MODULE 2
CORE JAVA FUNDAMENTALS

CHAPTER 1

DATA TYPES, OPERATORS

&

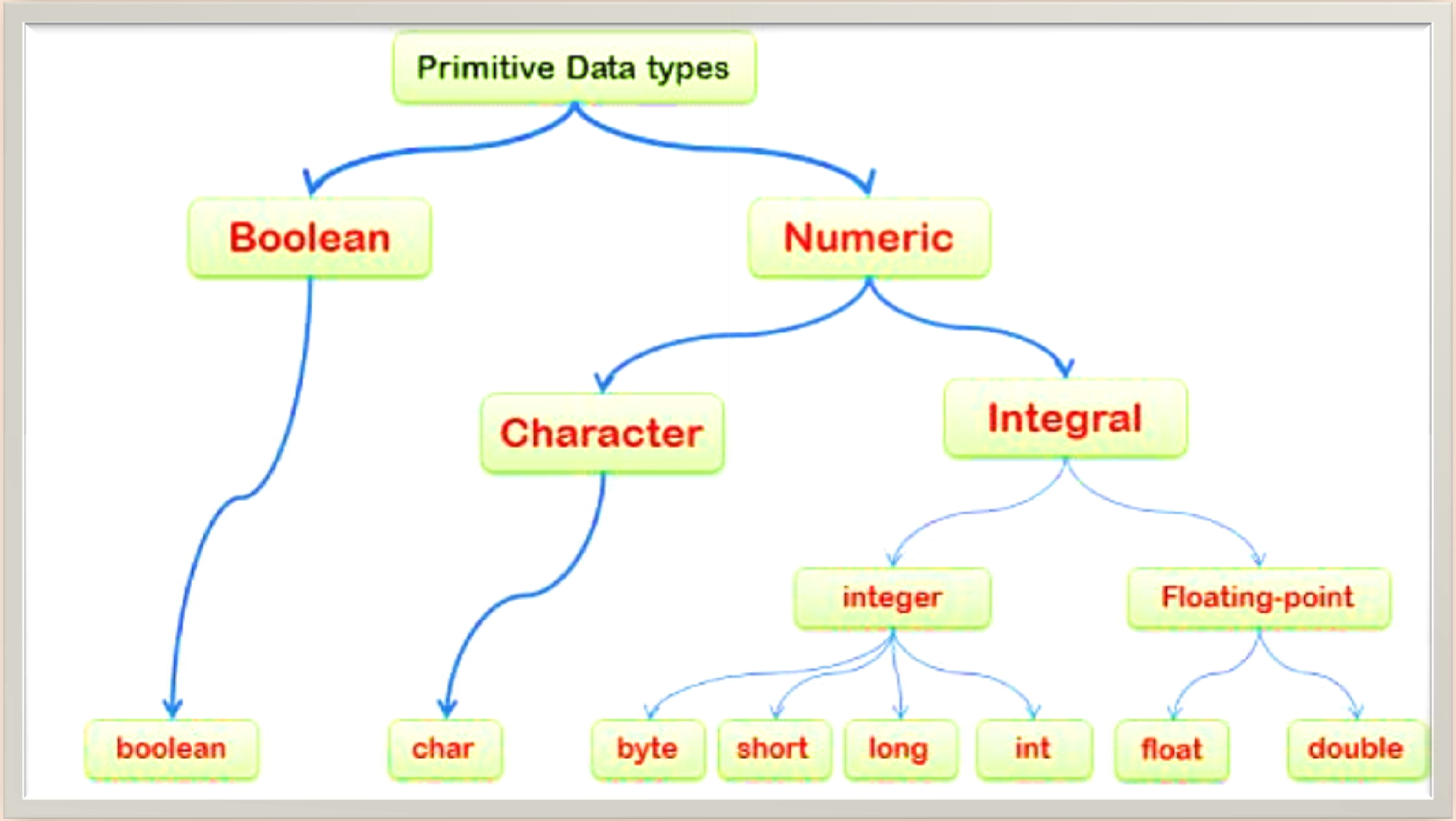
CONTROL STATEMENTS

DATA TYPES

- **Data type** defines the values that a variable can take, fo example if a variable has int data type, it can only take intege values.
- Data types specify the different sizes and values that can be stored in the variable.
- There are two types of data types in Java:

