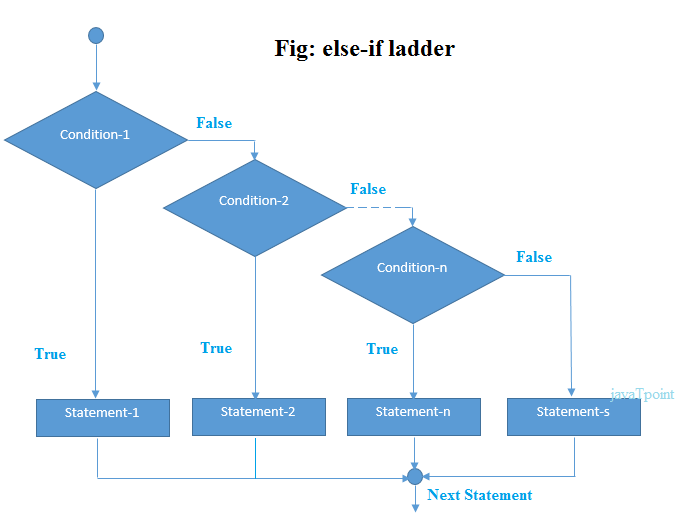
*}*

*...*

***else****{*

*//code to be executed if all the conditions are false*

*}*



**Example:**

***public******class****IfElseIfExample {*

***public******static******void****main(String[] args) {*

***int****marks=65;*

***if****(marks<50){*

*System.out.println("fail");*

*}****else******if****(marks>=50 && marks<60){*

*System.out.println("D grade");*

*}****else******if****(marks>=60 && marks<70){*

*System.out.println("C grade");*

*}****else******if****(marks>=70 && marks<80){*

*System.out.println("B grade");*

*}****else******if****(marks>=80 && marks<90){*

*System.out.println("A grade");*

*}****else******if****(marks>=90 && marks<100){*

*System.out.println("A+ grade");*

*}****else****{*

*System.out.println("Invalid!");*

*}*

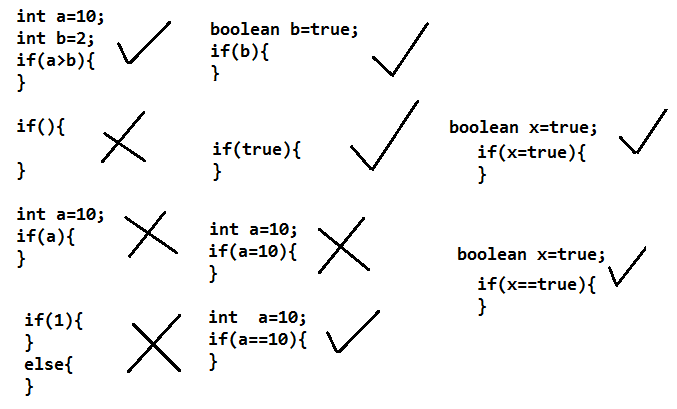
*}*

*}*

**Output:** *C grade*

**Rules**

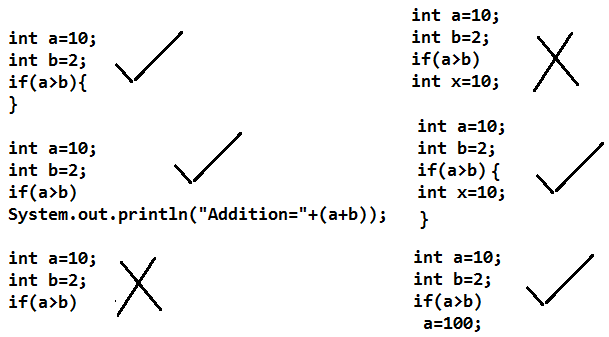
1. We can execute either if block or else block but we never execute both the blocks at a time.
2. We can skip either if block or else block but we never skip both the blocks at a time.
3. For if statement writing else block is optional but we can not write else block directly without if statement.
4. In If statement writing condition or expression mandatory which must be boolean type.



1. If we write more than one statement writing boundaries({,}) are required otherwise if we write 1 statement then writing boundaries({,}) are optional.
2. If we don’t write boundaries({,}) then,

-> we can write only one statement

-> writing this single statement is mandatory

 -> this single statement must not be any declarative statement

***// wap tocheck whether the given number is even or odd***

*class IfDemo2{*

*public static void main(String args[]){*

*int a =7;*

*if(a%2==0){*

*System.out.println(a+" is a Even Number ");*

*}else{*

*System.out.println(a+"is a Odd Number ");*

*}*

*}*

*}*

**//wap to check whether the given number is Positive or negative**

*class IfDemo3{*

*public static void main(String args[]){*

*int a =0;*

*if(a>0){*

*System.out.println(a+" is a Positive Number ");*

*}else if(a<0){*

*System.out.println(a+" is a Negative Number ");*

*}else{*

*System.out.println( "it is just zero ");*

*}*

*}*

*}*

***//wap to read the values from keyboard***

*import java.io.\*;*

*class ReadingData{*

*public static void main(String args[]) throws IOException{*

*BufferedReader br=new BufferedReader( new InputStreamReader(System.in));*

*System.out.println("Enter Any string");*

*String str = br.readLine();*

*System.out.println("str="+str);*

*System.out.println("Enter Any Integer");*

*int i = Integer.parseInt(br.readLine()); //parsing*

*System.out.println("i="+i);*

*System.out.println("Enter Any Double");*

*double d = Double.parseDouble(br.readLine());*

*System.out.println("d="+d);*

*System.out.println("Enter Any character");*

*char ch = (char) br.read(); //type casting*

*System.out.println("ch="+ch);*

*}*

*}*

**//wap to find big number in given 3 numbers**

*import java.io.\*;*

*class IfDemo4{*

*public static void main(String args[]) throws IOException{*

*BufferedReader br = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter Any 3 Numbers");*

*int a = Integer.parseInt( br.readLine());*

*int b = Integer.parseInt( br.readLine());*

*int c = Integer.parseInt( br.readLine());*

*if(a>=b&&a>=c){*

*System.out.println(a+" Is Big Number");*

*}else if(b>=c&&b>=a){*

*System.out.println(b+" Is Big Number");*

*}else{*

*System.out.println(c+" Is Big Number");*

*}*

*}*

*}*

**//wap to check whether the given character is vowel or not**

*import java.io.\*;*

*class IfDemo5{*

*public static void main(String args[]) throws IOException{*

*BufferedReader br = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter Any Character");*

*char ch = (char) br.read();*

*if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u'){*

*System.out.println(" It Is Vowel");*

*}else{*

*System.out.println(" It Is Not Vowel");*

*}*

*}*

*}*

**Assignments**

wap to check whether the user is eligible for voting or not

wap to check whether the given number is multiple of 5 or not

wap to check whether the given character is alphabet or not

wap to find bill amount according to the following information

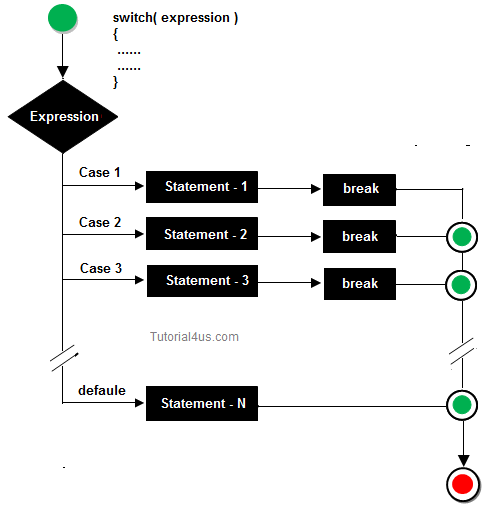
|  |  |
| --- | --- |
| Amount | Discount |
| <5000 | 10% |
| >=5000 and <10000 | 15% |
| >=10000 and <20000 | 20% |
| >=20000 | 50% |

take qty, rate and calculate amt, dis and finalbill

## *Java switch case*

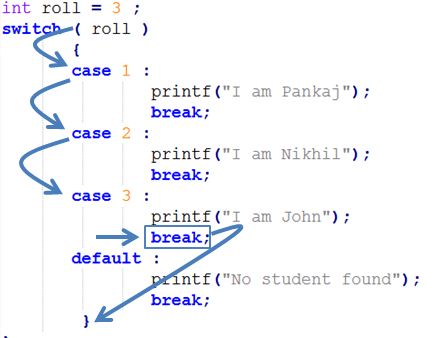
The **switch** statement in java language is used to execute the code from multiple conditions or case. It is same like if else-if ladder statement.

A switch statement work with byte, short, char and int primitive data type, it also works with enumerated types and string.



## Syntax : Switch Case in Java Programming

1. It is alternative to else-if ladder.
2. Switch Case Syntax is similar to – [C/C++ Switch](http://www.c4learn.com/c-programming/c-switch-case-statement/).
3. Switch allows you to choose a block of statements to run from a selection of code, based on the return value of an expression.
4. The expression used in the switch statement must return **an int, a String, or an enumerated value**.

[](http://img.c4learn.com/2012/03/How-Switch-Case-works-in-Java-Programming-language.jpg)

*switch (expression) {*

***case*** *value\_1 :*

*statement(s);*

***break****;*

***case*** *value\_2 :*

*statement(s);*

***break****;*

*. . .*

***case*** *value\_n :*

*statement(s);*

***break****;*

***default****:*

*statement(s);*

*}*

## Different Ways of Using Switch Case :

### Way 1 : Switch Case Using Integer Case Label

*switch (i) {*

***case*** *1 :*

***System****.out.println("One player is playing this game.");*

***break****;*

***case*** *2 :*

***System****.out.println("Two players are playing this game.");*

***break****;*

***case*** *3 :*

***System****.out.println("Three players are playing this game.");*

***break****;*

***default****:*

***System****.out.println("You did not enter a valid value.");*

*}*

### Way 2 : Switch Using String

*String name = "Pritesh";*

*switch (name) {*

***case*** *"Pritesh" :*

***System****.out.println("One player is playing this game.");*

***break****;*

***case*** *"Suraj" :*

***System****.out.println("Two players are playing this game.");*

***break****;*

***case*** *"Raj" :*

***System****.out.println("Three players are playing this game.");*

***break****;*

***default****:*

***System****.out.println("You did not enter a valid value.");*

*}*

If you are coming from C/C++ then you must understand this concept that – Java Supports String Inside Case Label.

Above program if used in C/C++ then it will give us compile time error.

## Purpose of switch: select one of many possible statements to execute

The if statement allows you to select one of two sections of code to execute based on a boolean value (only two possible values). The switch statement allows you to choose from many statements based on an integer (including char) or enum value.

### Syntax example

***switch*** *(expr) {*

***case*** *c1:*

*statements // do these if expr == c1*

***break****;*

***case*** *c2:*

*statements // do these if expr == c2*

***break****;*

***case*** *c2:*

***case*** *c3:*

***case*** *c4: // Cases can simply fall thru.*

*statements // do these if expr == any of c's*

***break****;*

*. . .*

***default****:*

*statements // do these if expr != any above*

*}*

## Switch keywords

**Switch**

The switch keyword is followed by a parenthesized integer expression, which is followed by the *cases*, all enclosed in braces.. The switch statement executes the case corresponding to the value of the expression. Normally the code in a case clause ends with a break statement, which exits the switch statement and continues with the statement following the switch. If there is no corresponding case value, the default clause is executed. If no case matched and there is no default clause, execution continues after the end of the switch statement.

**case**

The case keyword is followed by an integer constant and a colon. This begins the statements that are executed when the switch expression has that case value.

**default**

If no case value matches the switch expression value, execution continues at the default clause. This is the equivalent of the "else" for the switch statement. It is written after the last case be convention, and typically isn't followed by break because execution just continues out the bottom of switch if this is the last clause.

**break**

The break statement causes execution to exit to the statement after the end of the switch. If there is no break, execution flows thru into the next case. Flowing directly into the next case is almost always an error.

## Example - Random comment

*String comment; // The generated insult.*

*int which = (int)(Math.random() \* 3); // Result is 0, 1, or 2.*

*switch (which) {*

*case 0: comment = "You look so much better than usual.";*

*break;*

*case 1: comment = "Your work is up to its usual standards.";*

*break;*

*case 2: comment = "You're quite competent for so little experience.";*

*break;*

*default: comment = "Oops -- something is wrong with this code.";*

*}*

## Equivalent if statement

A switch statement can often be rewritten as an if statement in a straightforward manner. For example, the preceding switch statement could be written as follows. When one of a number of blocks of code is selected based on a single value, the switch statement is generally easier to read. The choice of if or switch should be based on which is more readable.

*String comment; // The generated insult.*

*int which = (int)(Math.random() \* 3); // Result is 0, 1, or 2.*

*if (which == 0) {*

*comment = "You look so much better than usual.";*

*} else if (which == 1) {*

*comment = "Your work is up to its usual standards.";*

*} else if (which == 2) {*

*comment = "You're quite competent for so little experience.";*

*} else {*

*comment = "Oops -- something is wrong with this code.";*

*}*

## Defensive programming

Always include a *default* clause in your switch statement as a general policy of *defensive programming* - assume there will be bugs in your code and make sure they are caught.

## Where to use switch?

The ability of switch to choose between many sections of code seems to make it more powerful than *if*. However, selecting sections of code depending on specific integer values turns out not to be very common. If you are handling specific coded values (eg, the number of the button that was clicked in a JOptionPane), or processing characters (whose codes are treated like numbers), you may find it useful.

**Efficiency?**   
Some compilers can produce more efficient code for certain switch statements than for equivalent ifstatements. I haven't bothered to test the Java compiler because, if there is a speed difference, it would be extremely small and the choice between switch and if should be based on readability.

## Comments on switch

Java's switch statement, which was taken directly from C++ to increase its attractiveness to C++ programmers, is not well loved.

* **No ranges**. It doesn't allow ranges, eg case 90-100:. Many other languages do.
* **Integers only**. It requires integers and doesn't allow useful types like String. Many other languages do.
* **Error-prone**. It is error-prone and a common source of bugs - forgetting break or default silently ignores errors. Some languages have eliminated these dangerous situations.

## How Switch Case Statement is different in java from c programming

1. In C Programming we were unable to write Case label of data type  ‘String’. In java we can have String inside “Switch”.

## Live Example 1 : Choose Day of the Week

***class*** *SwitchDate{*

***public******static*** *void main(****String****[] args){*

*int week = 3;*

***switch*** *(week){*

***case*** *1:****System****.out.println("monday");* ***break****;*

***case*** *2:****System****.out.println("tuesday");****break****;*

***case*** *3:****System****.out.println("wednesday");****break****;*

***case*** *4:****System****.out.println("thursday");****break****;*

***case*** *5:****System****.out.println("friday");****break****;*

***case*** *6:****System****.out.println("saturday");****break****;*

***case*** *7:****System****.out.println("sunday");****break****;*

***default****:****System****.out.println("Invalid week");****break****;*

*}*

*}*

*}*

## Limitations of switch statement

**Logical operators cannot be used with switch statement. For instance**

## *Example*

***case*** *k>=20: // not allowed*

## Example of switch case

***import*** *java.util.\*;*

***class*** *switchCase{*

***public******static******void*** *main(String arg[]){*

***int*** *ch;*

*System.****out****.println("Enter any number (1 to 7) :");*

*Scanner s=****new*** *Scanner(System.****in****);*

*ch=s.nextInt();*

***switch****(ch){*

***case*** *1:*

*System.****out****.println("Today is Monday");*

***break****;*

***case*** *2:*

*System.****out****.println("Today is Tuesday");*

***break****;*

***case*** *3:*

*System.****out****.println("Today is Wednesday");*

***break****;*

***case*** *4:*

*System.****out****.println("Today is Thursday");*

***break****;*

***case*** *5:*

*System.****out****.println("Today is Friday");*

***break****;*

***case*** *6:*

*System.****out****.println("Today is Saturday");*

***break****;*

***case*** *7:*

*System.****out****.println("Today is Sunday");*

***default****:*

*System.****out****.println("Only enter value 1 to 7");*

*}*

*}*

*}*

## Output

***Enter any number (1 to 7) :***

*5*

*Today is Friday*

## Java Switch Statement is fall-through

The java switch statement is fall-through. It means it executes all statement after first match if break statement is not used with switch cases.

**Example:**

***public******class****SwitchExample2 {*

***public******static******void****main(String[] args) {*

***int****number=20;*

***switch****(number){*

***case****10: System.out.println("10");*

***case****20: System.out.println("20");*

***case****30: System.out.println("30");*

***default****:System.out.println("Not in 10, 20 or 30");*

*}*

*}*

*}*

**Output:**

20

30

Not in 10, 20 or 30

**Note:**

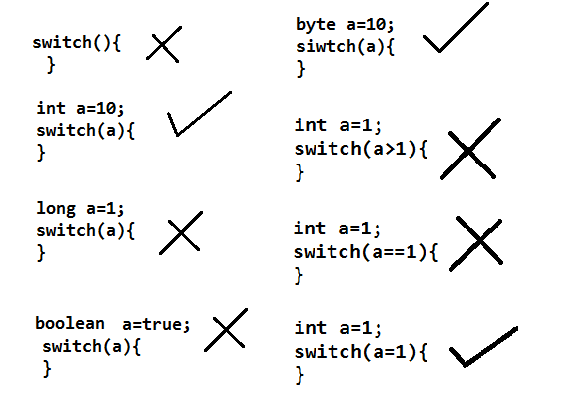
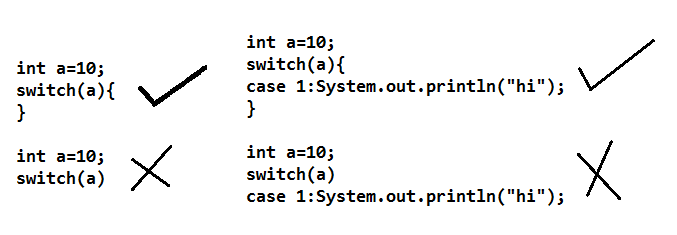
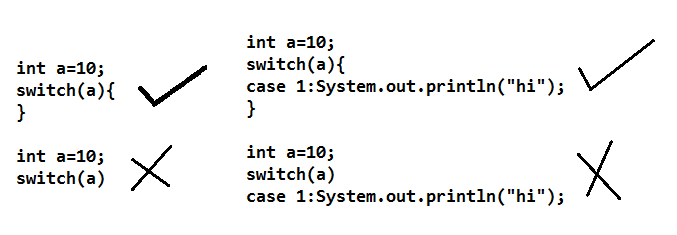
1. We can say switch-case is replacement to the if..else..if ladder when it contains many equal comparisons.

2. But if..else..if ladder contains any other >,<,!=,>=,<= comparisons then we can not say switch-case is the replacement to if..else..if ladder.

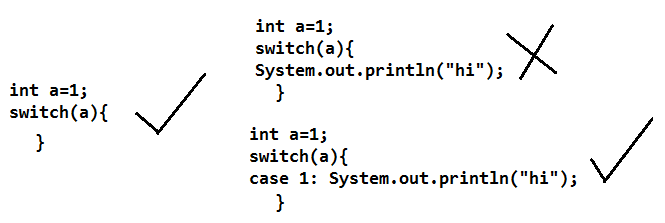
3.Generally we go for switch-case when we want to make multiple equal comparisons

**Rules:**

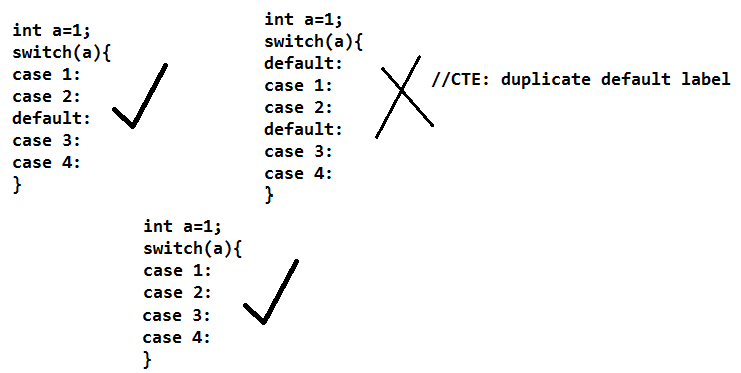
1. Writing the expression or value for switch-case is mandatory which must be any byte/ short/ int/ char, but it should not be any long/ float/ double/ Boolean.

  
2. In switch -case we write any number of statements every time we have to provide boundaries({,})it means mandatory.

3. In switch-case writing case statements or default statements are optional. But if we want to write any statement we have to write inside the case statements or default statements.