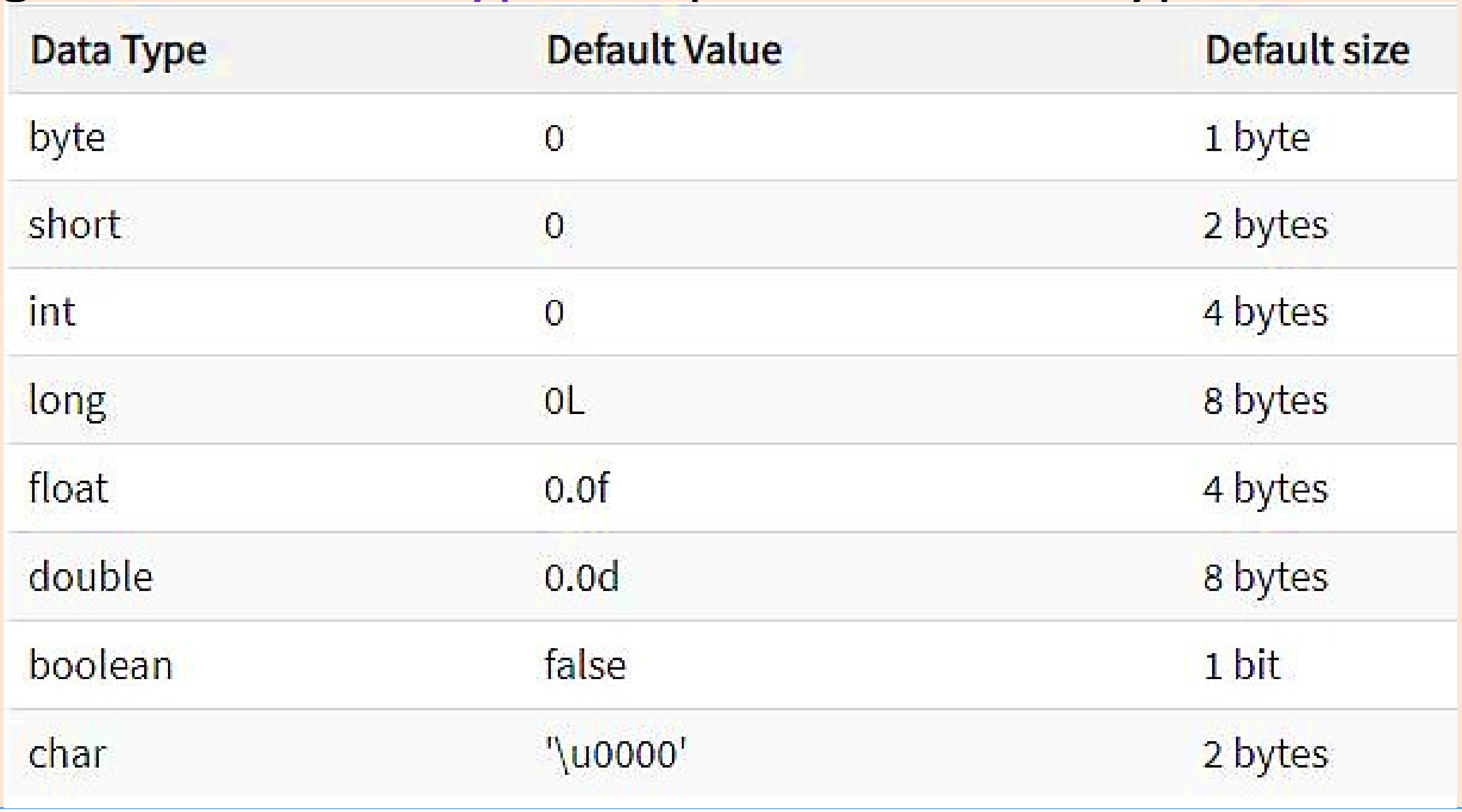
  Primitive data types

  Non-primitive data types


Primitive Data types
Boolean
Numeric
Character
Integral
integer
Floating-point
boolean
char
byte
short
long
int
float
double