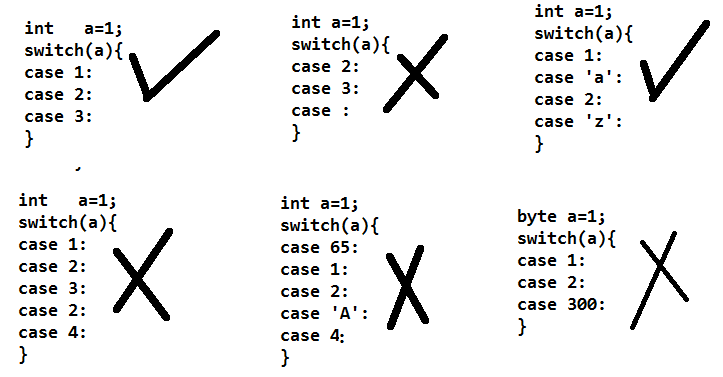


4.-In switch-case we can write default statement any where inside the switch-case but it is recommended to write default statements in the ending section of the switch-case.

 - At most 1 default statement we can write.

5. -In switch-case we can write any number of case statements if write a case its label must be specified which should not be duplicated

- In a case we can write any number of statements



6. By default switch-case will enter into any case if case label is successfully compared and execute until end of the switch-case or until the break statement found this nature of switch-case is called fall-through behavior

- But if we want to stop this continuity of switch-case and execute the statements according to our requirement we have to use break statement.

- break is transferring statement which will stop the continuity of switch-case and transfer the control from inside the switch-case to outside the switch case.

**//wap to perform all arithmetical operations using switch-case**

*import java.io.\*;*

*class Switch2{*

*public static void main(String args[]) throws IOException{*

*BufferedReader br = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enterr Any 2 Numbers");*

*int a= Integer.parseInt(br.readLine());*

*int b=Integer.parseInt(br.readLine());*

*System.out.println("1.Addition");*

*System.out.println("2.Subtraction");*

*System.out.println("3.Multiplication");*

*System.out.println("4.Quotient");*

*System.out.println("5.Remainder");*

*System.out.println("Enter Your Choice");*

*int choice=Integer.parseInt(br.readLine());*

*switch(choice){*

*case 1: System.out.println("Addition="+(a+b));*

*break;*

*case 2: System.out.println("Subtraction="+(a-b));*

*break;*

*case 3: System.out.println("Multiplication="+(a\*b));*

*break;*

*case 4: System.out.println("Quotient="+(a/b));*

*break;*

*case 5: System.out.println("Remainder="+(a%b));*

*break;*

*default: System.out.println("Invalid choice");*

*}*

*}*

*}*

**// wap to find whether the given character is vowel or not using switch-case**

*import java.io.\*;*

*class Switch3{*

*public static void main(String args[]) throws IOException{*

*BufferedReader br = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter Any Character");*

*char ch= (char) br.read();*

*switch(ch){*

*case 'a':*

*case 'e':*

*case 'i':*

*case 'o':*

*case 'u':*

*case 'A':*

*case 'E':*

*case 'I':*

*case 'O':*

*case 'U': System.out.println("It is vowel");*

*break;*

*default: System.out.println("It is not a vowel");*

*}*

*}*

*}*

**Assignments**

- Wap to perform all arithmetical operations using switch-case by taking arithmetical operator.

- Wap to display month name for given month number

- Wap to display the season of the given month

- Wap to calculate the Bill according to following information

|  |  |
| --- | --- |
| Room Type | Per Day charge |
| Suit | 1800 rs/- |
| Deluxe | 1350 rs/- |
| Ac | 1200 rs/- |
| Ordinary | 800 rs/- |

***JAVA-LOOP CONTROL***

A **loop** statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:



Java programming language provides the following types of loop to handle looping requirements.

|  |  |
| --- | --- |
| **Loop Type** | **Description** |
| [**while loop**](http://www.tutorialspoint.com/java/java_while_loop.htm) | Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body. |
| [**for loop**](http://www.tutorialspoint.com/java/java_for_loop.htm) | Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| [**do...while loop**](http://www.tutorialspoint.com/java/java_do_while_loop.htm) | Like a while statement, except that it tests the condition at the end of the loop body |

## Java for loop

## For Loop in Java Programming :

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loop in java.

* **Simple For Loop**
* **For-each or Enhanced For Loop**
* **Labeled For Loop**

## Java Simple For Loop

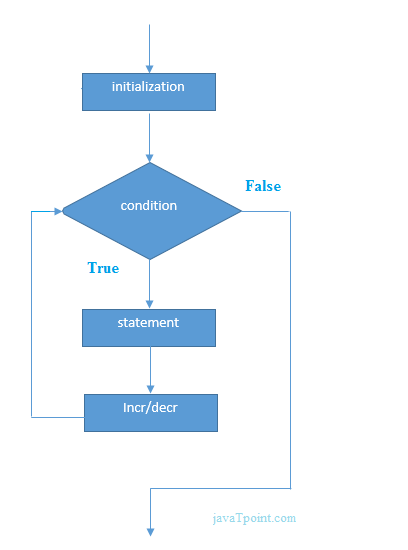
The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

**Syntax:**

***for****(initialization;condition;incr/decr){*

*//code to be executed*

*}*



**Example:**

***public******class****ForExample {*

***public******static******void****main(String[] args) {*

***for****(****int****i=1;i<=10;i++){*

*System.out.println(i);*

*}*

*}*

*}*

**Output:**

1

2

3

4

5

6

7

8

9

10

## Java For-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

**Syntax:**

***for****(Type var:array){*

*//code to be executed*

*}*

**Example:**

***public******class****ForEachExample {*

***public******static******void****main(String[] args) {*

***int****arr[]={12,23,44,56,78};*

***for****(****int****i:arr){*

*System.out.println(i);*

*}*

*}*

*}*

**Output:**

12

23

44

56

78

## Java Labeled For Loop

We can have name of each for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.

Normally, break and continue keywords breaks/continues the inner most for loop only.

**Syntax:**

*labelname:*

*for(initialization;condition;incr/decr){*

*//code to be executed*

*}*

**Example:**

***public******class****LabeledForExample {*

***public******static******void****main(String[] args) {*

*aa:*

***for****(****int****i=1;i<=3;i++){*

*bb:*

***for****(****int****j=1;j<=3;j++){*

***if****(i==2&&j==2){*

***break****aa;*

*}*

*System.out.println(i+" "+j);*

*}*

*}*

*}*

*}*

**Output:**

1 1

1 2

1 3

2 1

If you use **break bb;**, it will break inner loop only which is the default behavior of any loop.

***public******class****LabeledForExample {*

***public******static******void****main(String[] args) {*

*aa:*

***for****(****int****i=1;i<=3;i++){*

*bb:*

***for****(****int****j=1;j<=3;j++){*

***if****(i==2&&j==2){*

***break****bb;*

*}*

*System.out.println(i+" "+j);*

*}*

*}*

*}*

*}*

**Output:**

1 1

1 2

1 3

2 1

3 1

3 2

3 3

## Java Infinitive For Loop

If you use two semicolons ;; in the for loop, it will be infinitive for loop.

**Syntax:**

***for****(;;){*

*//code to be executed*

*}*

**Example:**

***public******class****ForExample {*

***public******static******void****main(String[] args) {*

***for****(;;){*

*System.out.println("infinitive loop");*

*}*

*}*

*}*

**Output:**

infinitive loop

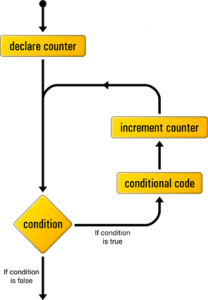
infinitive loop

infinitive loop

infinitive loop

infinitive loop

1. For Loop is one of the looping statement in java programming.
2. For Loop is used to execute set of statements repeatedly until the condition is true.
3. For Loop checks the contrition and executes the set of the statements , It is loop control statement in java.
4. For Loop contain the following statements such as “**Initialization**” , “**Condition**” and “**Increment/Decrement**” statement.

[](http://www.c4learn.com/wp-content/uploads/2012/03/for-Loop.png)

## Example - Printing a table of squares

Here is a loop written as both a *while* loop and a *for* loop. First using *while*:

*int number = 1;*

*while (number <= 12) {*

*System.out.println(number + " squared is " + (number \* number));*

*number++;*

*}*

And here is the same loop using *for*.

*for (int number = 1; number <= 12; number++) {*

*System.out.println(number + " squared is " + (number \* number));*

*}*

## Example - Counting doubled characters

This code will look at each character in a string, *sentence*, and count the number of times any character occurs doubled.

*String sentence = ...;*

*int doubleCount = 0; // Number of doubled characters.*

*// Start at second char (index 1).*

*for (int pos = 1; pos < sentence.length(); pos++) (*

*// Compare each character to the previous character.*

*if (sentence.charAt(pos) == sentence.charAt(pos-1)) {*

*doubleCount++;*

*}*

*}*

## Syntax For Loop : Java Programming

***for*** *(initialization; condition ; increment) {*

*Statement1*

*Statement2*

*...*

*StatementN*

*}*

## Live Example : For Loop Statement

*class ForDemo {*

*public* ***static******void*** *main(String[] args){*

***for****(****int*** *i=1; i<11; i++){*

*System.out.println("Count is : " + i);*

*}*

*}*

*}*

## Output :

*Count is: 1*

*Count is: 2*

*Count is: 3*

*Count is: 4*

*Count is: 5*

*Count is: 6*

*Count is: 7*

*Count is: 8*

*Count is: 9*

*Count is: 10*

## Explanation :

### 1. Initialization

1. Consider above program , in this program we are able to see that for loop statement contain 3 statements first one is “Initialization”.
2. ‘i’ is loop control variable used to control the complete loop.
3. Its value is initialized with 1.
4. Initialization statement gets executed once.

### 2. Condition

1. Inside first step value of i is initialized .
2. After initialization the value of control variable is checked against the condition present inside “Condition”.
3. i < 11 is true condition so that body part of the loop gets executed and it print the Count.
4. As soon as it finishes the execution of the body part it will jump to the “Increment/Decrement” statement.

### 3. Increment / Decrement

1. It will alter the value of Control variable by incrementing it or by decrementing it.
2. After alteration it again executes “Condition” of the loop.
3. Again if the condition evaluates to be true then it will again executes the body part as long as condition remains true.

## Keep in mind :

1. **Initialization expression** initializes the loop
2. **Initialization expression** executed once when the loop begins.
3. When the **Condition expression** evaluates to false, the loop terminates.
4. The **Increment expression** is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

**Consider Following Program :**

*class Sample {*

*public* ***static******void*** *main(String[] args){*

***for****(****int*** *i=1; i ; i++){*

*System.out.println("Count is : " + i);*

*}*

*}*

*}*

### In Java Integer is not an True Expression :

1. It will throw error since In Java we cannot consider Positive integer as “True”.
2. We must use comparison operator in order to make expression True.
3. We can Use boolean Operators instead.

*C:Java>javac Sample.java*

*Sample.java:3: incompatible types*

*found :* ***int***

*required: boolean*

***for****(****int*** *i=1; i ; i++){*

*^*

*1 error*

## Legal and Illegal Ways of For Loop :

### Example 1 : Compile Error

*public* ***static******void*** *main(String[] args){*

***int*** *exp;*

***for****(****int*** *i=1; exp ; i++){*

*System.out.println("Count is : " + i);*

*}*

*}*

### Example 2 : Error Free

*public* ***static******void*** *main(String[] args){*

*boolean exp;*

***for****(****int*** *i=1; exp ; i++){*

*System.out.println("Count is : " + i);*

*}*

*}*

**//wap to demo on for loop and print 1-10 numbers**

class For1{

public static void main(String args[]){

for(int i=1;i<=10;i++){

System.out.println(i);

}

}

}

**//wap to print Even Numbers between 1-100**

*class For2{*

*public static void main(String args[]){*

*for(int i=1;i<=100;i++){*

*if(i%2==0)*

*System.out.println(i);*

*}*

*}*

*}*

**Rules**

1. In for loop 3 sections must be separated by using 2 " ; " symbols.

2. writing 3 sections in for loop are optional.

- if we dont write anything in initialization section then compiler wont write anything.

- if we dont write anything in Incr/Decr section then compiler wont write anything.

 -But if we dont write anything in condition section then compiler always write boolean value true.

3.-In initialization section we can write any valid java statements

-In Incr/Decr section also we can write any valid java statements wont write anything.

-But In condition section we can write anything which must be boolean type

***Eg:***

*class ForTest{*

*public static void main(String args[]){*

*int i=1;*

*for( System.out.println("HI");i<=10; System.out.println("java")){*

*System.out.println(i);*

*i++;*

*}*

*}*

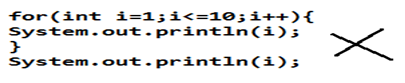
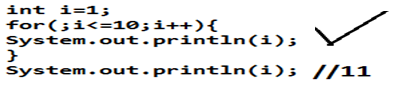
*}*

4.-In initialization section we can write any number of initializations which must be same type

-In Incr/Decr section also we can write any number of incr/decr statements which must be separated by ","

 -In condition section also we can write any number of conditions which must be combined using logical operators.

5. variables declared inside the for loop we can not access out side the for loop

6. In for loop if we write multiple statements then writing curly braces are mandatory otherwise if we write single statement then writing curly braces are optional, but in this case writing statement is mandatory and it should not be any declarative statement.

**//wap to find factorial of the given number**

*import java.io.\*;*

*class For3{*

*public static void main(String args[]) throws IOException{*

*BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter Any Number");*

*int n=Integer.parseInt(kbd.readLine());*

*int f=1;*

*for(int i=1;i<=n;i++){*

*f = f \* i;*

*}*

*System.out.println(n+" != "+f);*

*}*

*}*

**//wap to print mathematical table of given number**

*import java.io.\*;*

*class For4{*

*public static void main(String args[]) throws IOException{*

*BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter Any Number");*

*int n=Integer.parseInt(kbd.readLine());*

*System.out.println("Mathematical of "+n);*

*for(int i=1;i<=10;i++){*

*System.out.println(n+"X"+i+"="+n\*i);*

*}*

*}*

*}*

**//wap to check whether the given number is prime number or not**

*import java.io.\*;*

*class PrimeNumber{*

*public static void main(String args[]) throws IOException{*

*BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter Any Number");*

*int n=Integer.parseInt(kbd.readLine());*

*int c=0;*

*System.out.println("Factors of "+n);*

*for(int i=1;i<=n;i++){*

*if(n%i==0){*

*System.out.println(i);*

*c++;*

*}*

*}*

*if(c==2){*

*System.out.println("It is a PrimeNumber");*

*} else{*

*System.out.println("It is Not a PrimeNumber");*

*}*

*}*

*}*

**Assignments**

\*Wap to calculate sum of 1-100 numbers

\*Wap to print 1-100 Even numbers without using % operator

\*Wap to print 1-100 in reverse order

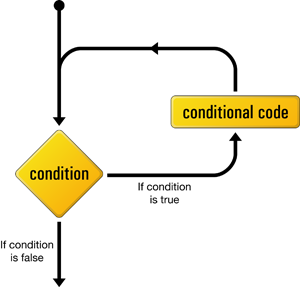
\*Wap to print power of given 2 numbers.

## Java while loop

## While Loop statement in Java : Iterative Statement (Loop Control Statement)

1. In java “**while**” is iteration statements like for and do-while.
2. It is also called as “**Loop Control Statement**“.
3. “**While Statement**” repeatedly executes the same set of instructions until a termination condition is met.
4. While loop is **Entry Controlled Loop** because condition is check at the entrance.
5. If initial condition is true then and then only control enters into the while loop body
6. In[for loop](http://www.c4learn.com/java/java-for-loop/) initialization,condition and increment all three statements are combined into the one statement , in “**while loop**” all these statements are written **as separate statements**.
7. Conditional Expression written inside while must **return boolean value**.

## Flow Diagram :

[](http://img.c4learn.com/2012/03/while-loop-in-java-programming-language.png)

## Syntax :

***while****(condition) {*

*// body of loop*

*}*

* The condition is any Boolean expression.
* The body of the loop will be executed as long as the conditional expression is true.
* When condition becomes false, control passes to the next line of code immediately following the loop.
* The **curly braces are unnecessary if only a single statement is being repeated**.

## Live Example :

class WhileDemo {

public **static** **void** main(String[] args){

**int** cnt = 1;

**while** (cnt < 11) {

System.out.println("Number Count : " + cnt);

count++;

}

}

}

## Java Infinitive While Loop

If you pass **true** in the while loop, it will be infinitive while loop.

**Syntax:**

***while****(****true****){*

*//code to be executed*

*}*

**Example:**

***public******class****WhileExample2 {*

***public******static******void****main(String[] args) {*

***while****(****true****){*

*System.out.println("infinitive while loop");*

*}*

*}*

*}*

**Output:**

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

## Legal Ways of Writing While loop :

### While Loop : Single Statement Inside Loop Body

*class WhileDemo {*

*public* ***static******void*** *main(String[] args){*

***int*** *cnt = 1;*

***while*** *(cnt < 11)*

*System.out.println("Number Count : " + cnt++);*

*}*

*}*

Single Statement is a part of While Loop as there is no opening and closing curly braces.

Suppose we want to embed multiple statements as a part of while loop body then we can put all the statements in a block.

### While Loop : Boolean Condition

*class WhileDemo {*

*public* ***static******void*** *main(String[] args){*

*boolean b1 = true;*

***while*** *(b1){*

*System.out.println("Number Count : " + cnt);*

*cnt++;*

***if****(cn==5)*

*b1 = false;*

*}*

*}*

*}*

We can put single variable as while condition but it must be of type boolean. thus the following statement will cause compile time failure –

***int*** *b1 = 12;*

***while*** *(b1){*

*System.out.println("Number Count : " + cnt);*

*cnt++;*

***if****(cn==5)*

*b1 = 1;*

*}*

### While Loop with no body

*class NoBody {*

*public* ***static******void*** *main(String args[]) {*

***int*** *i, j;*

*i = 10;*

*j = 20;*

***while****(i < j); // no body in this loop*

*{*

*System.out.println("Out of the Loop");*

*}*

*}*

*}*

**\*Note :** Semicolon after while condition is perfectly legal.

## Example with flowchart

The following shows some code and the equivalent flowchart.

|  |  |
| --- | --- |
| *int n = 0;*  *int i = 1;*  *while (i < 4) {*  *n++;*  *i++;*  *}*  *. . .* | while flowchart |

## Here is another example of code and an equivalent flowchart

|  |  |
| --- | --- |
| *int n = 0;*  *int i = 1;*  *while (i < 4) {*  *int j = 1;*  *while (j<=i) {*  *n += 1;*  *j++;*  *}*  *i = i + 1;*  *}* | *nested while flowchart* |

**//wap to print 1-10 numbers using while loop**

*class While1{*

*public static void main(String args[]){*

*int i=1;*

*while(i<=10){*

*System.out.println(i);*

*i++;*

*}*

*}*

*}*

**Rules**

1.In while loop writing the condition or expression is mandatory which must be boolean type.

2. In while loop if we write multiple statements then writing curly braces are mandatory otherwise if we write single statement then writing curly braces are optional, but in this case writing statement is mandatory and it should not be any declarative statement.

3. In java every statement should get a control and execute at any 1 point of time and compiler never support unreachable statements.

**//wap to find whether the given number is perfect number or not**

*import java.io.\*;*

*class While2{*

*public static void main(String args[]) throws IOException{*

*BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter Any Number");*

*int n=Integer.parseInt(kbd.readLine());*

*int sum=0;*

*System.out.println("Factors of "+n);*

*int i=1;*

*while(i<=n/2){*

*if(n%i==0){*

*System.out.println(i);*

*sum=sum+i;*

*}*

*i++;*

*}*

*if(sum==n){*

*System.out.println("It is a Perfect Number");*

*}else{*

*System.out.println("It is Not a Perfect Number");*

*}*

*}*

*}*

**//wap to find sum of the digits of given number**

*import java.io.\*;*

*class While3{*

*public static void main(String args[]) throws IOException{*

*BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter Any Number");*

*int n=Integer.parseInt(kbd.readLine());*

*int sum=0;*

*int r;*

*while(n!=0){*

*r=n%10;*

*sum=sum+r;*

*n=n/10;*

*}*

*System.out.println("sum of the digits: "+sum);*

*}*

*}*

**Assignments**

//wap to find big digit in the given number

//wap to display the reverse order of the given number and check whether it is palindrome number or not

//wap to check whether the given number is Armstrong or not

153

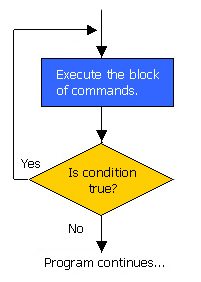
1\*1\*1+5\*5\*5+3\*3\*3

=1+125+27=153

## Java do-while loop

## Do-While Loop statement in Java : Exit Controlled Loop in Java

1. In java “**Do-while**” is iteration statements like [for loop](http://www.c4learn.com/java/java-for-loop/) and [while loop](http://www.c4learn.com/java/java-while-loop/).
2. It is also called as “**Loop Control Statement**“.
3. “**Do-while Statement**”  is **Exit Controlled Loop** because condition is check at the last moment.
4. Irrespective of the condition , control enters into the do-while loop , after completion of body execution , condition is checked whether true/false. If condition is false then it will jump out of the loop.
5. Conditional Expression written inside while must **return boolean value**.
6. In do-while loop body gets executed once whatever may be the condition but condition must be true if you need to execute body for second time.

[](http://img.c4learn.com/2012/03/do-while-loop-flowchart.jpg)

## Syntax : Do-While Loop

***do*** *{*

*Statement1;*

*Statement2;*

*Statement3;*

*. . . .*

*StatementN;*

*}* ***while*** *(expression);*

**\*Note :** Semicolon at the end of while is mandatory otherwise you will get compile error.

## Java Infinitive do-while Loop

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

**Syntax:**

***while****(****true****){*

*//code to be executed*

*}*

**Example:**

***public******class****DoWhileExample2 {*

***public******static******void****main(String[] args) {*

***do****{*

*System.out.println("infinitive do while loop");*

*}****while****(****true****);*

*}*

*}*

**Output:**

infinitive do while loop

infinitive do while loop

infinitive do while loop

## Live Example 1 : Printing Numbers

class DoWhile {

public **static** **void** main(String args[]) {

**int** n = 5;

**do**{

System.out.println("Sample : " + n);

n--;

}**while**(n > 0);

}

## Output :

*Sample : 5*

*Sample : 4*

*Sample : 3*

*Sample : 2*

*Sample : 1*

*Sample : 0*

**//wap to demo on do..while**

*import java.io.\*;*

*class DoWhile1{*

*public static void main(String args[]) throws IOException{*

*BufferedReader br = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter Any Number");*

*int n = Integer.parseInt(br.readLine());*

*int i=1;*

*do{*

*System.out.println(i);*

*i++;*

*}*

*while(i<=n);*

*}*

*}*

Rules

1. do..while loop follows same set of rules that are followed by while loop.

2. do..while loop must be terminated by ;

3. in between do block and while we should not write any statements;

***Eg****:*

*do{*

*System.out.println("hi"); X-invalid*

*}*

*System.out.println("ok");*

*while(i<=n);*

**Difference between for,while,do..while loops**

In for, while Loops we write condition while beginning of the loop where statements are executed 0 or more times

But in do..while Loop we write condition while ending of the loop where statements are executed 1 or more times

**// Wap to print power of given 2 numbers**

*import java.io.\*;*

*class DoWhile2{*

*public static void main(String args[]) throws IOException{*

*BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter N Number");*

*int n=Integer.parseInt(kbd.readLine());*

*System.out.println("Enter Power value");*

*int p=Integer.parseInt(kbd.readLine());*

*int i=1;*

*int res=1;*

*do{*

*res = res \* n;*

*i++;*

*}while(i<=p);*

*System.out.println(n+ "to the power of"+p+" is "+res);*

*}*

*}*

**//wap to find the multiplication of 2 numbers without using \* operator**

import java.io.\*;

class DoWhile3{

public static void main(String args[]) throws IOException{

BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter Any 2 Numbers");

int a=Integer.parseInt(kbd.readLine());

int b=Integer.parseInt(kbd.readLine());

int i=1;

int res=0;

do{

res = res + a;

i++;

}while(i<=b);

System.out.println(" a\*b = "+res);

}

}

**//wap to find the division of 2 numbers without using /,% operators and print** quotient and remainder

**Nested loop:**  Nested loop means writing a loop inside any other loop

**// wap to print 1perfect numbers in the range of 1-1000**

class NLoop1{

public static void main(String args[]){

for(int i=1;i<=1000;i++){

int j=1;

int sum=0;

while(j<=i/2){

if(i%j==0){

sum=sum+j;

}

j++;

}

if(sum==i){

System.out.println(i);

}

}

}

}

**//wap to print following output**

1 2 3 .... 10

2 4 6 .... 20

.

.

10 20 30 .... 100

*class NestedLoopDemo{*

*public static void main(String args[]){*

*for(int i=1;i<=10;i++){*

*for(int j=1;j<=10;j++){*

*System.out.print(i\*j+" ");*

*}*

*System.out.println();*

*}*

*}*

*}*

//wap to print following design

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

*class NestedLoopDemo{*

*public static void main(String args[]){*

*for(int i=1;i<=5;i++){*

*for(int j=1;j<=5;j++){*

*System.out.print("\*");*

*}*

*System.out.println();*

*}*

*}*

*}*

**// wap to print following design**

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

*class NestedLoopDemo{*

*public static void main(String args[]){*

*for(int i=1;i<=5;i++){*

*for(int j=1;j<=i;j++){*

*System.out.print("\*");*

*}*

*System.out.println();*

*}*

*}*

*}*

**//wap to print following design**

1

22

333 //System.out.print(i);

4444

55555

**//wap to print following design**

1

12

123 //System.out.print(j);

1234

12345

**//wap to print following design**

5

55

555 //System.out.print("5");

5555

55555

**//wap to print following designs**

1 1

2 3 0 1

4 5 6 1 0 1

7 8 9 10 0 1 0 1

11 12 13 14 15 1 0 1 0 1

**//wap to print following shape**

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

class NestedLoopDemo{

public static void main(String args[]){

int sp=15;

for(int i=1;i<=5;i++){

for(int k=1;k<=sp;k++){

System.out.print(" ");

}

for(int j=1;j<=i;j++){

System.out.print("\*");

}

System.out.println();

sp--;

}

}

}

**//wap to print following shape**

\*

\*\*\*

\*\*\*\*\* // for(int i=1;i<=22;i=i+2)

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

**//wap to print following shape**

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*

\*

## Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Java supports the following control statements. Click the following links to check their detail.

|  |  |
| --- | --- |
| **Control Statement** | **Description** |
| [**break statement**](http://www.tutorialspoint.com/java/java_break_statement.htm) | Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch. |
| [**continue statement**](http://www.tutorialspoint.com/java/java_continue_statement.htm) | Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |

## Java break statement

## Break Statement in Java Programming Language :

1. Break statement in Java Programming is used to come out of the loop control statements.
2. Break Statement is used to break : [for Loop](http://www.c4learn.com/java/java-for-loop/) | [while Loop](http://www.c4learn.com/java/java-while-loop/) | [do-while loop](http://www.c4learn.com/java/java-do-while-loop/)
3. Break Statements skips remaining statements and execute immediate statement after loop.
4. Break statement is used whenever **our condition is satisfied inside loop**and we have to come outside the loop.
5. Break statement is **used to make looping statements more flexible** and provides more power to it.

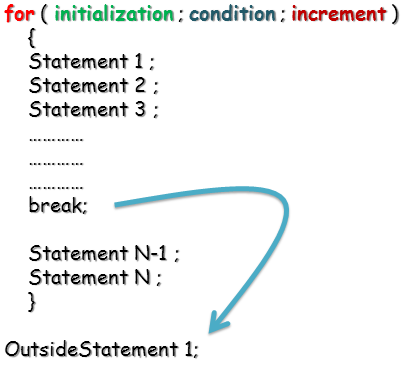
## Syntax : Break Statement In Java Programming Language

***break****;*

## Different Ways of Using Break Statement in Java :

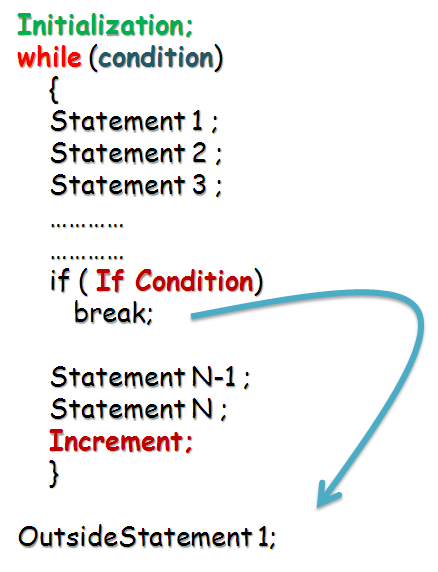
### 1.Break Statement used in For Loop

* Break statement **will take control out of the loop**.
* All the statements will gets executed i.e statement 1 , statement 2 …. when it encounters **break statement it will break loop** and **take control outside the loop.**

[](http://img.c4learn.com/2012/03/break-statement-in-java-programming-language.png)

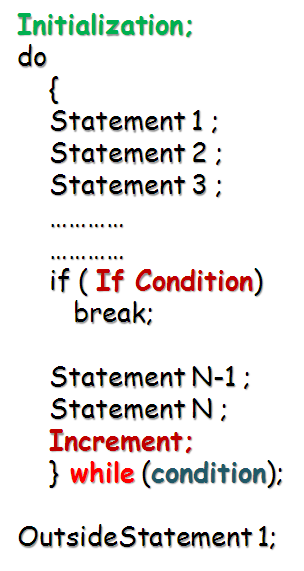
* In the above diagram break statement will take control outside the loop . i.e, it will execute **OutsideStatement1** next to break.

### 2.Break Statement used in While Loop

[](http://img.c4learn.com/2012/03/break-statement-in-while-loop-java-programming-language.png)

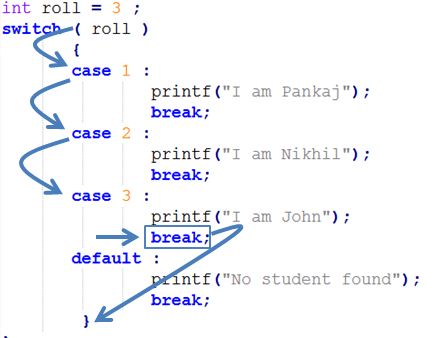
* In this example we have written break statement inside while loop.
* Whenever this statement executes then break will move control outside the while loop.

### 3.Break Statement used in While Loop

[](http://img.c4learn.com/2012/03/break-statement-in-do-while-loop-java-programming-language.png)

* In do while it executes  similarly as that of for and while loop.
* break statement terminates do while loop.

### 4.Break Statement can also be used in Switch Case

[](http://www.c4learn.com/java/java-switch-case/)

## Java Break Statement with Inner Loop

It breaks inner loop only if you use break statement inside the inner loop.

**Example:**

**public** **class** BreakExample2 {

**public** **static** **void** main(String[] args) {

**for**(**int** i=1;i<=3;i++){

**for**(**int** j=1;j<=3;j++){

**if**(i==2&&j==2){

**break**;

                        }

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

                    }

            }

}

}

**Output:**

1 1

1 2

1 3

2 1

3 1

3 2

3 3

Break statement used inside any loop to stop the continuity of the particular loop and transfer the control from inside the loop to outside the loop by skipping the remaining iterations of the loop.

**//WAP TO demo on break statement**

*class BreakDemo{*

*public static void main(String args[]){*

*for(int i=1;i<=10000;i++){*

*if(i>10){*

*break;*

*}*

*System.out.println(i);*

*}*

*System.out.println("HI");*

*}*

*}*

**Note:**

It is always recommended to write break statement using if statement

## Java continue statement

## Continue Statement in Java programming : Skip Part of Loop

Continue Statement in Java is used to skip the part of loop. Unlike break statement it does not terminate the loop , instead it skips the remaining part of the loop and control again goes to check the condition again.

## Syntax : Continue Statement

*{*

*//loop body*

*----------*

*-----------*

***continue****;*

*-----------*

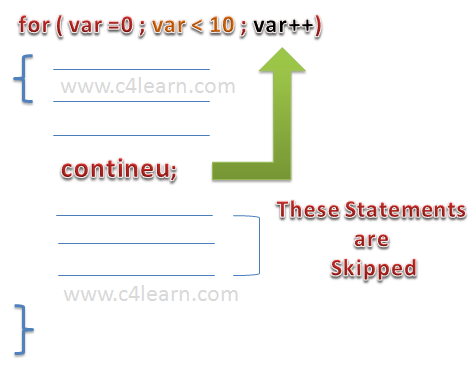
*}*

## Explanation :

1. Continue Statement is **Jumping Statement in Java Programming** like break.
2. Continue Statement skips the Loop and **Re-Executes Loop with new condition**.
3. Continue Statement **can be used only in Loop Control Statements** such as [For Loop](http://www.c4learn.com/java/java-for-loop/) | [While Loop](http://www.c4learn.com/java/java-while-loop/) | [do-While Loop](http://www.c4learn.com/java/java-do-while-loop/).
4. **Continue** is Keyword in Java Programming.

## Different Ways of Writing Continue Statement in Java

### Way 1 : Continue Statement Written Inside For Loop

[](http://img.c4learn.com/2012/03/continue-statement-in-java-Programming-skipping-loop.png)

1. Continue Statement is used to **skip the remaining part of the for loop**

***for*** *(****int*** *i = 0; i < max; i++) {*

*// Interested only in p's*

***if*** *(searchMe.charAt(i) != 'p')*

***continue****;*

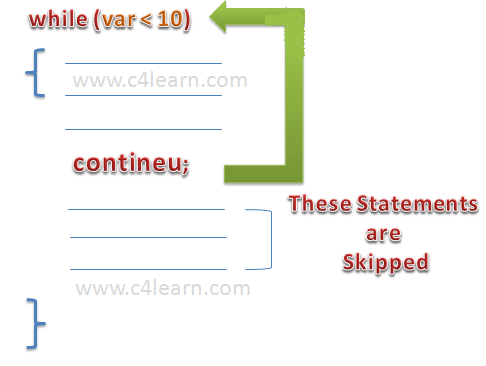
*numPs++; //Process P's*

*}*

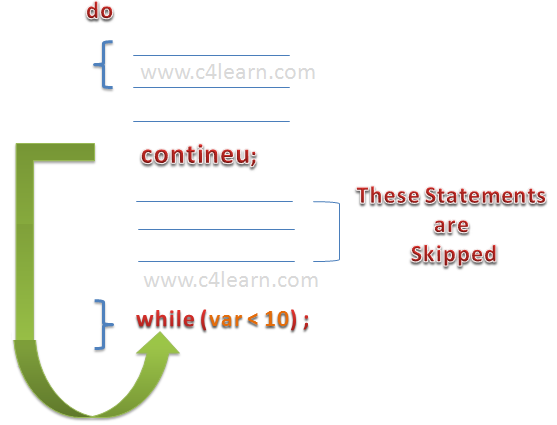
As shown in the above code snippet we can see , if we **get any character other than p then we are using continue statement to skip the loop**and if we get ‘P’ then we are incrementing Counter.

1. Continue Statement in for **loop takes control to “Increment” step**. i.e after continue “**i++**” will gets executed. (show in diagram)

### Way 2 : Continue Statement Written Inside While Loop

[](http://img.c4learn.com/2012/03/Continue-Statement-in-While-Loop.png)  
**\*Note :** It will executes condition in while loop

### Way 3 : Continue Statement Written Inside Do-While Loop

[](http://img.c4learn.com/2012/03/Continue-Statement-in-do-While-Loop.png)

## Java Continue Statement with Inner Loop

It continues inner loop only if you use continue statement inside the inner loop.

**Example:**

**public** **class** ContinueExample2 {

**public** **static** **void** main(String[] args) {

**for**(**int** i=1;i<=3;i++){

**for**(**int** j=1;j<=3;j++){

**if**(i==2&&j==2){

**continue**;

                        }

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

                    }

            }

}

}

Output:

1 1

1 2

1 3

2 1

2 3

3 1

3 2

3 3

## Live Example : Continue Statement in Java Programming

In this example we are going to count the number of occurrences of “w”

*class ContinueDemo {*

*public* ***static******void*** *main(String[] args) {*

*String str = "www.c4learn.com";*

***int*** *max = str.length();*

***int*** *count = 0;*

***for*** *(****int*** *i = 0; i < max; i++) {*

***if*** *(str.charAt(i) != 'w')*

***continue****;*

*count++;*

*}*

*System.out.println("Counting W : " + count );*

*}*

*}*

## Output :

Counting W : 3

* count statement is executed only 3 times as **there are only 3 w’s present in string str**.

## Illegal Way of Writing Continue :

### Way 1 : Continue as a Part of Normal Java Statement

*class ContinueDemo {*

*public* ***static******void*** *main(String[] args) {*

*String str = "www.c4learn.com";*

***int*** *max = str.length();*

***int*** *count = 0;*

***continue****;*

*System.out.println("Hello continue");*

*}*

*}*

Above Statement will cause Compile time Error.

### Way 2 : Don’t Use Continue Inside If

1. Continue **can be used inside If statement** iff “If Statement” is part of any Loop.
2. **Don’t Use Continue inside if which is not a part of loop.**

class ContinueDemo {

public **static** **void** main(String[] args) {

String str = "www.c4learn.com";

**int** max = str.length();

**int** count = 0;

**if**(str.charAt(i) != 'p')

**continue**;

System.out.println("Hello continue");

}

}

## Enhanced for loop in Java:

As of Java 5, the enhanced for loop was introduced. This is mainly used to traverse collection of elements including arrays.

## Syntax:

The syntax of enhanced for loop is:

*for(declaration : expression){*

*//Statements*

*}*

**Declaration:** The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.

**Expression:** This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

## Example:

*public class Test {*

*public static void main(String args[]){*

*int [] numbers = {10, 20, 30, 40, 50};*

*for(int x : numbers ){*

*System.out.print( x );*

*System.out.print(",");*

*}*

*System.out.print("\n");*

*String [] names ={"James", "Larry", "Tom", "Lacy"};*

*for( String name : names ) {*

*System.out.print( name );*

*System.out.print(",");*

*}*

*}*

*}*

This would produce the following result:

10,20,30,40,50,

James,Larry,Tom,Lacy,

**Syntax Of for-each Loop :**

The syntax of for-each loop is as follows

|  |  |
| --- | --- |
|  | *for(Data\_Type variable : array or collection){*  *}* |

Where Data\_Type specifies type and variable specifies iteration variable.

**How for-each loop works :**

The iteration variable in the for-each loop receives every element of an array or collection one at a time starting from first element to last element. i.e In the first iteration, it gets the first element. In the second iteration, it gets the second element and so on. Thus it iterates all elements of an array or the collection. The type of iteration variable must be compatible with the type of array or collection.

**Example of for-each loop which iterates an array :**

|  |  |
| --- | --- |
|  | *public class ForEachLoop{*  *public static void main(String[] args) {*  *//An array of strings*  *String[] str = {"First", "Second", "Third", "Fourth", "Fifth"};*  *//iterating every element of str using for-each loop*  *for (String s : str)  {*  *System.out.println(s);*  *}*  *}*  *}* |

**Output :**  
First  
Second  
Third  
Fourth  
Fifth

Example of for-each loop which iterates the collection :

|  |  |
| --- | --- |
|  | public class ForEachLoop{      public static void main(String[] args)    {          //An ArrayList of strings          ArrayList<String> list = new ArrayList<String>();          //Adding elements to ArrayList          list.add("First");          list.add("Second");          list.add("Third");          list.add("Fourth");           //iterating every element of list using for-each loop          for (String s : list)        {              System.out.println(s);          }      }  } |

**Output :**  
First  
Second  
Third  
Fourth

**Nested for-each Loop :**

for-each loop can be nested like normal for loop. Here is the example for Nested for-each loop which iterates two dimensional array.

|  |  |
| --- | --- |
|  | public class ForEachLoop{      public static void main(String[] args)    {          //Two Dimensional Array          int[][] twoDArray = { {1, 2, 3, 4}, {5, 6, 7, 8} };           //iterating every element of twoDArray using for-each loop          for (int[] oneDArray : twoDArray){              System.out.print("[");              //iterating every element of oneDArray using for-each loop              for (int i : oneDArray)            {                  System.out.print(i+"\t");              }              System.out.println("]");          }      }  } |
|  |  |

**Output :**  
[1 2 3 4 ]  
[5 6 7 8 ]

**Advantages of for-each loop :**

You need not to specify the initialization, condition and increment or decrement as you specify in the normal for loop.

It increases the readability of the code.

**Disadvantages Of for-each loop :**

You can’t iterate only few elements of an array or collection using for-each loop.

**//wap to demo on continue statement**

class ContinueDemo{

public static void main(String args[]){

for(int i=1;i<=15;i++){

if(i==2||i==7||i==11){

continue;

}

System.out.println(i);

}

}

}

**Note:**

It is allways recommended to write continue statement using if statement.