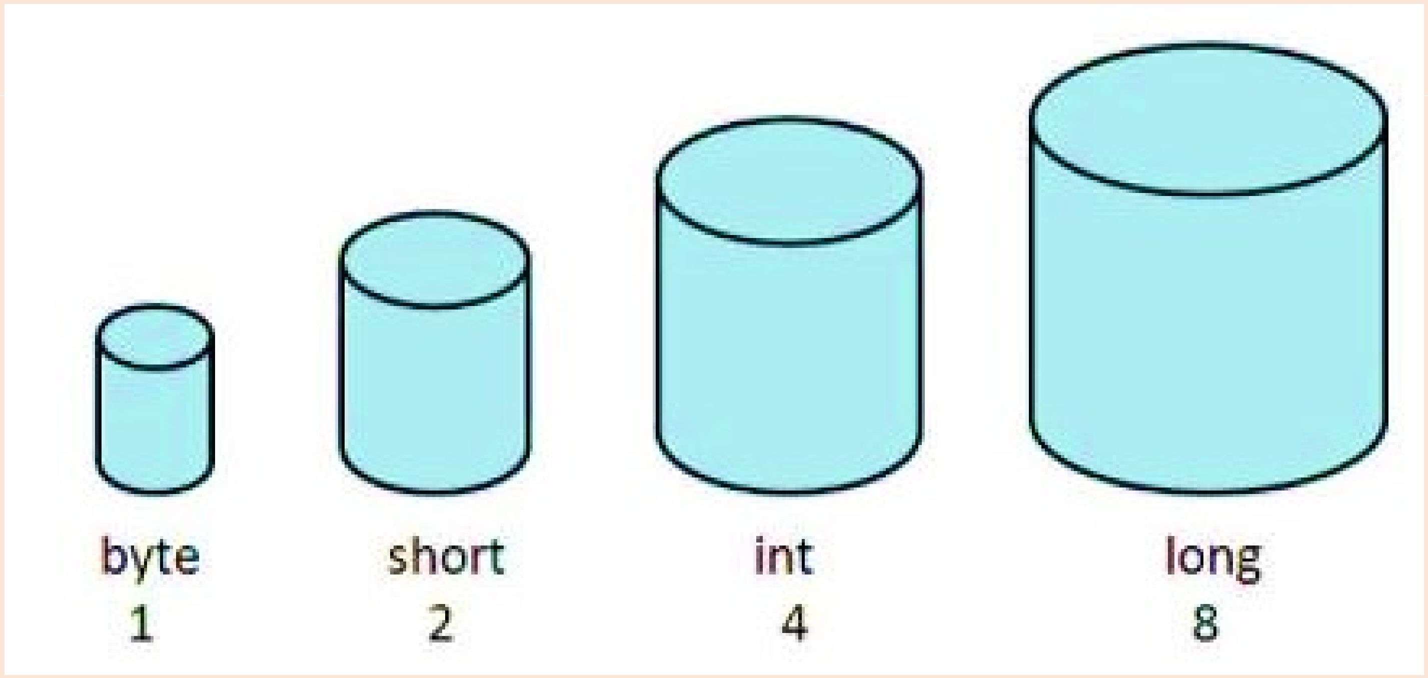
❖ Primitive Data Types (Fundamental Data Types)