**3.return**

return is a transferring statement which is used to stop the continuity of method execution.

**//wap to demo on return statement**

class ReturnDemo{

public static void main(String args[]){

System.out.println("Hi");

if(10>2){

return;

}

System.out.println("Bye!!!");

}

}

**Note:**

It is allways recommended to write return statement using if statement.

## 1) [Fibonacci series](http://www.javatpoint.com/fibonacci-series-in-java)

Write a java program to print fibonacci series without using recursion and using recursion.

**Input:** 10

**Output:** 0 1 1 2 3 5 8 13 21 34

## 2) [Prime number](http://www.javatpoint.com/prime-number-program-in-java)

Write a java program to check prime number.

**Input:** 44

**Output:** not prime number

**Input:** 7

**Output:** prime number

## 3) [Palindrome number](http://www.javatpoint.com/palindrome-program-in-java)

Write a java program to check palindrome number.

**Input:** 329

**Output:** not palindrome number

**Input:** 12321

**Output:** palindrome number

## 4) [Factorial number](http://www.javatpoint.com/factorial-program-in-java)

Write a java program to print factorial of a number.

**Input:** 5

**Output:** 120

**Input:** 6

**Output:** 720

## 5) [Armstrong number](http://www.javatpoint.com/armstrong-number-in-java)

Write a java program to check Armstrong number.

**Input:** 153

**Output:** Armstrong number

**Input:** 22

**Output:** not Armstrong number

## 6) [Bubble Sort](http://www.javatpoint.com/bubble-sort-in-java)

Write a java program to sort an array elements using bubble sort algorithm.

**Input:** 18 9 33 4 84 32

**Output:** 4 9 18 32 33 84

## 7) [Selection Sort](http://www.javatpoint.com/selection-sort-in-java)

Write a java program to sort an array elements using selection sort algorithm.

**Input:** 18 9 33 4 84 32

**Output:** 4 9 18 32 33 84

## 8) [Insertion Sort](http://www.javatpoint.com/insertion-sort-in-java)

Write a java program to sort an array elements using insertion sort algorithm.

**Input:** 18 9 33 4 84 32

**Output:** 4 9 18 32 33 84

# Fibonacci series in Java

In fibonacci series, next number is the sum of previous two numbers for example 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 etc. The first two numbers of fibonacci series are 0 and 1.

**There are two ways to write the fibonacci series program in java:**

* **Fibonacci Series without using recursion**
* **Fibonacci Series using recursion**

## Fibonacci Series in Java without using recursion

Let's see the fibonacci series program in java without using recursion.

***class****FibonacciExample1{*

***public******static******void****main(String args[])  {*

***int****n1=0,n2=1,n3,i,count=10;*

*System.out.print(n1+" "+n2);//printing 0 and 1*

***for****(i=2;i<count;++i)*

*//loop starts from 2 because 0 and 1 are already printed*

*{*

*n3=n1+n2;*

*System.out.print(" "+n3);*

*n1=n2;*

*n2=n3;*

*}*

*}*

*}*

**Output:**

0 1 1 2 3 5 8 13 21 34

## Fibonacci Series using recursion in java

Let's see the fibonacci series program in java using recursion.

***class****FibonacciExample2{*

***static******int****n1=0,n2=1,n3=0;*

***static******void****printFibonacci(****int****count){*

***if****(count>0){*

*n3 = n1 + n2;*

*n1 = n2;*

*n2 = n3;*

*System.out.print(" "+n3);*

*printFibonacci(count-1);*

*}*

*}*

***public******static******void****main(String args[]){*

***int****count=10;*

*System.out.print(n1+" "+n2);//printing 0 and 1*

*printFibonacci(count-2);*

*//n-2 because 2 numbers are already printed*

*}*

*}*

**Output:**

0 1 1 2 3 5 8 13 21 34

# Prime Number Program in Java

Prime number in Java: **Prime number** is a number that is greater than 1 and divided by 1 or itself. In other words, prime numbers can't be divided by other numbers than itself or 1. For example 2, 3, 5, 7, 11, 13, 17.... are the prime numbers.

#### Note: 0 and 1 are not prime numbers. The 2 is the only even prime number because all the other even numbers can be divided by 2.

Let's see the prime number program in java. In this java program, we will take a number variable and check whether the number is prime or not.

**class** PrimeExample{

**public** **static** **void** main(String args[]){

**int** i,m=0,flag=0;

**int** n=17;//it is the number to be checked

  m=n/2;

**for**(i=2;i<=m;i++){

**if**(n%i==0){

   System.out.println("Number is not prime");

   flag=1;

**break**;

   }

  }

**if**(flag==0)

  System.out.println("Number is prime");

}

}

**Output:**

Number is prime

# Palindrome Program in Java

Palindrome number in java: A **palindrome number** is a number that is same after reverse. For example 545, 151, 34543, 343, 171, 48984 are the palindrome numbers.

## Palindrome number algorithm

* Get the number to check for palindrome
* Hold the number in temporary variable
* Reverse the number
* Compare the temporary number with reversed number
* If both numbers are same, print "palindrome number"
* Else print "not palindrome number"

Let's see the palindrome program in java. In this java program, we will get a number variable and check whether number is palindrome or not.

**class** PalindromeExample{

**public** **static** **void** main(String args[]){

**int** r,sum=0,temp;

**int** n=454;//It is the number variable to be checked for palindrome

  temp=n;

**while**(n>0){

   r=n%10;  //getting remainder

   sum=(sum\*10)+r;

   n=n/10;

  }

**if**(temp==sum)

   System.out.println("palindrome number ");

**else**

   System.out.println("not palindrome");

}

}

**Output:**

palindrome number

Let's see the palindrome program in java. In this java program, we will get a number variable and check whether number is palindrome or not.

**class** PalindromeExample{

**public** **static** **void** main(String args[]){

**int** r,sum=0,temp;

**int** n=454;//It is the number variable to be checked for palindrome

  temp=n;

**while**(n>0){

   r=n%10;  //getting remainder

   sum=(sum\*10)+r;

   n=n/10;

  }

**if**(temp==sum)

    System.out.println("palindrome number ");

**else**

    System.out.println("not palindrome");

}

}

**Output:**

palindrome number

**Armstrong Number in Java**

Armstrong Number in Java: **Armstrong number** is *a number that is equal to the sum of cubes of its digits* for example 0, 1, 153, 370, 371, 407 etc.

Let's try to understand why **153** is an Armstrong number.

153 = (1\*1\*1)+(5\*5\*5)+(3\*3\*3)

where:

(1\*1\*1)=1

(5\*5\*5)=125

(3\*3\*3)=27

So:

1+125+27=153

Let's try to understand why **371** is an Armstrong number.

371 = (3\*3\*3)+(7\*7\*7)+(1\*1\*1)

where:

(3\*3\*3)=27

(7\*7\*7)=343

(1\*1\*1)=1

So:

27+343+1=371

Let's see the java program to check Armstrong Number.

**class** ArmstrongExample{

**public** **static** **void** main(String[] args)  {

**int** c=0,a,temp;

**int** n=153;//It is the number to check armstrong

    temp=n;

**while**(n>0)

    {

    a=n%10;

    n=n/10;

    c=c+(a\*a\*a);

    }

**if**(temp==c)

    System.out.println("armstrong number");

**else**

        System.out.println("Not armstrong number");

   }

}

**Output:**

armstrong number

**Bubble Sort in Java**

We can create a java program to sort array elements using bubble sort. Bubble sort algorithm is known as the simplest sorting algorithm.

In bubble sort algorithm, array is traversed from first element to last element. Here, current element is compared with the next element. If current element is greater than the next element, it is swapped.

***public******class****BubbleSortExample {*

***static******void****bubbleSort(****int****[] arr) {*

***int****n = arr.length;*

***int****temp = 0;*

***for****(****int****i=0; i < n; i++){*

***for****(****int****j=1; j < (n-i); j++){*

***if****(arr[j-1] > arr[j]){*

*//swap elements*

*temp = arr[j-1];*

*arr[j-1] = arr[j];*

*arr[j] = temp;*

*}*

*}*

*}*

*}*

***public******static******void****main(String[] args) {*

***int****arr[] ={3,60,35,2,45,320,5};*

*System.out.println("Array Before Bubble Sort");*

***for****(****int****i=0; i < arr.length; i++){*

*System.out.print(arr[i] + " ");*

*}*

*System.out.println();*

*bubbleSort(arr);//sorting array elements using bubble sort*

*System.out.println("Array After Bubble Sort");*

***for****(****int****i=0; i < arr.length; i++){*

*System.out.print(arr[i] + " ");*

*}*

*}*

*}*

**Output:**

*Array Before Bubble Sort*

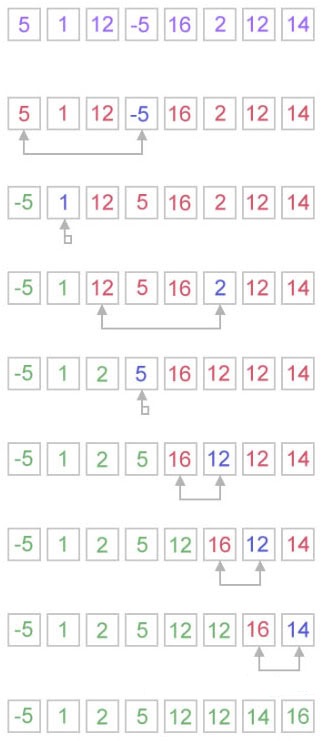
*3 60 35 2 45 320 5*

*Array After Bubble Sort*

*2 3 5 35 45 60 320*

**Selection Sort in Java**

We can create a java program to sort array elements using selection sort. In selection sort algorithm, we search for the lowest element and arrange it to the proper location. We swap the current element with the next lowest number.



***public******class****SelectionSortExample {*

***public******static******void****selectionSort(****int****[] arr){*

***for****(****int****i = 0; i < arr.length - 1; i++)   {*

***int****index = i;*

***for****(****int****j = i + 1; j < arr.length; j++){*

***if****(arr[j] < arr[index]){*

*index = j;//searching for lowest index*

*}*

*}*

***int****smallerNumber = arr[index];*

*arr[index] = arr[i];*

*arr[i] = smallerNumber;*

*}*

*}*

***public******static******void****main(String a[]){*

***int****[] arr1 = {9,14,3,2,43,11,58,22};*

*System.out.println("Before Selection Sort");*

***for****(****int****i:arr1){*

*System.out.print(i+" ");*

*}*

*System.out.println();*

*selectionSort(arr1);//sorting array using selection sort*

*System.out.println("After Selection Sort");*

***for****(****int****i:arr1){*

*System.out.print(i+" ");*

*}*

*}*

*}*

**Output:**

Before Selection Sort

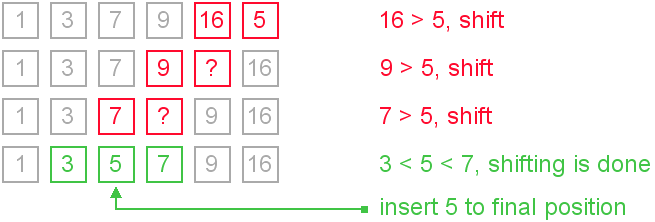
9 14 3 2 43 11 58 22

After Selection Sort

2 3 9 11 14 22 43 58

**Insertion Sort in Java**

We can create a java program to sort array elements using insertion sort. Insertion is good for small elements only because it requires more time for sorting large number of elements.



Let's see a simple java program to sort an array using insertion sort algorithm.

***public******class****InsertionSortExample {*

***public******static******void****insertionSort(****int****array[]) {*

***int****n = array.length;*

***for****(****int****j = 1; j < n; j++) {*

***int****key = array[j];*

***int****i = j-1;*

***while****( (i > -1) && ( array [i] > key ) ) {*

*array [i+1] = array [i];*

*i--;*

*}*

*array[i+1] = key;*

*}*

*}*

***public******static******void****main(String a[]){*

***int****[] arr1 = {9,14,3,2,43,11,58,22};*

*System.out.println("Before Insertion Sort");*

***for****(****int****i:arr1){*

*System.out.print(i+" ");*

*}*

*System.out.println();*

*insertionSort(arr1);//sorting array using insertion sort*

*System.out.println("After Insertion Sort");*

***for****(****int****i:arr1){*

*System.out.print(i+" ");*

*}*

*}*

*}*

**Output:**

Before Insertion Sort

9 14 3 2 43 11 58 22

After Insertion Sort

2 3 9 11 14 22 43 58

## *While loop vs For loop*

**Counting.** A for loop is preferred to a while loop when counting through a series of numbers -- in this case all character positions in a string.

**Equivalent**. A for loop has the same *condition* as the equivalent while loop, but also incorporates an*initialization*, which would be before the while statement, and the *increment*, which would be at the end of thewhile body. You can write the loop either way, but putting the initialization, condition, and increment in one statement increases the readability.

|  |  |
| --- | --- |
| For loop | While loop |
| for (**int i=0;** i<input.length(); **i++**) {  reversed = input.substring(i, i+1) + reversed;  } | **int i = 0;**  while (i<input.length()) {  reversed = input.substring(i, i+1) + reversed;  **i++;**  } |

## A single character - String or char?

This program uses substring(...) to get a single character. It would be more efficient to use charAt(...), which returns a single primitive char value.

*for (****int i=0;*** *i<input.length();* ***i++****) {*

*reversed = input.****charAt(i)*** *+ reversed;*

*}*

*public class Reverse {*

*public static void main(String[] args) {*

*String input; // Used for the input string.*

*String reversed; // Reversed form or the input string.*

*while (true) {*

*input = JOptionPane.showInputDialog(null, "Enter a string");*

*if (input == null) break;*

*reversed = "";*

*for (int i=0; i<input.length(); i++) {*

*reversed = input.substring(i, i+1) + reversed;*

*}*

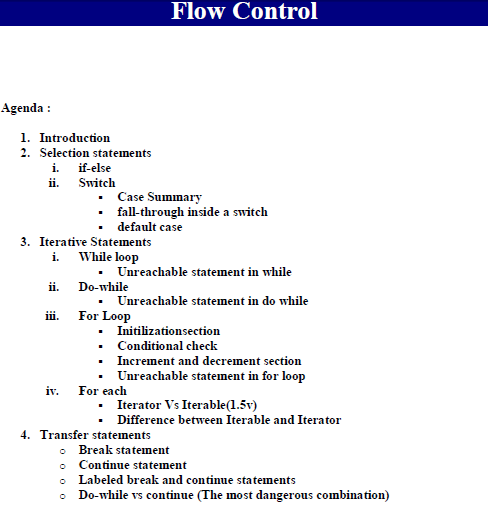
*JOptionPane.showMessageDialog(null, "Reversed:\n" + reversed);*

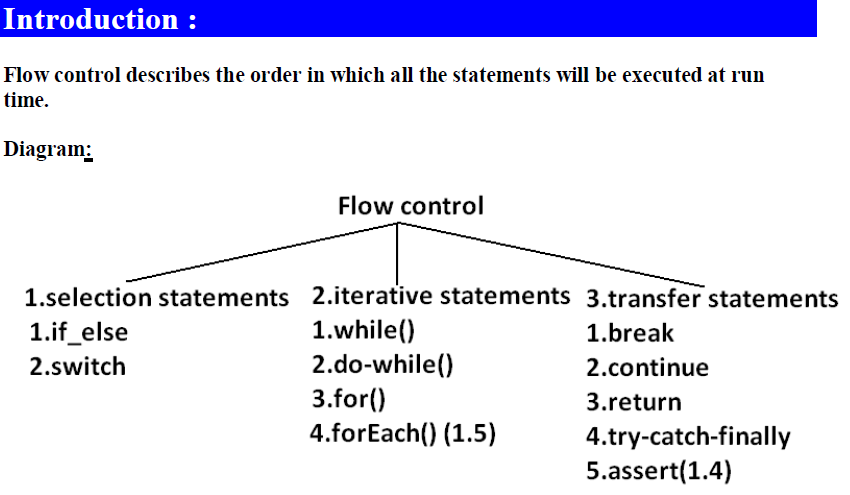
*}*

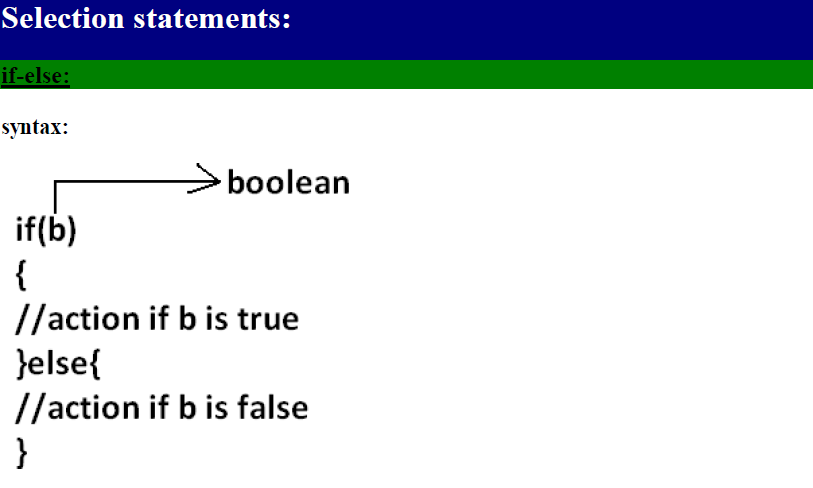
*}*

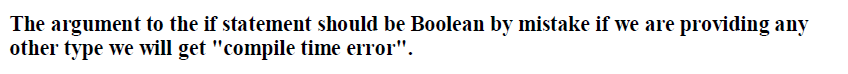
*}*

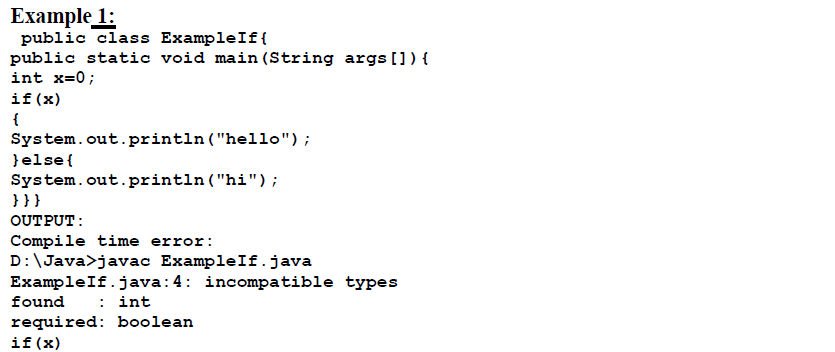
***SCJP:***

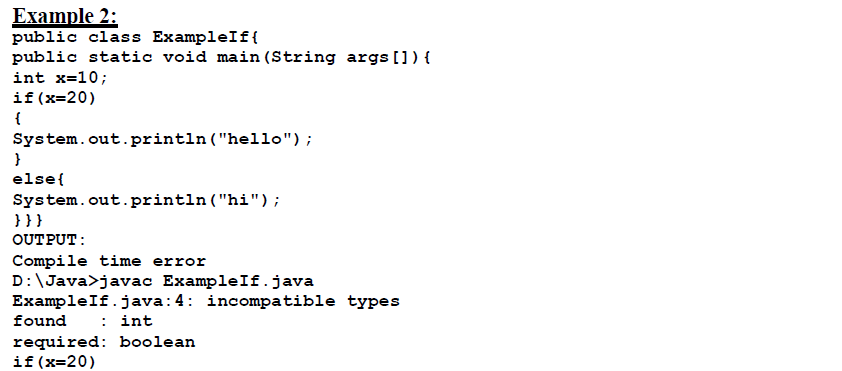


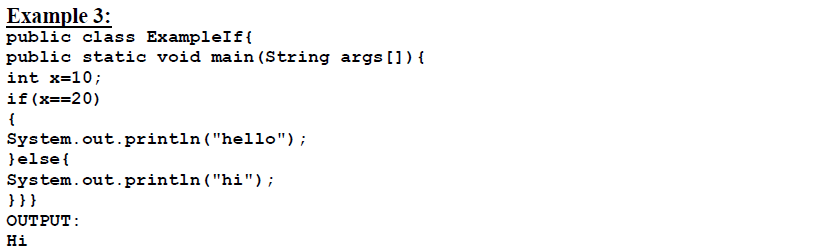


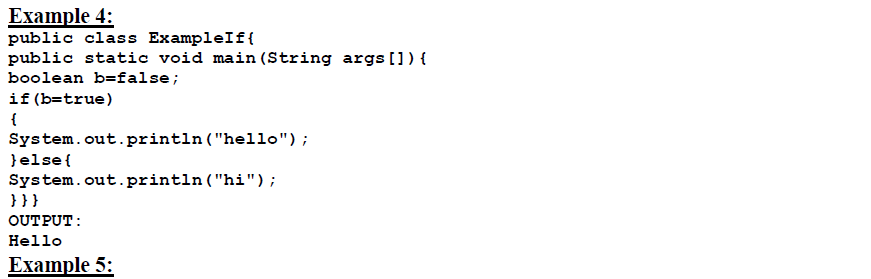


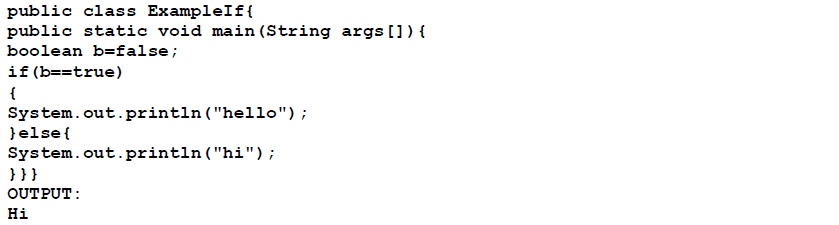


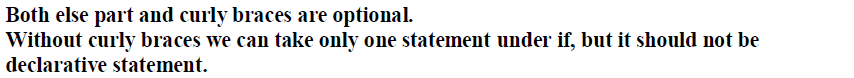


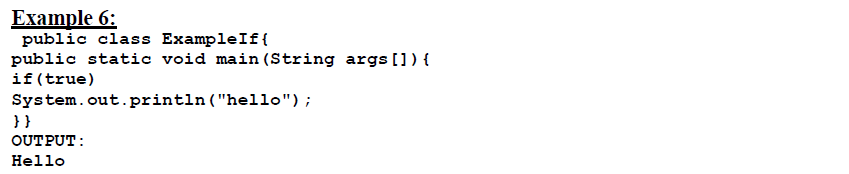


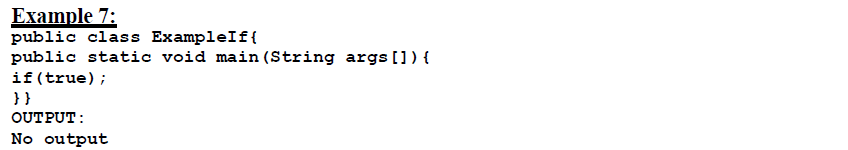


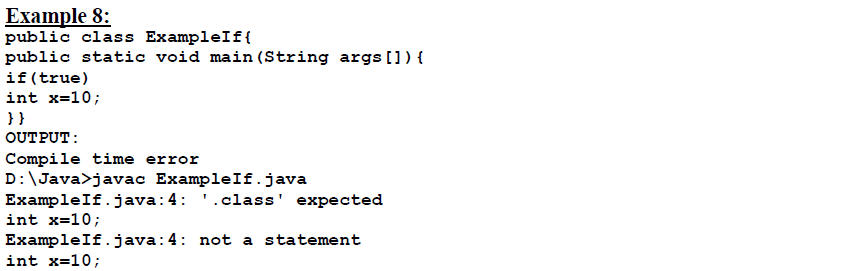


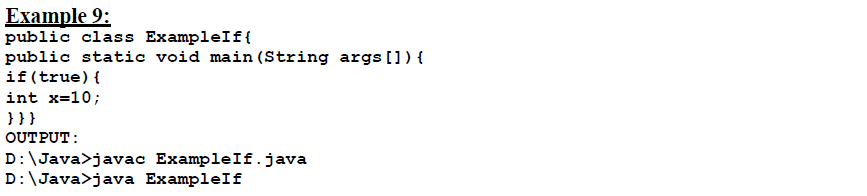


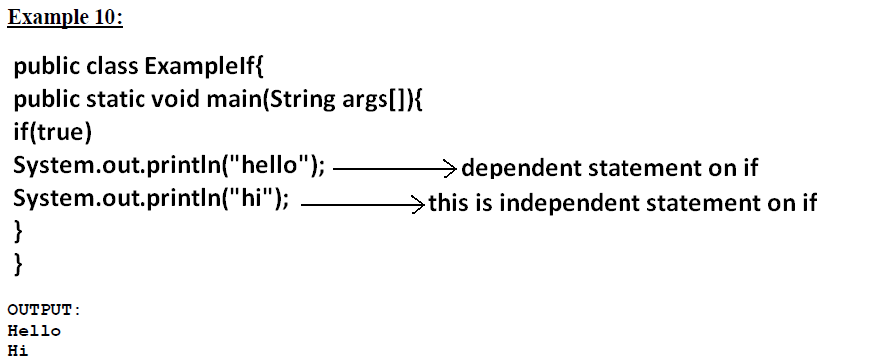








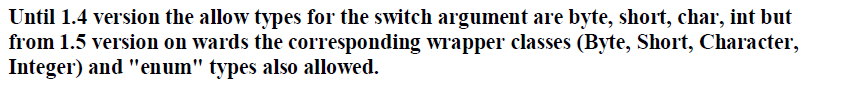


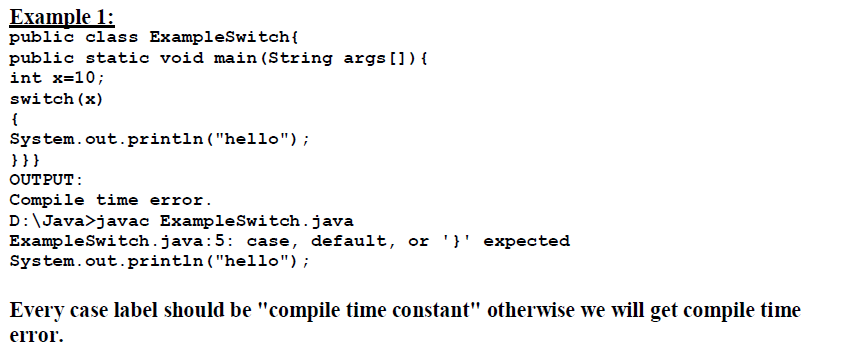
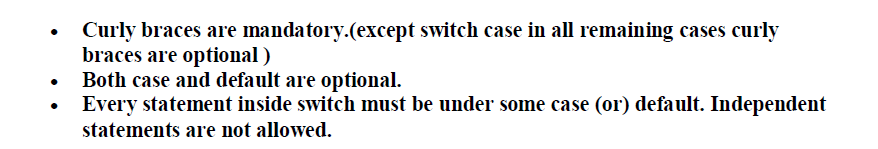
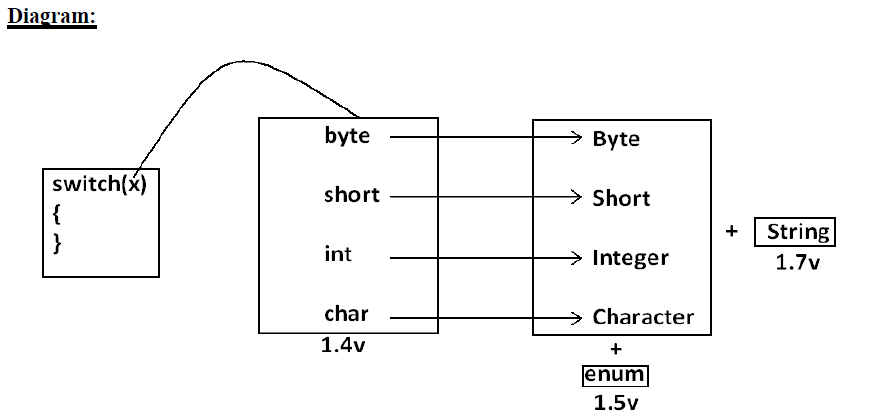


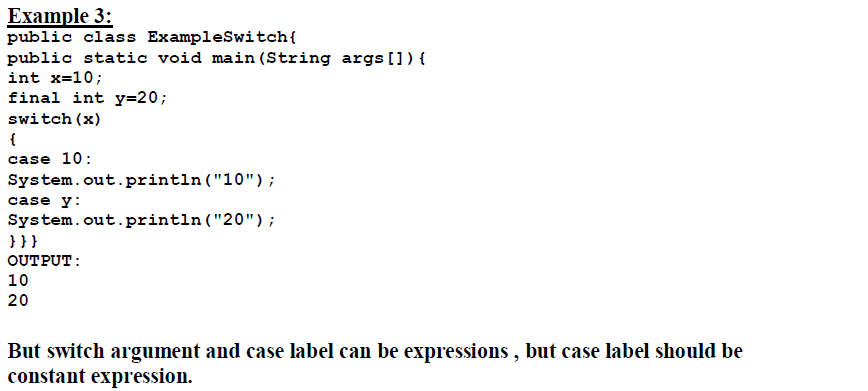
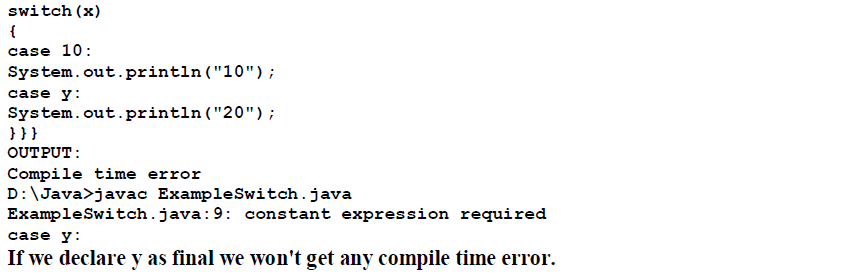
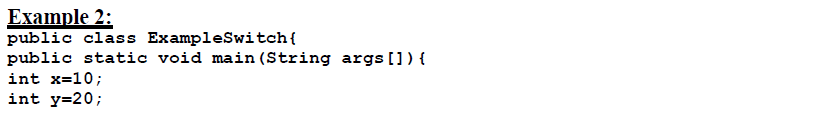


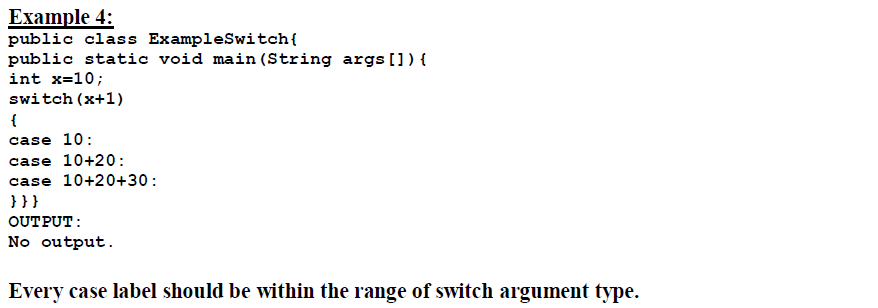


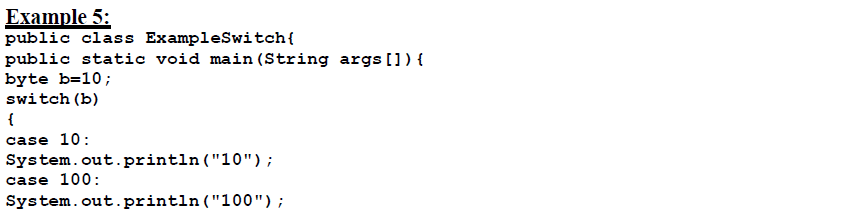


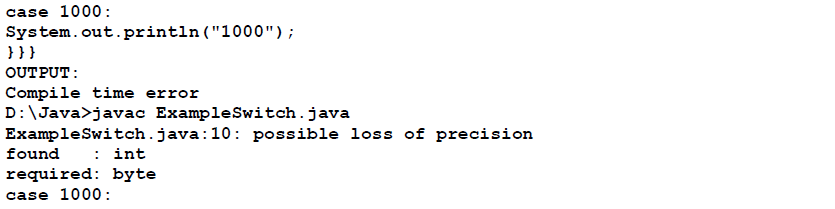


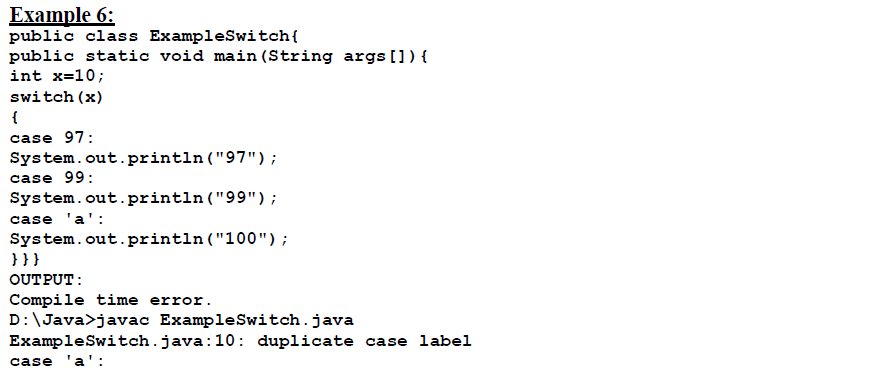
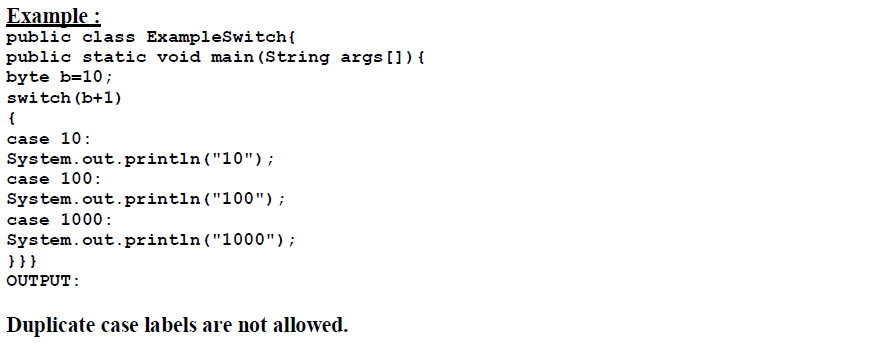


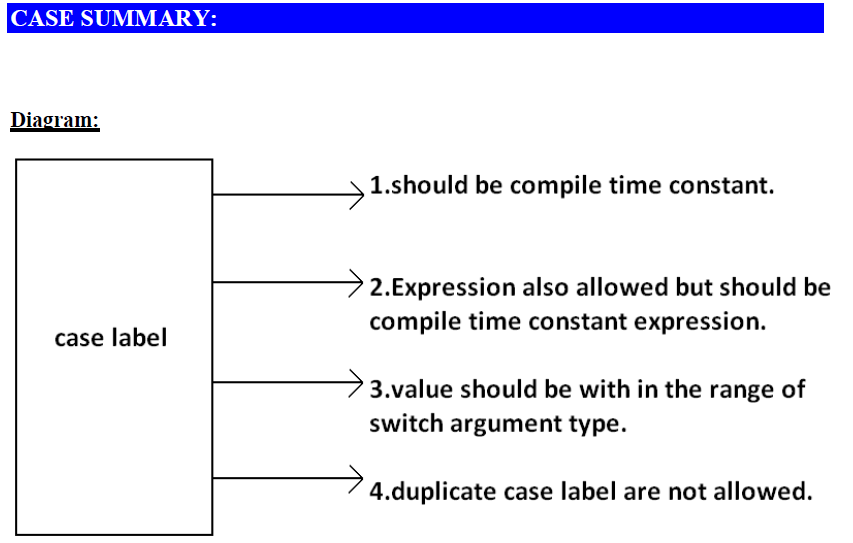


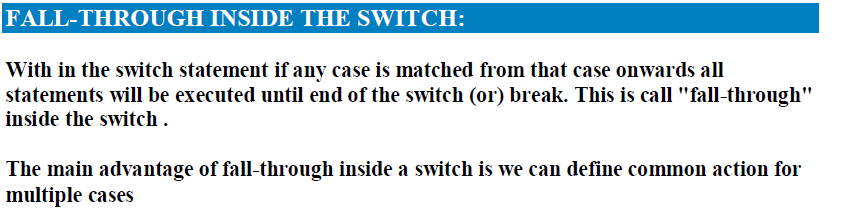


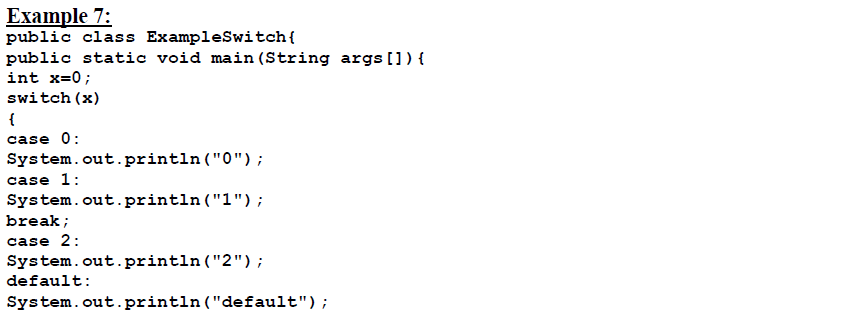


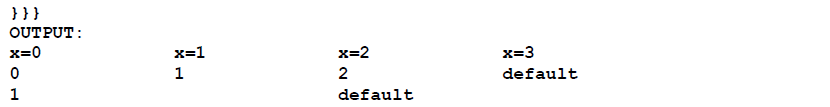


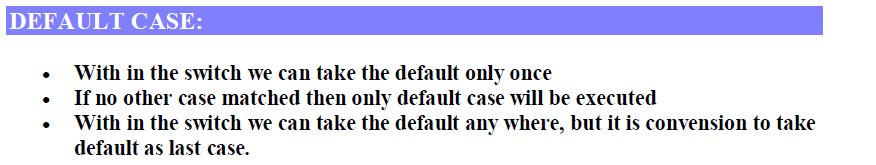


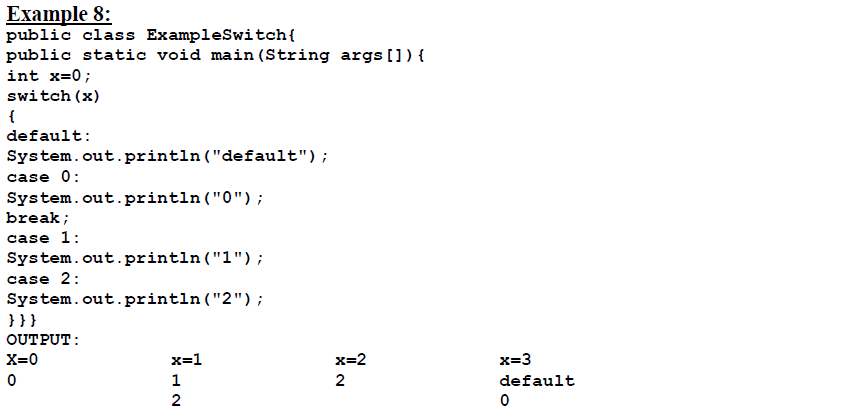


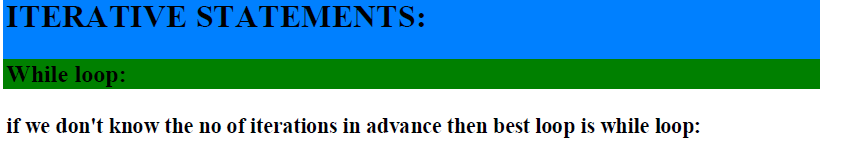


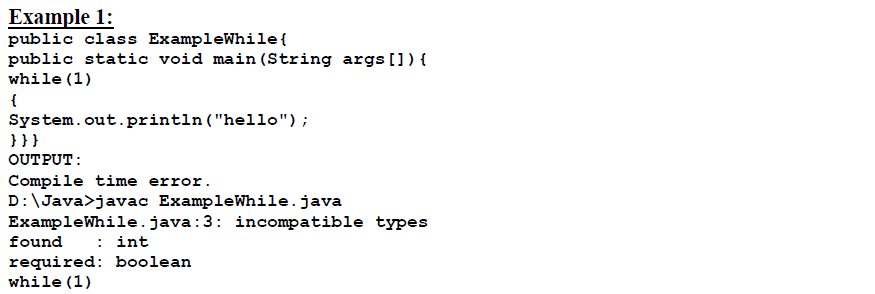
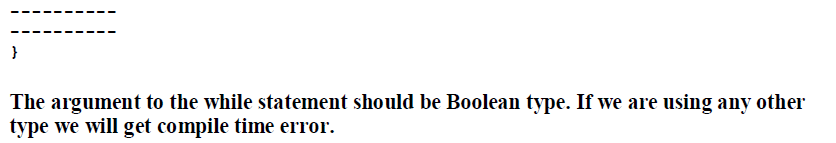