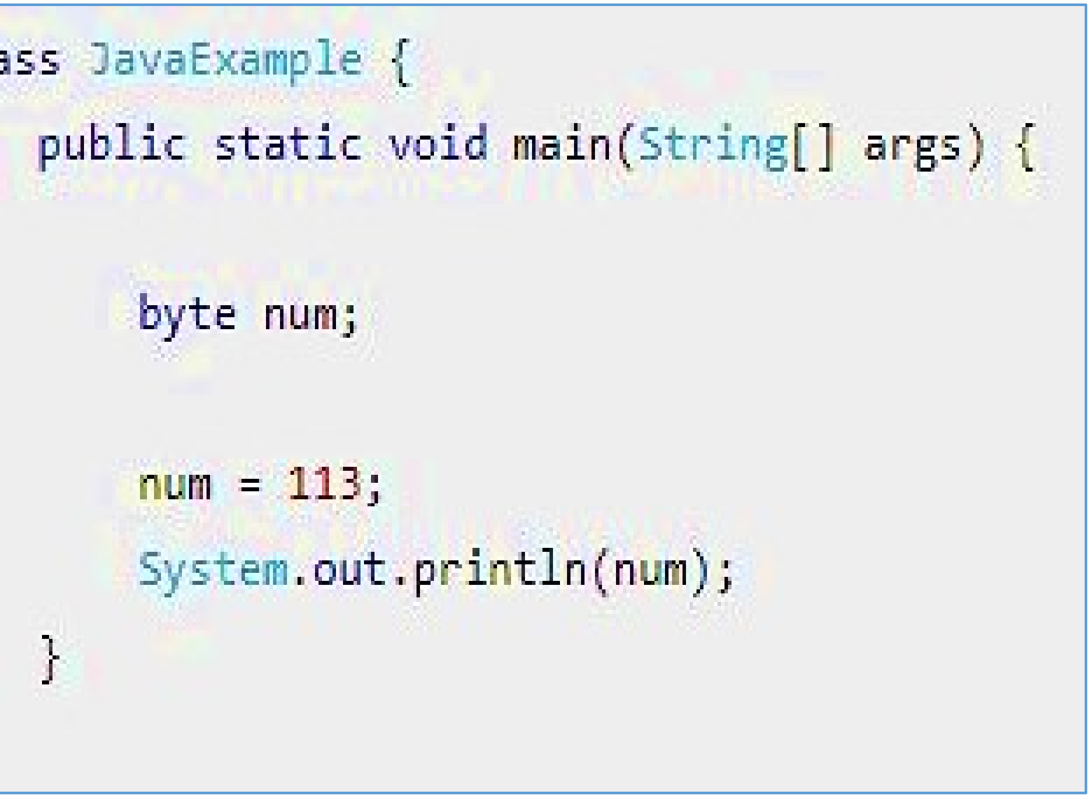
Primitive Data Types are predefined and available within the Java language. There are 8 types of primitive data types:



Data Type	Default Value	Default size
byte	0	1 byte
short	0	2 bytes
int	0	4 bytes
long	0L	8 bytes
float	0.0f	4 bytes
double	0.0d	8 bytes
boolean	false	1 bit
char	'\u0000'	2 bytes

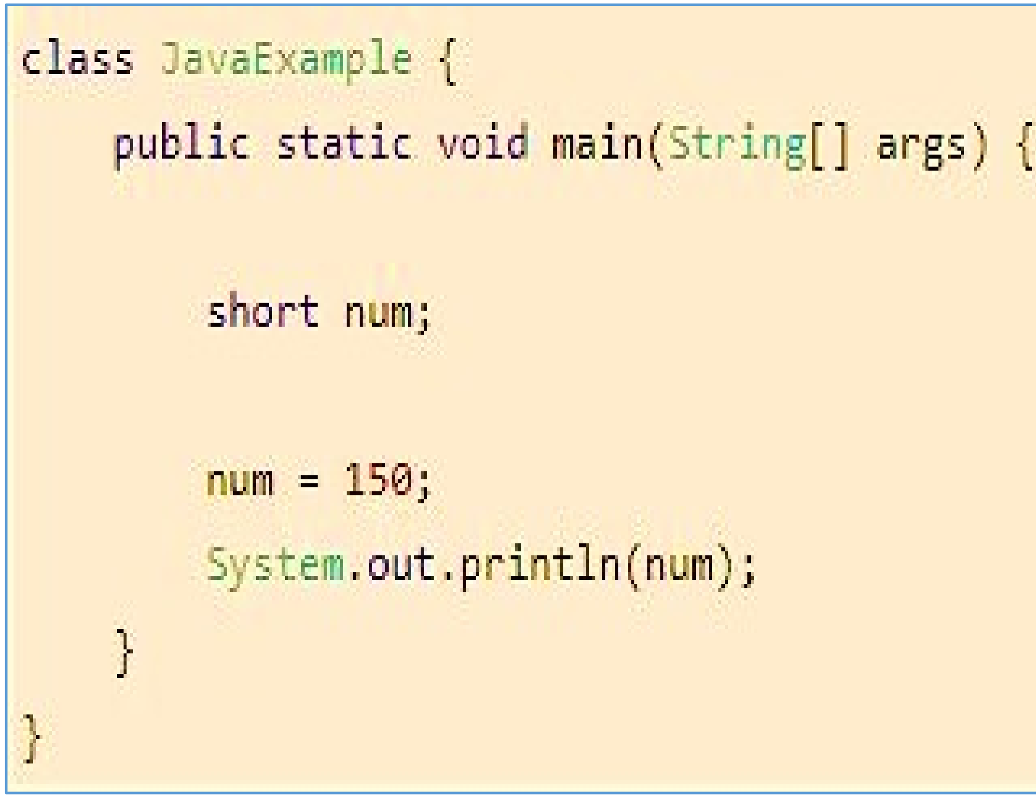
- **byte**, **short**, **int** and **long** data types are used for storing who numbers.
- **float** and **double** are used for fractional numbers.
- **char** is used for storing characters(letters).
- **boolean** data type is used for variables that holds either true c false.