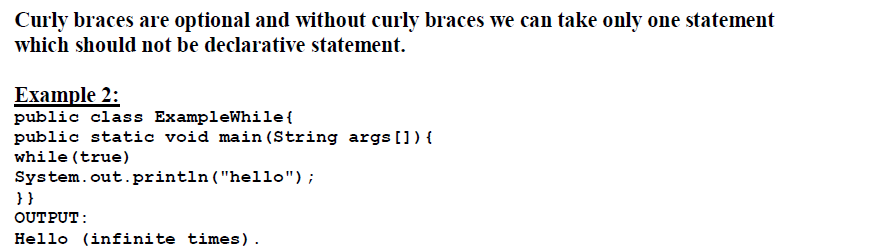


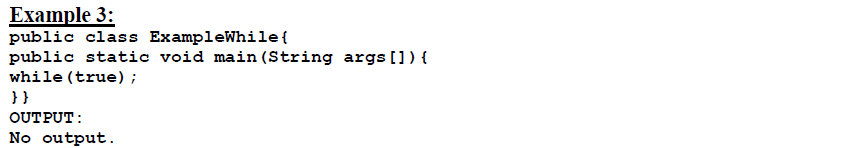


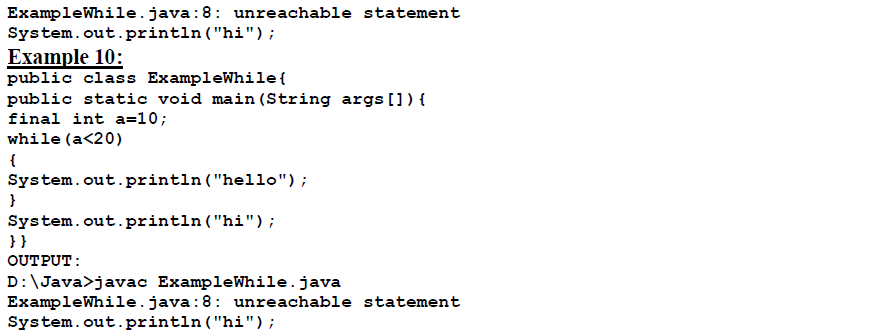
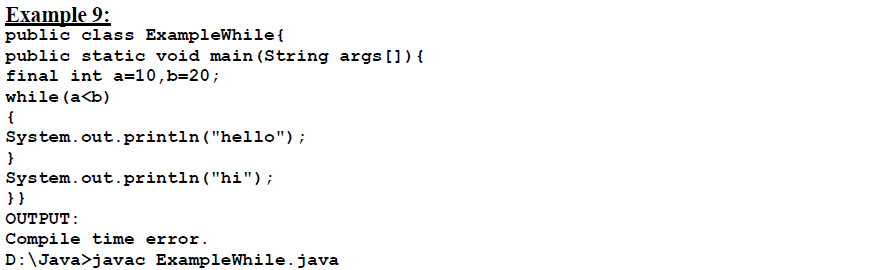
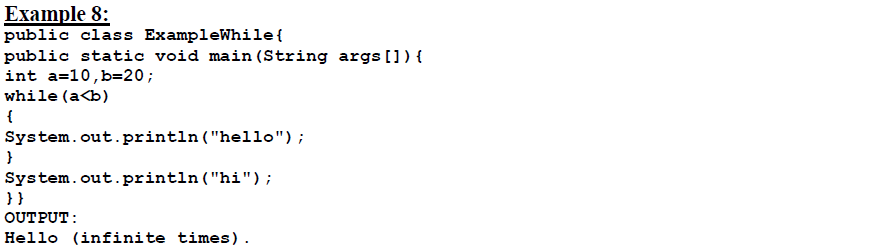
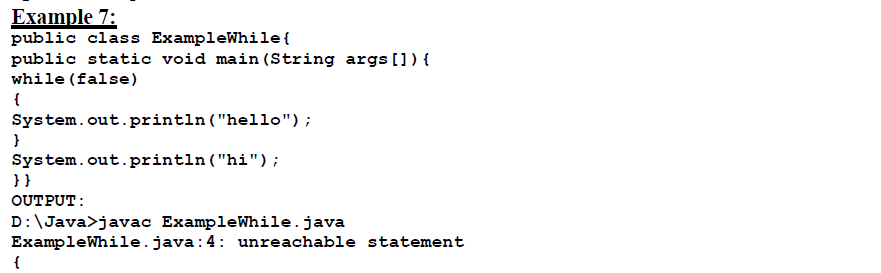
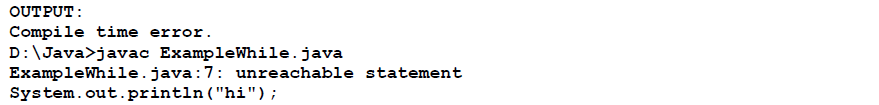
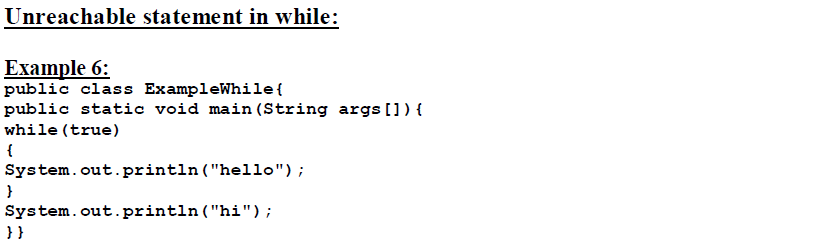
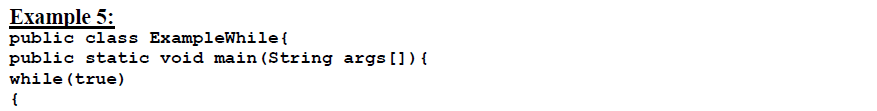
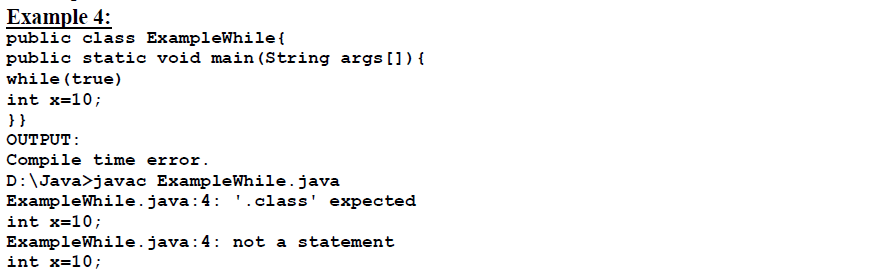


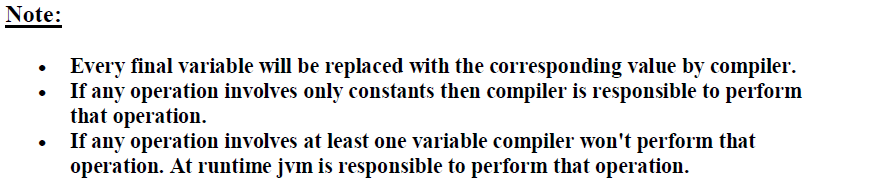


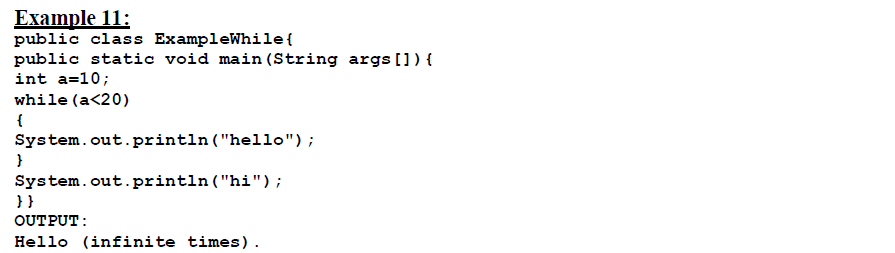


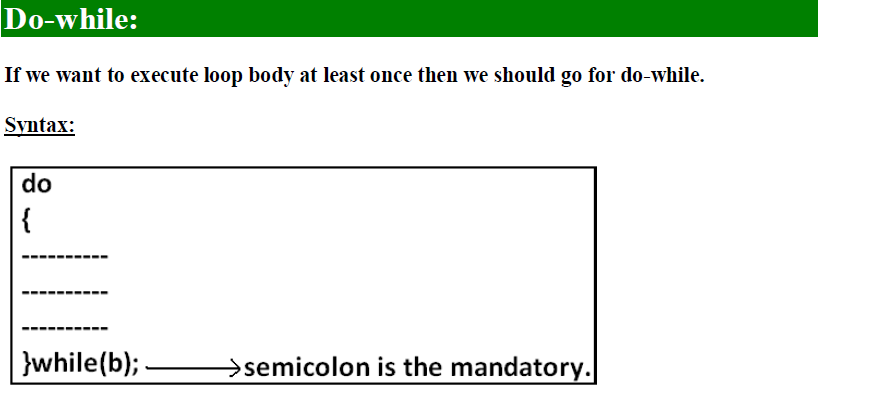


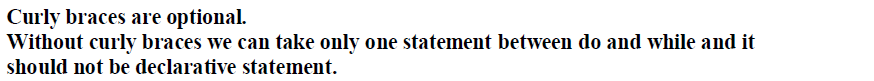


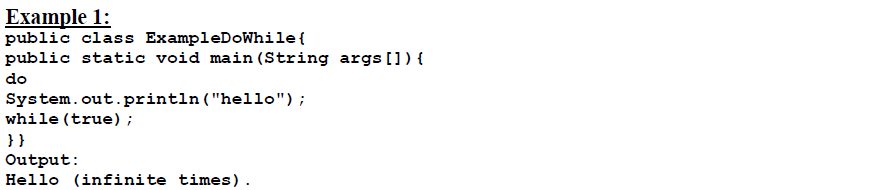


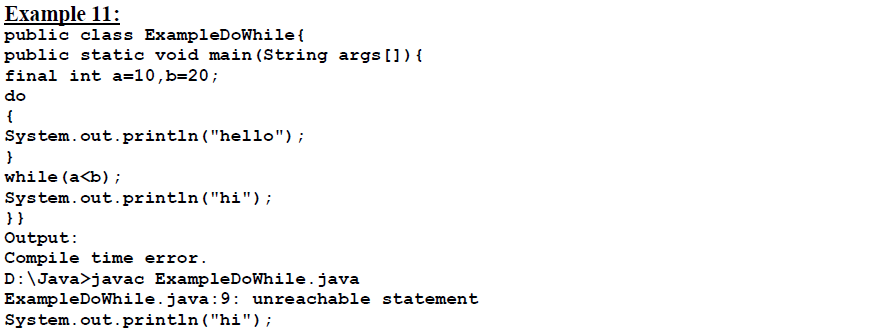
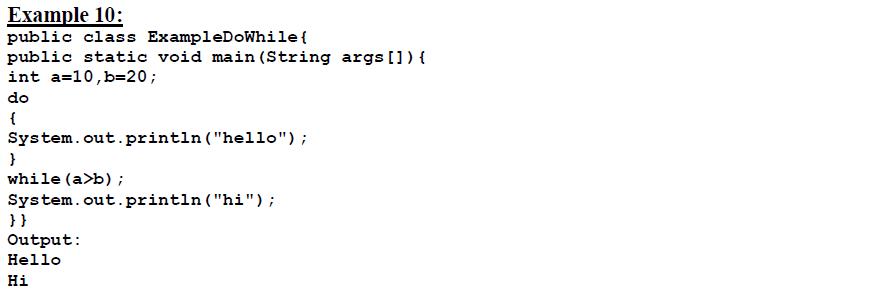
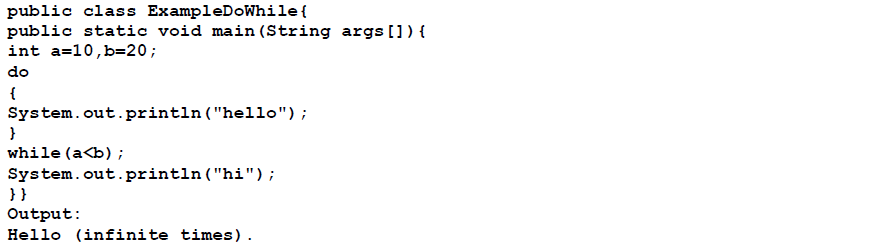
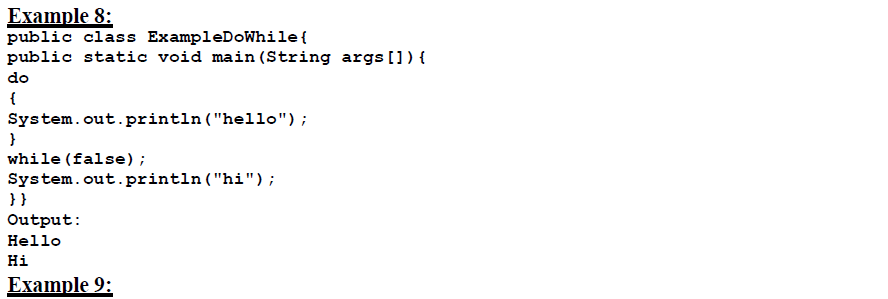
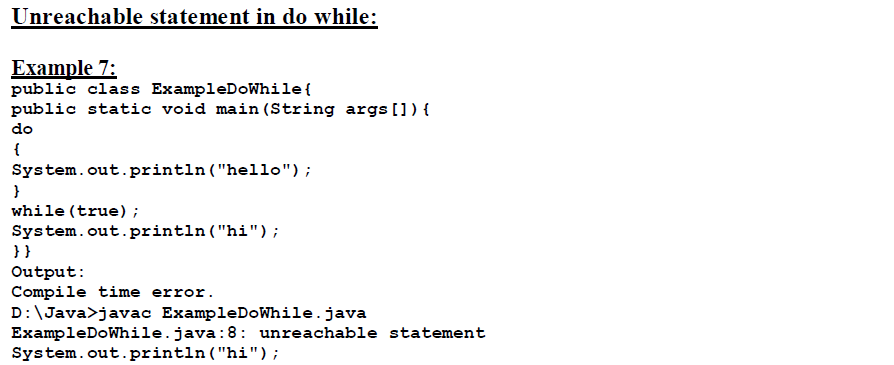
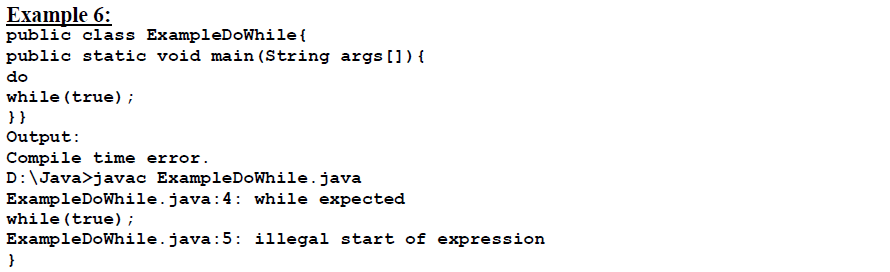
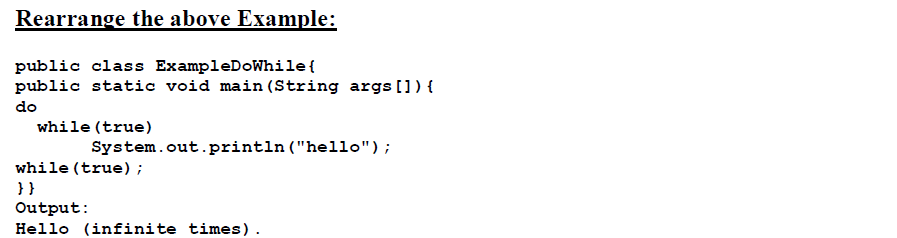
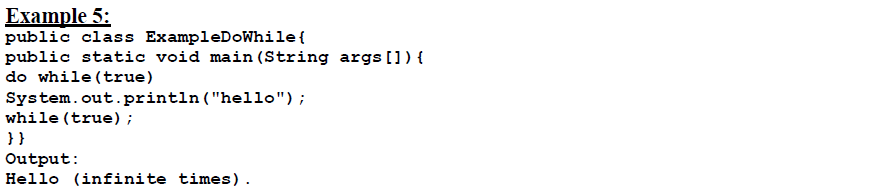
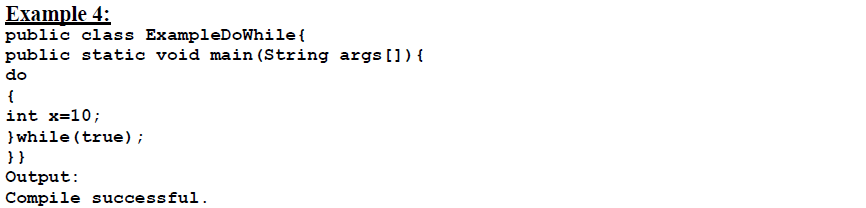
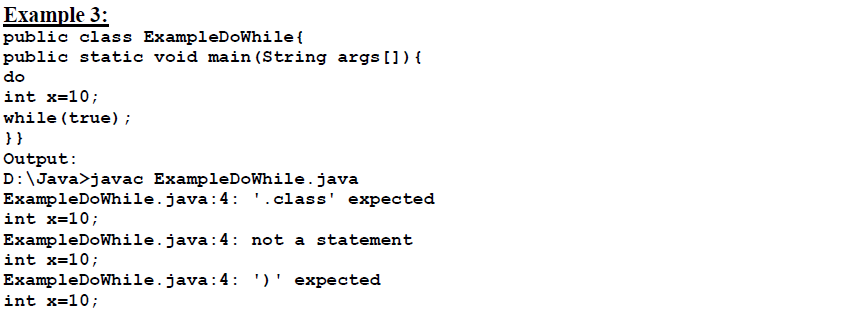
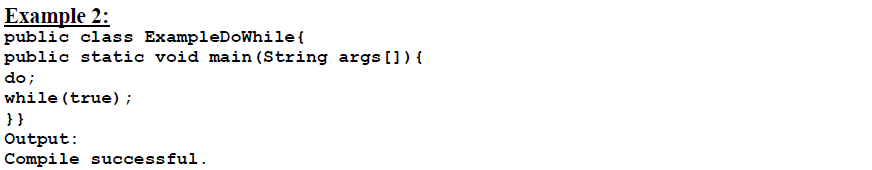