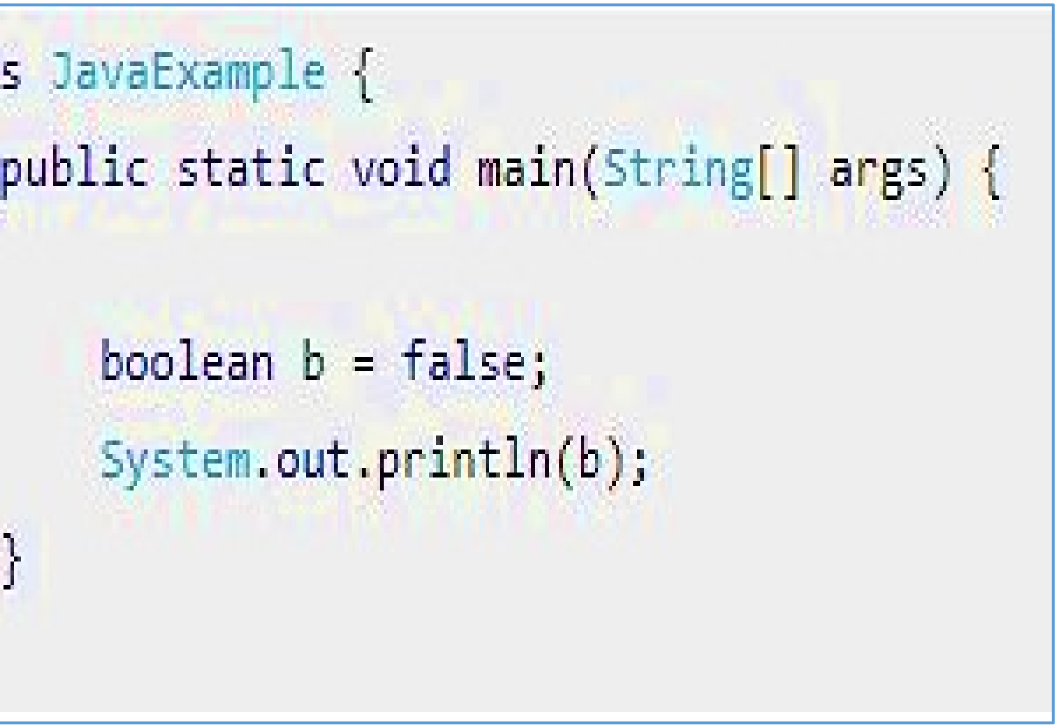
```
ass JavaExample {
 public static void main(String[] args) {

     byte num;

     num = 113;
     System.out.println(num);
 }
```