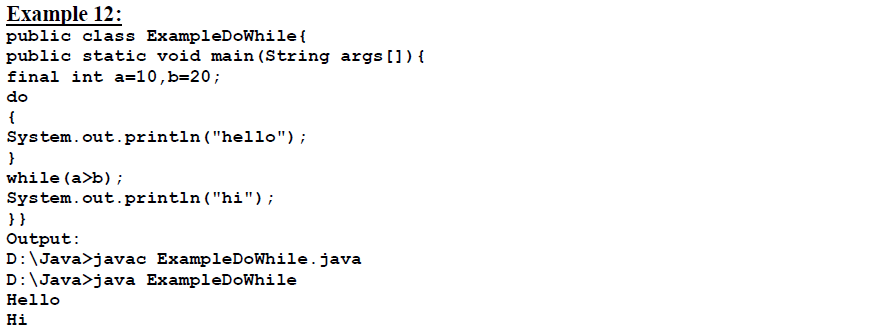


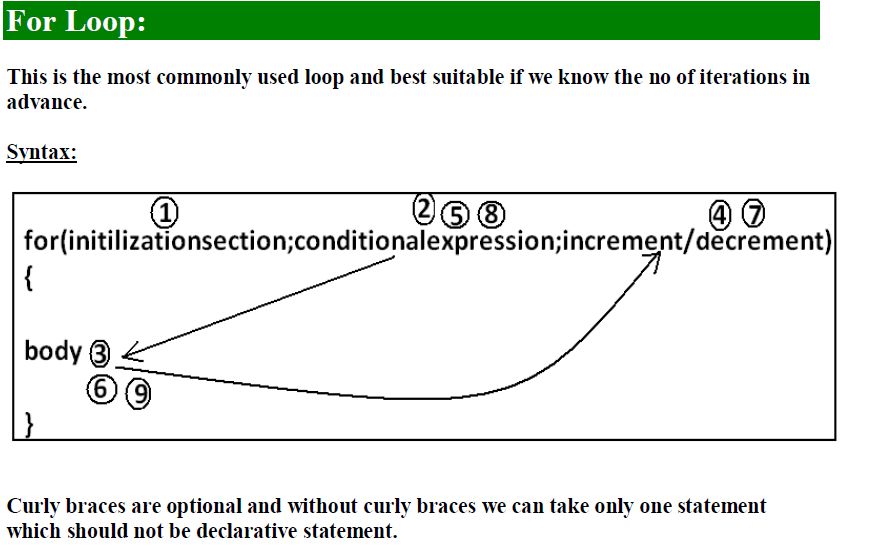


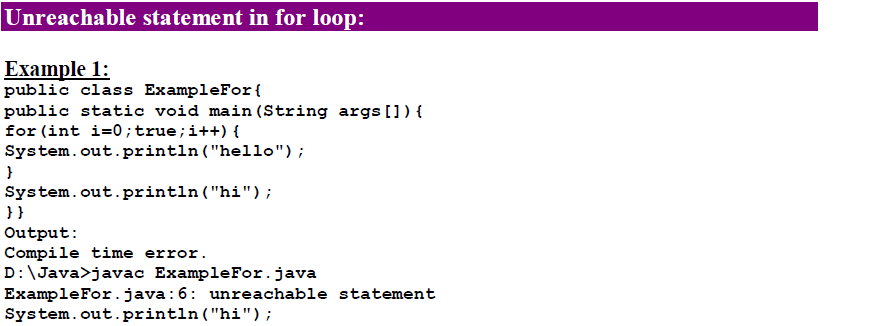
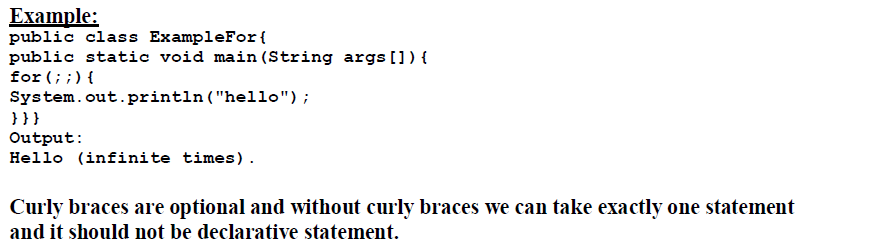
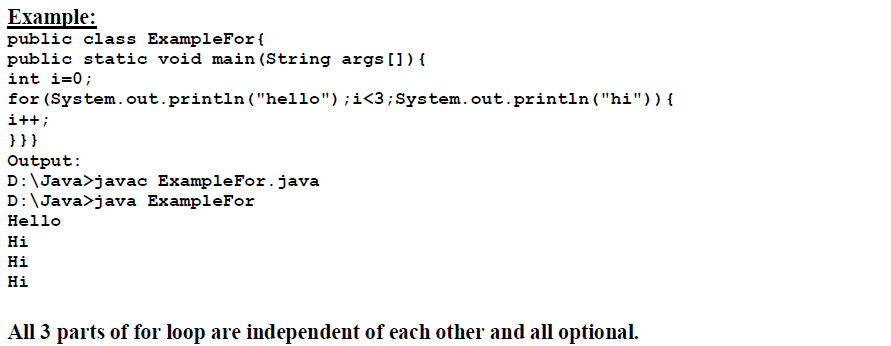
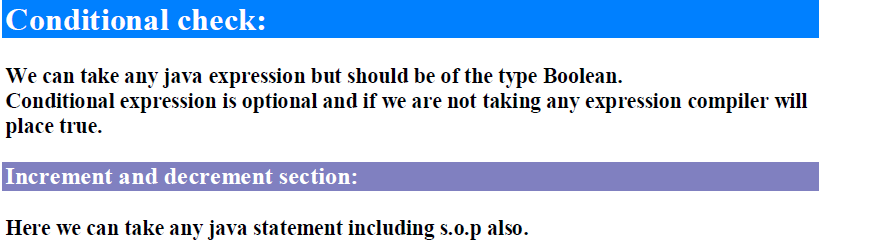
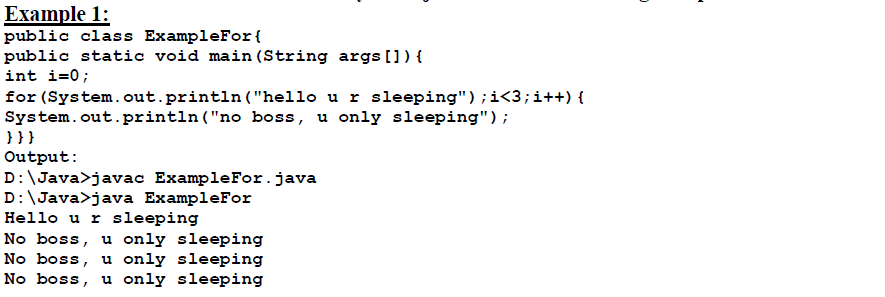
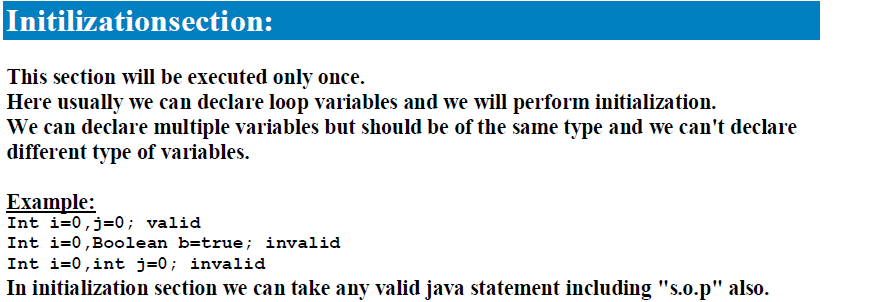


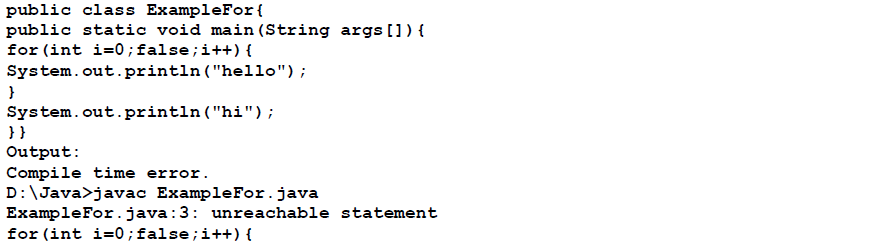


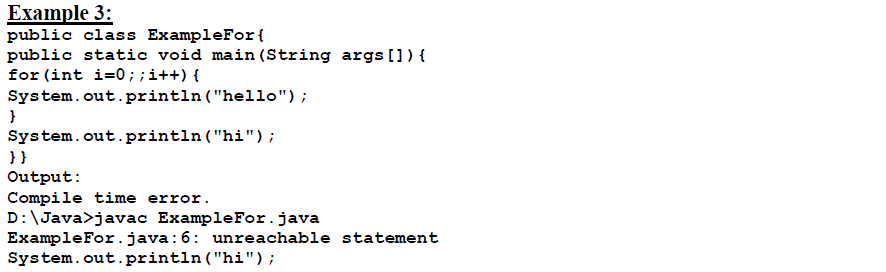


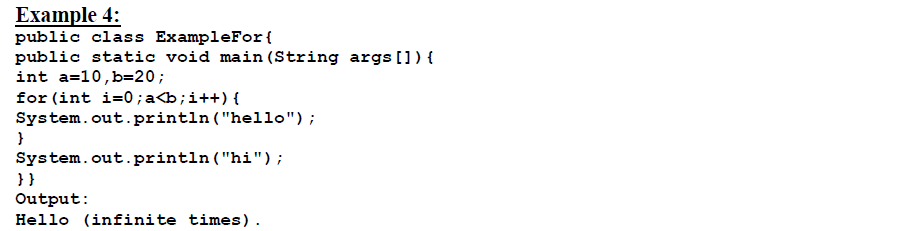


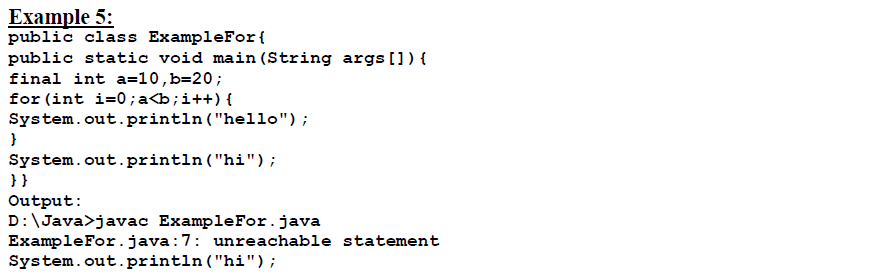


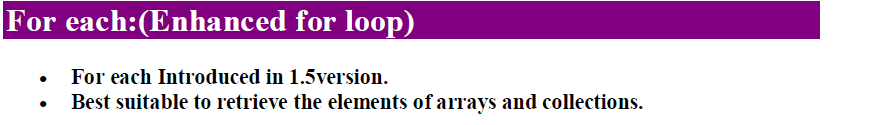


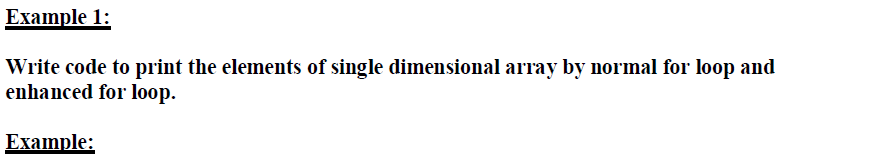


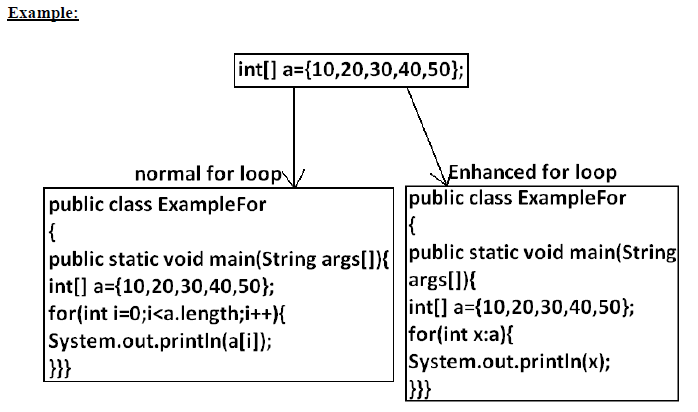


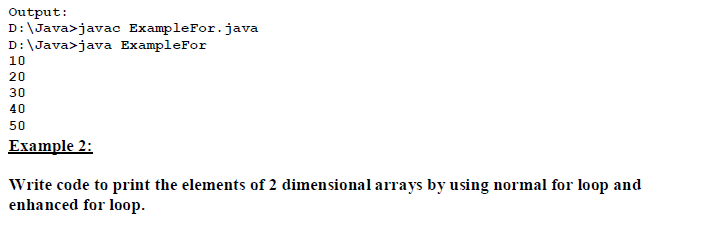


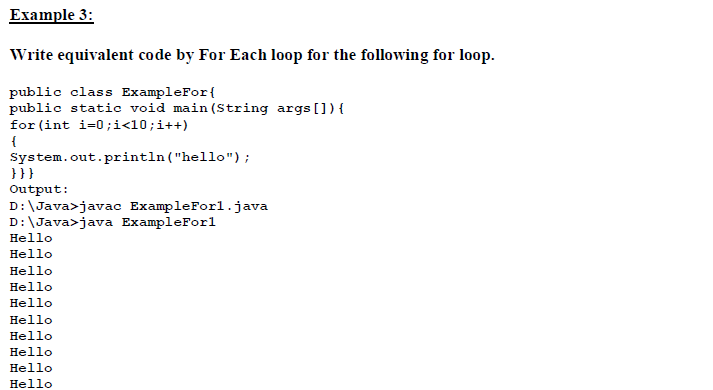
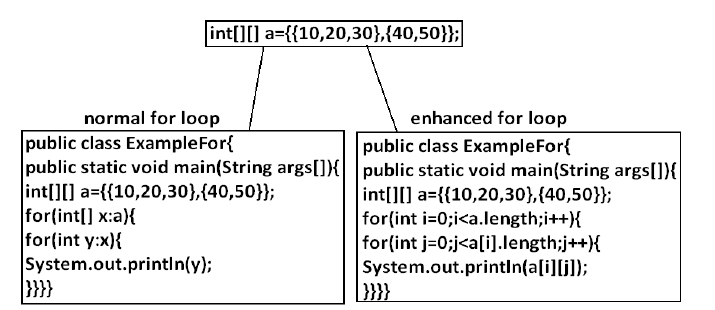


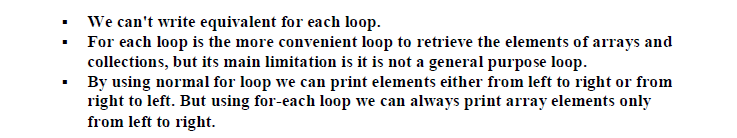


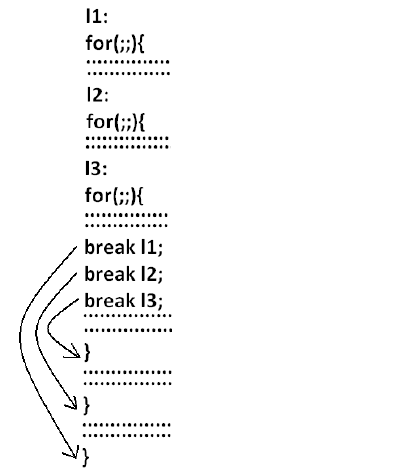
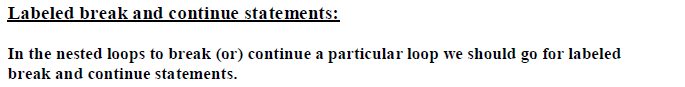
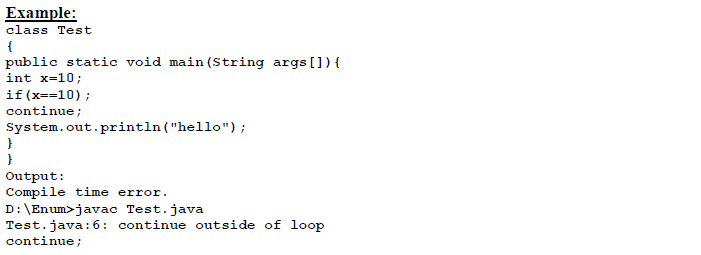
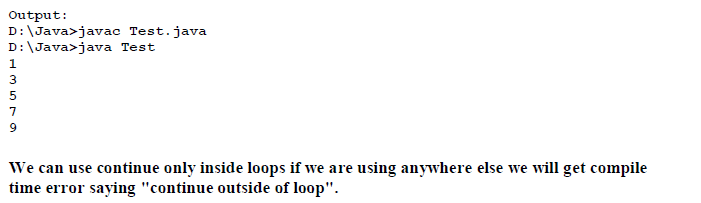
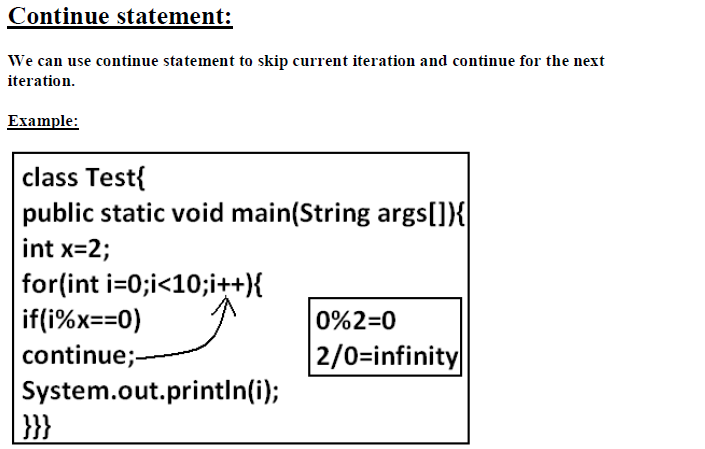
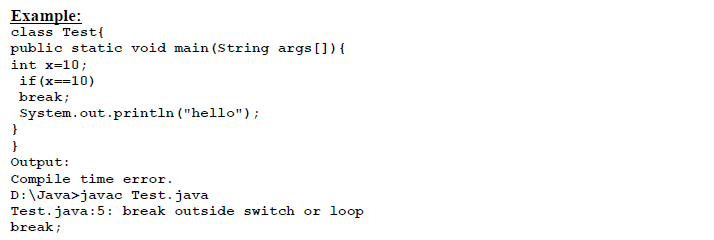
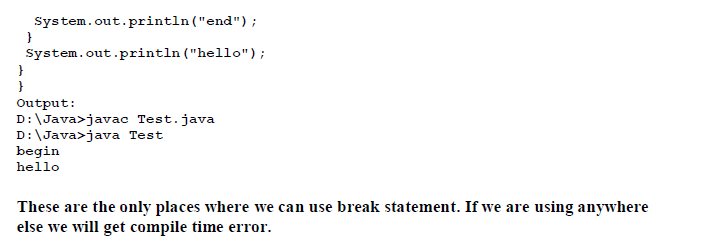
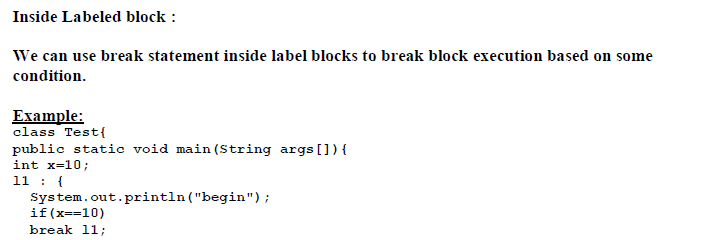
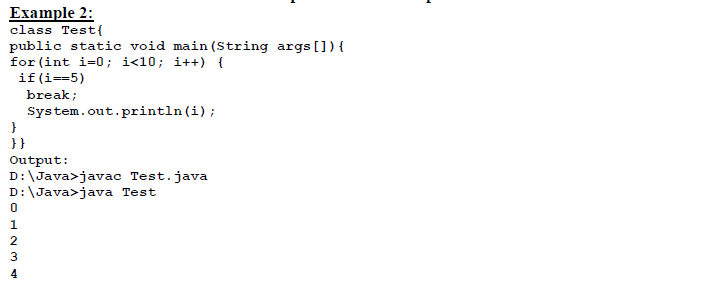
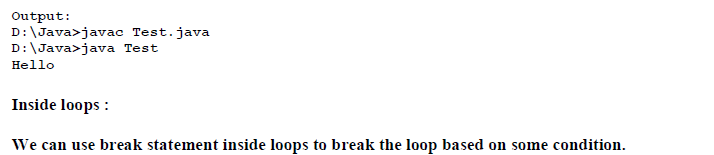
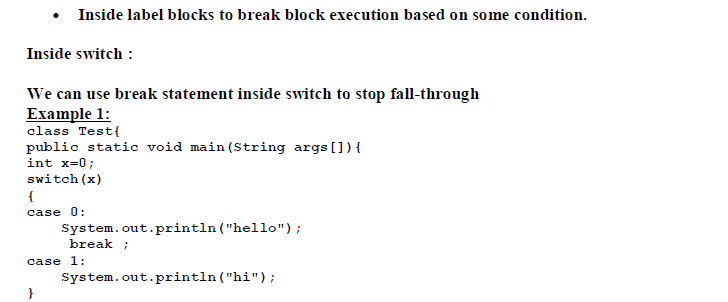
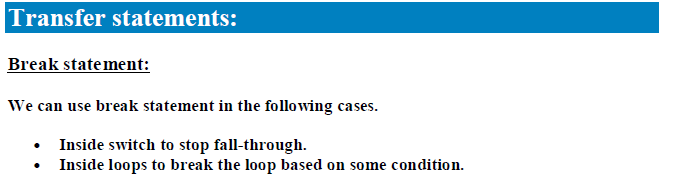
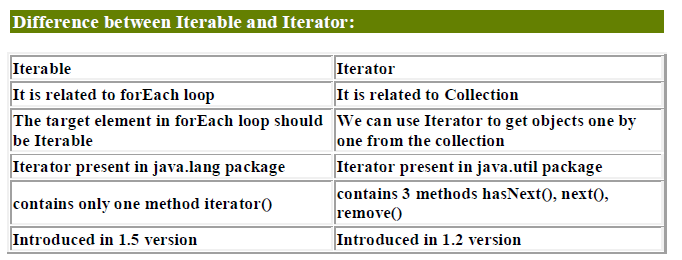
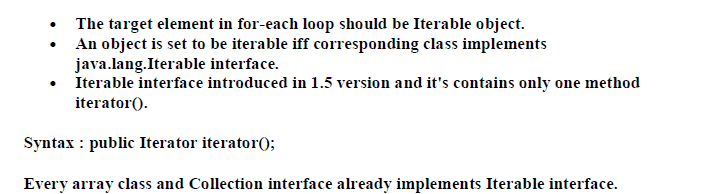
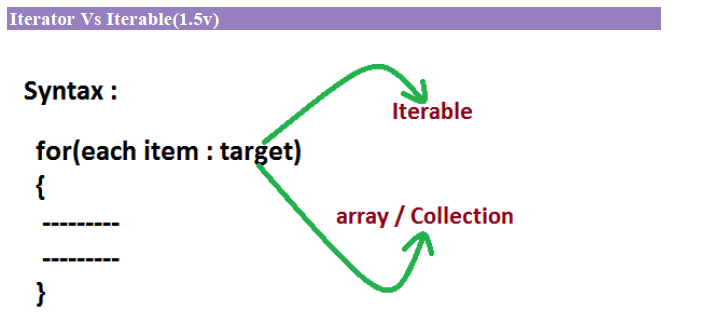


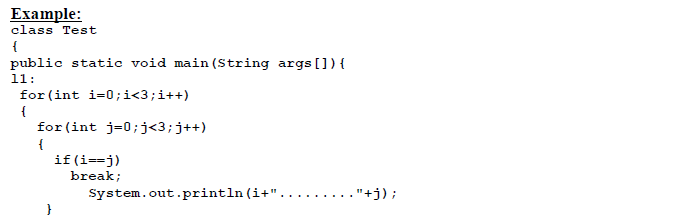


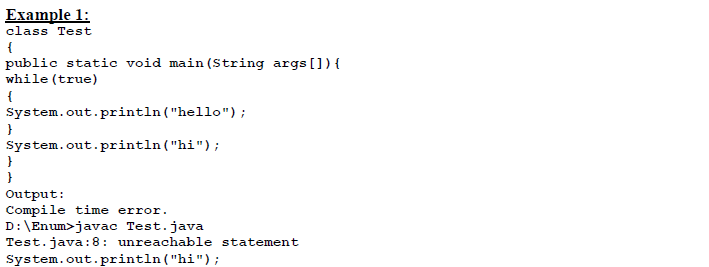
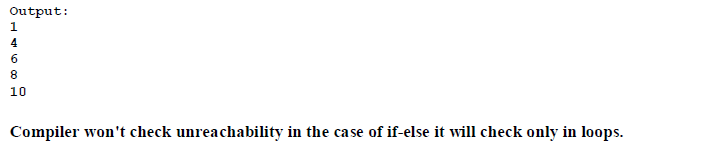
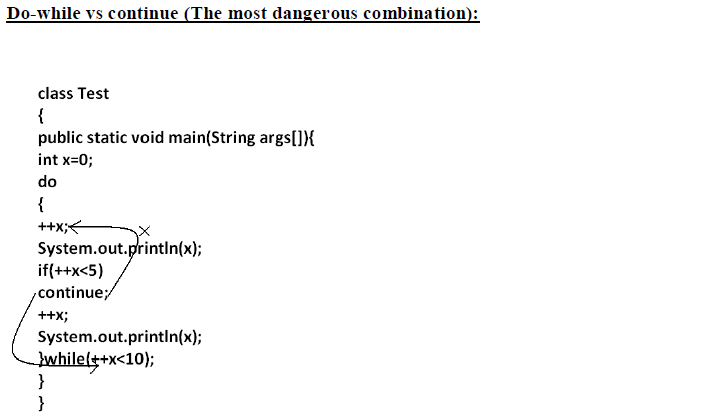


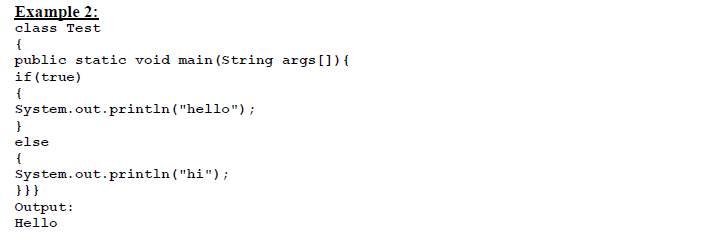












**Flow Control - Objective Questions**

1)

*public void foo( boolean a, boolean b){*

*if( a ) {*

*System.out.println("A"); /\* Line 5 \*/*

*}else if(a && b) /\* Line 7 \*/{*

*System.out.println( "A && B");*

*}else /\* Line 11 \*/{*

*if ( !b ) {*

*System.out.println( "notB") ;*

*}else{*

*System.out.println( "ELSE" ) ;*

*}*

*}*

*}*

A. If a is true and b is true then the output is "A && B"

B. If a is true and b is false then the output is "notB"

C. If a is false and b is true then the output is "ELSE"

D. If a is false and b is false then the output is "ELSE"

**Answer: C**

The output is "ELSE". Only when a is false do the output lines after 11 get some chance of executing.If b is true then !b become false,then else block will execute.

2) **What will be the output of the program?**

*int i = l, j = -1;*

*switch (i){*

*case 0, 1:*

*j = 1; /\* Line 4 \*/*

*case 2:*

*j = 2;*

*default:*

*j = 0;*

*}*

*System.out.println("j = " + j);*

A. j = -1

B. j = 0

C. j = 1

D. Compilation fails.

**Answer: D**

The case statement takes only a single argument. The case statement on line 4 is given two arguments so the compiler complains.

**3)**

**What will be the output of the program?**

*public class Switch2{*

*final static short x = 2;*

*public static int y = 0;*

*public static void main(String [] args){*

*for (int z=0; z < 3; z++){*

*switch (z){*

*case x: System.out.print("0 ");*

*case x-1: System.out.print("1 ");*

*case x-2: System.out.print("2 ");*

*}*

*}*

*}*

*}*

A. 0 1 2

B. 0 1 2 1 2 2

C. 2 1 0 1 0 0

D. 2 1 2 0 1 2

**Answer: D**

The case expressions are all legal because x is marked final, which means the expressions can be evaluated at compile time. In the first iteration of the for loop case x-2 matches, so 2 is printed. In the second iteration, x-1 is matched so 1 and 2 are printed (remember, once a match is found all remaining statements are executed until a break statement is encountered). In the third iteration, x is matched. So 0 1 and 2 are printed.

**4)**

*public void test(int x){*

*int odd = 1;*

*if(odd) /\* Line 4 \*/{*

*System.out.println("odd");*

*}else{*

*System.out.println("even");*

*}*

*}*

**Which statement is true?**

A. Compilation fails.

B. "odd" will always be output.

C. "even" will always be output.

D. "odd" will be output for odd values of x, and "even" for even values.

**Answer: A**

The compiler will complain because of incompatible types (line 4), the if expects a boolean but it gets an integer.

**5)**

*public class While{*

*public void loop(){*

*int x= 0;*

*while ( 1 ) /\* Line 6 \*/{*

*System.out.print("x plus one is " + (x + 1)); /\* Line 8 \*/*

*}*

*}*

*}*

A. There is a syntax error on line 1.

B. There are syntax errors on lines 1 and 6

C. There are syntax errors on lines 1, 6, and 8.

D. There is a syntax error on line 6.

**Answer: D**

Using the integer 1 in the while statement, or any other looping or conditional construct for that matter, will result in a compiler error. This is old C Program syntax, not valid Java.

A, B and C are incorrect because line 1 is valid (Java is case sensitive so While is a valid class name). Line 8 is also valid because an equation may be placed in a String operation as shown.

**6)**

What will be the output of the program?

*int i = 1, j = 10;*

*do{*

*if(i > j){*

*break;*

*}*

*j--;*

*} while (++i < 5);*

*System.out.println("i = " + i + " and j = " + j);*

A. i = 6 and j = 5

B. i = 5 and j = 5

C. i = 6 and j = 4

D. i = 5 and j = 6

**Answer: D**

The order is, test i against j, if bigger, it breaks from the loop, decrements j by one, and then tests the

loop condition, where a pre-incremented by one i is tested for being lower than 5. The test is at the end of the loop, so i can reach the value of 5 before it fails. So it goes, start:

(1, 10) ( 2, 9) ( 3, 8) ( 4, 7) ( 5, 6) loop condition fails.

7)

**What will be the output of the program?**

*boolean bool = true;*

*if(bool = false) /\* Line 2 \*/{*

*System.out.println("a");*

*}else if(bool) /\* Line 6 \*/{*

*System.out.println("b");*

*}else if(!bool) /\* Line 10 \*/{*

*System.out.println("c"); /\* Line 12 \*/*

*}else{*

*System.out.println("d");*

*}*

A. a

B. b

C. c

D. d

**Answer: C**

Look closely at line 2, is this an equality check (==) or an assignment (=). The condition at line 2 evaluates to false and also assigns false to bool. bool is now false so the condition at line 6 is not true. The condition at line 10 checks to see if bool is not true ( if !(bool == true) ), it isn't so line 12 is executed.

**8)**

What will be the output of the program?

*public class If1{*

*static boolean b;*

*public static void main(String [] args){*

*short hand = 42;*

*if ( hand < 50 & !b ) /\* Line 7 \*/*

*hand++;*

*if ( hand > 50 ); /\* Line 9 \*/*

*else if ( hand > 40 ){*

*hand += 7;*

*hand++;*

*}else*

*--hand;*

*System.out.println(hand);*

*}*

*}*

A. 41

B. 42

C. 50

D. 51

**Answer: D**

In Java, boolean instance variables are initialized to false, so the if test on line 7 is true and hand is

incremented. Line 9 is legal syntax, a do nothing statement. The else-if is true so hand has 7 added to it and

is then incremented.

9)

**What will be the output of the program?**

*for (int i = 0; i < 4; i += 2){*

*System.out.print(i + " ");*

*}*

*System.out.println(i); /\* Line 5 \*/*

A. 0 2 4

B. 0 2 4 5

C. 0 1 2 3 4

D. Compilation fails.

**Answer: D**

Compilation fails on the line 5 - System.out.println(i); as the variable i has only been declared within the for loop. It is not a recognized variable outside the code block of loop.

**10)**

**What will be the output of the program?**

*public class Delta{*

*static boolean foo(char c){*

*System.out.print(c);*

*return true;*

*}*

*public static void main( String[] argv ){*

*int i = 0;*

*for (foo('A'); foo('B') && (i < 2); foo('C')){*

*i++;*

*foo('D');*

*}*

*}*

*}*

A. ABDCBDCB

B. ABCDABCD

C. Compilation fails

D. An exception is thrown at runtime.

**Answer: A**

In the for loop first initialization part will execute only once.then condition part,if it is true loop body

will be executed.After that increment/decrement section will be executed.and then again condition and so on.

**Directions:**

*1. int i = 10;*

*2. while(i++ <= 10){*

*3. i++;*

*4. }*

*5. System.out.print(i);*

**Question:**

**What will be the result?**

Option A): 13

Option B): 10

Option C): 12

Option D): 14

**Correct Answer is Option A):13**

**1.**

**What will be the output of the program?**

*class PassA {*

*public static void main(String [] args) {*

*PassA p = new PassA();*

*p.start();*

*}*

*void start() {*

*long [] a1 = {3,4,5};*

*long [] a2 = fix(a1);*

*System.out.print(a1[0] + a1[1] + a1[2] + " ");*

*System.out.println(a2[0] + a2[1] + a2[2]);*

*}*

*long [] fix(long [] a3) {*

*a3[1] = 7;*

*return a3;*

*}*

*}*

**Explanation:**

**Output: 15 15**

The reference variables a1 and a3 refer to the same long array object. When the [1] element is updated in the fix() method, it is updating the array referred to by a1. The reference variable a2 refers to the same array object.

So Output: 3+7+5+" "3+7+5

Output: 15 15 Because Numeric values will be added

**What will be the output of the program?**

*class Test {*

*public static void main(String [] args){*

*Test p = new Test();*

*p.start();*

*}*

*void start() {*

*boolean b1 = false;*

*boolean b2 = fix(b1);*

*System.out.println(b1 + " " + b2);*

*}*

*boolean fix(boolean b1) {*

*b1 = true;*

*return b1;*

*}*

*}*

**Answer:** false true

**Explanation:**

The boolean b1 in the fix() method is a different boolean than the b1 in the start() method. The b1 in the start() method is not updated by the fix() method.

**What will be the output of the program?**

*class PassS {*

*public static void main(String [] args) {*

*PassS p = new PassS();*

*p.start();*

*}*

*void start() {*

*String s1 = "slip";*

*String s2 = fix(s1);*

*System.out.println(s1 + " " + s2);*

*}*

*String fix(String s1) {*

*s1 = s1 + "stream";*

*System.out.print(s1 + " ");*

*return "stream";*

*}*

*}*

**Answer:** slipstream slip stream

**Explanation:**When the fix() method is first entered, start()'s s1 and fix()'s s1 reference variables both refer to the same String object (with a value of "slip"). Fix()'s s1 is reassigned to a new object that is created when the concatenation occurs (this second String object has a value of "slipstream"). When the program returns to start(), another String object is created, referred to by s2 and with a value of "stream".

**What will be the output of the program?**

*class BitShift {*

*public static void main(String [] args) {*

*int x = 0x80000000;*

*System.out.print(x + " and ");*

*x = x >>> 31;*

*System.out.println(x);*

*}*

*}*

*Answer: -2147483648 and 1*

***Explanation:***

*Option A is correct. The >>> operator moves all bits to the right, zero filling the left bits. The bit transformation looks like this:*

***Before****: 1000 0000 0000 0000 0000 0000 0000 0000*

***After****: 0000 0000 0000 0000 0000 0000 0000 0001*

*Option C is incorrect because the >>> operator zero fills the left bits, which in this case changes the sign of x, as shown.*

*Option B is incorrect because the output method print() always displays integers in base 10.*

*Option D is incorrect because this is the reverse order of the two output numbers.*

**What will be the output of the program?**

*class Equals {*

*public static void main(String [] args) {*

*int x = 100;*

*double y = 100.1;*

*boolean b = (x = y); /\* Line 7 \*/*

*System.out.println(b);*

*}*

*}*

***Answer:*** *Compilation fails*

***Explanation:***

*The code will not compile because in line 7, the line will work only if we use (x==y) in the line. The == operator compares values to produce a boolean, whereas the = operator assigns a value to variables.*

*Option A, B, and D are incorrect because the code does not get as far as compiling. If we corrected this code, the output would be false.*

**What will be the output of the program?**

*class Test {*

*public static void main(String [] args) {*

*int x=20;*

*String sup = (x < 15) ? "small" : (x < 22)? "tiny" : "huge";*

*System.out.println(sup);*

*}*

*}*

***Answer:*** *tiny*

***Explanation:***

*This is an example of a nested ternary operator. The second evaluation (x < 22) is true, so the "tiny" value is assigned to sup.*

**What will be the output of the program?**

*class Test {*

*public static void main(String [] args) {*

*int x= 0;*

*int y= 0;*

*for (int z = 0; z < 5; z++) {*

*if (( ++x > 2 ) && (++y > 2)) {*

*x++;*

*}*

*}*

*System.out.println(x + " " + y);*

*}*

*}*

***Answer:*** *6 3*

***Explanation****:*

*In the first two iterations x is incremented once and y is not because of the short circuit && operator. In the third and forth iterations x and y are each incremented, and in the fifth iteration x is doubly incremented and y is incremented.*

**What will be the output of the program?**

*class Test {*

*public static void main(String [] args) {*

*int x= 0;*

*int y= 0;*

*for (int z = 0; z < 5; z++) {*

*if (( ++x > 2 ) || (++y > 2)) {*

*x++;*

*}*

*}*

*System.out.println(x + " " + y);*

*}*

*}*

***Answer:*** *8 2*

***Explanation:***

The first two iterations of the for loop both x and y are incremented. On the third iteration x is incremented, and for the first time becomes greater than 2. The short circuit or operator || keeps y from ever being incremented again and x is incremented twice on each of the last three iterations.

**What will be the output of the program?**

*class Bitwise {*

*public static void main(String [] args) {*

*int x = 11 & 9;*

*int y = x ^ 3;*

*System.out.println( y | 12 );*

*}*

*}*

***Answer:*** *14*

***Explanation:***

*The & operator produces a 1 bit when both bits are 1. The result of the & operation is 9. The ^ operator produces a 1 bit when exactly one bit is 1; the result of this operation is 10. The | operator produces a 1 bit when at least one bit is 1; the result of this operation is 14.*

**What will be the output of the program?**

*class SSBool {*

*public static void main(String [] args) {*

*boolean b1 = true;*

*boolean b2 = false;*

*boolean b3 = true;*

*if ( b1 & b2 | b2 & b3 | b2 ) /\* Line 8 \*/*

*System.out.print("ok ");*

*if ( b1 & b2 | b2 & b3 | b2 | b1 ) /\*Line 10\*/*

*System.out.println("dokey");*

*}*

*}*

***Answer****: dokey*

***Explanation:***

*The & operator has a higher precedence than the | operator so that on line 8 b1 and b2 are evaluated together as are b2 & b3. The final b1 in line 10 is what causes that if test to be true. Hence it prints "dokey".*

***What will be the output of the program?***

*class SC2 {*

*public static void main(String [] args) {*

*SC2 s = new SC2();*

*s.start();*

*}*

*void start() {*

*int a = 3;*

*int b = 4;*

*System.out.print(" " + 7 + 2 + " ");*

*System.out.print(a + b);*

*System.out.print(" " + a + b + " ");*

*System.out.print(foo() + a + b + " ");*

*System.out.println(a + b + foo());*

*}*

*String foo() {*

*return "foo";*

*}*

*}*

***Answer****: 72 7 34 foo34 7foo*

***Explanation:***

*Because all of these expressions use the + operator, there is no precedence to worry about and all of the expressions will be evaluated from left to right. If either operand being evaluated is a String, the + operator will concatenate the two operands; if both operands are numeric, the + operator will add the two operands.*

**What will be the output of the program?**

*class Test {*

*static int s;*

*public static void main(String [] args) {*

*Test p = new Test();*

*p.start();*

*System.out.println(s);*

*}*

*void start() {*

*int x = 7;*

*twice(x);*

*System.out.print(x + " ");*

*}*

*void twice(int x) {*

*x = x\*2;*

*s = x;*

*}*

*}*

***Answer****: 7 14*

***Explanation:***

*The int x in the twice() method is not the same int x as in the start() method. Start()'s x is not affected by the twice() method. The instance variable s is updated by twice()'s x, which is 14.*

**What will be the output of the program?**

*class Two {*

*byte x;*

*}*

*class PassO {*

*public static void main(String [] args) {*

*PassO p = new PassO();*

*p.start();*

*}*

*void start() {*

*Two t = new Two();*

*System.out.print(t.x + " ");*

*Two t2 = fix(t);*

*System.out.println(t.x + " " + t2.x);*

*}*

*Two fix(Two tt) {*

*tt.x = 42;*

*return tt;*

*}*

*}*

***Answer****: 0 42 42*

***Explanation:***

*In the fix() method, the reference variable tt refers to the same object (class Two) as the t reference variable. Updating tt.x in the fix() method updates t.x (they are one in the same object). Remember also that the instance variable x in the Two class is initialized to 0.*

**What will be the output of the program?**

*class BoolArray {*

*boolean [] b = new boolean[3];*

*int count = 0;*

*void set(boolean [] x, int i) {*

*x[i] = true;*

*++count;*

*}*

*public static void main(String [] args) {*

*BoolArray ba = new BoolArray();*

*ba.set(ba.b, 0);*

*ba.set(ba.b, 2);*

*ba.test();*

*}*

*void test() {*

*if ( b[0] && b[1] | b[2] )*

*count++;*

*if ( b[1] && b[(++count - 2)] )*

*count += 7;*

*System.out.println("count = " + count);*

*}*

*}*

***Answer:*** *count = 3*

***Explanation:***

*The reference variables b and x both refer to the same boolean array. count is incremented for each call to the set() method, and once again when the first if test is true. Because of the && short circuit operator, count is not incremented during the second if test.*

**What will be the output of the program?**

*public class Test {*

*public static void leftshift(int i, int j) {*

*i <<= j;*

*}*

*public static void main(String args[]) {*

*int i = 4, j = 2;*

*leftshift(i, j);*

*System.out.printIn(i);*

*}*

*}*

***Answer:*** *4*

***Explanation:***

*Java only ever passes arguments to a method by value (i.e. a copy of the variable) and never by reference. Therefore the value of the variable i remains unchanged in the main method.*

*If you are clever you will spot that 16 is 4 multiplied by 2 twice, (4 \* 2 \* 2) = 16. If you had 16 left shifted by three bits then 16 \* 2 \* 2 \* 2 = 128. If you had 128 right shifted by 2 bits then 128 / 2 / 2 = 32. Keeping these points in mind, you don't have to go converting to binary to do the left and right bit shifts.*