Output 113

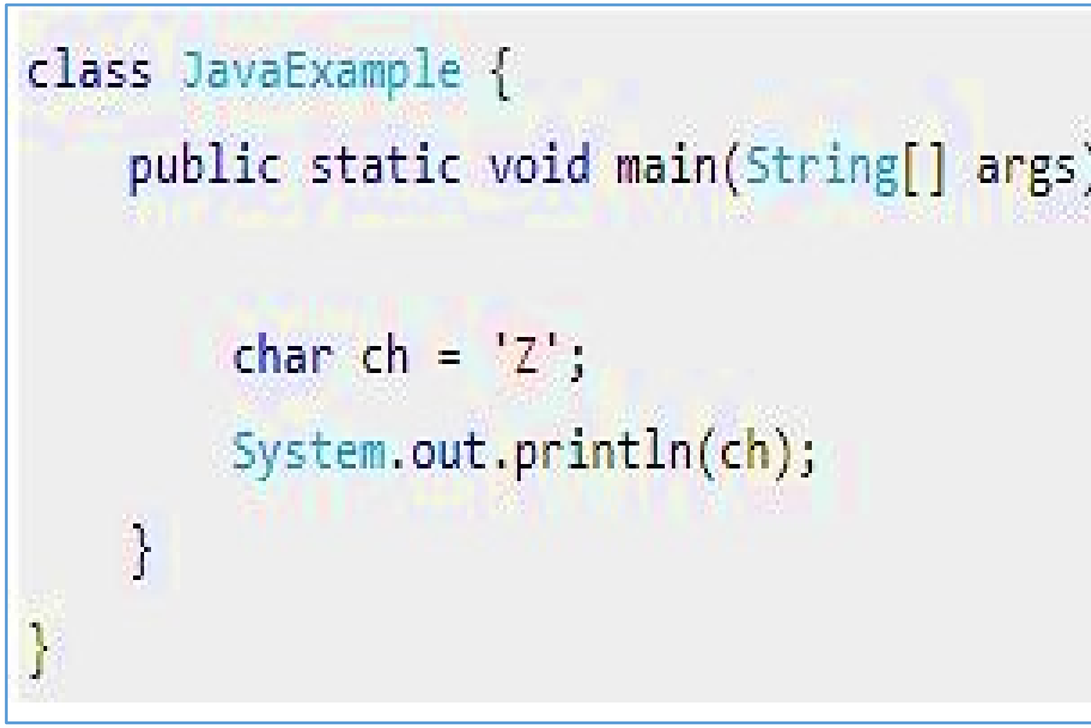


```
class JavaExample {
    public static void main(String[] args) {

        short num;

        num = 150;
        System.out.println(num);
    }
}
```

Output 150



```
s JavaExample {
public static void main(String[] args) {

    boolean b = false;
    System.out.println(b);
}
```



```
class JavaExample {
    public static void main(String[] args

        char ch = 'Z';
        System.out.println(ch);
    }
}
```

Output false

Output Z

Variables In JAVA

- **Variable in Java** is a data container that stores the data values during Java program execution.
- Variable is a memory location name of the data.
- variable="vary + able" that means its value can be changed.
- In order to use a variable in a program we need to perform 2 steps

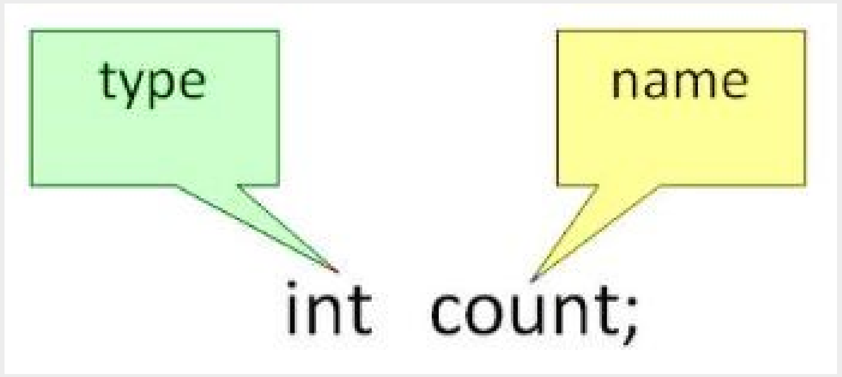
  1. Variable Declaration

  2. Variable Initialization

1. Variable Declaration

Syntax: data_type variable_name ;

Eg:
```
int a,b,c;
float pi;
double d;
```

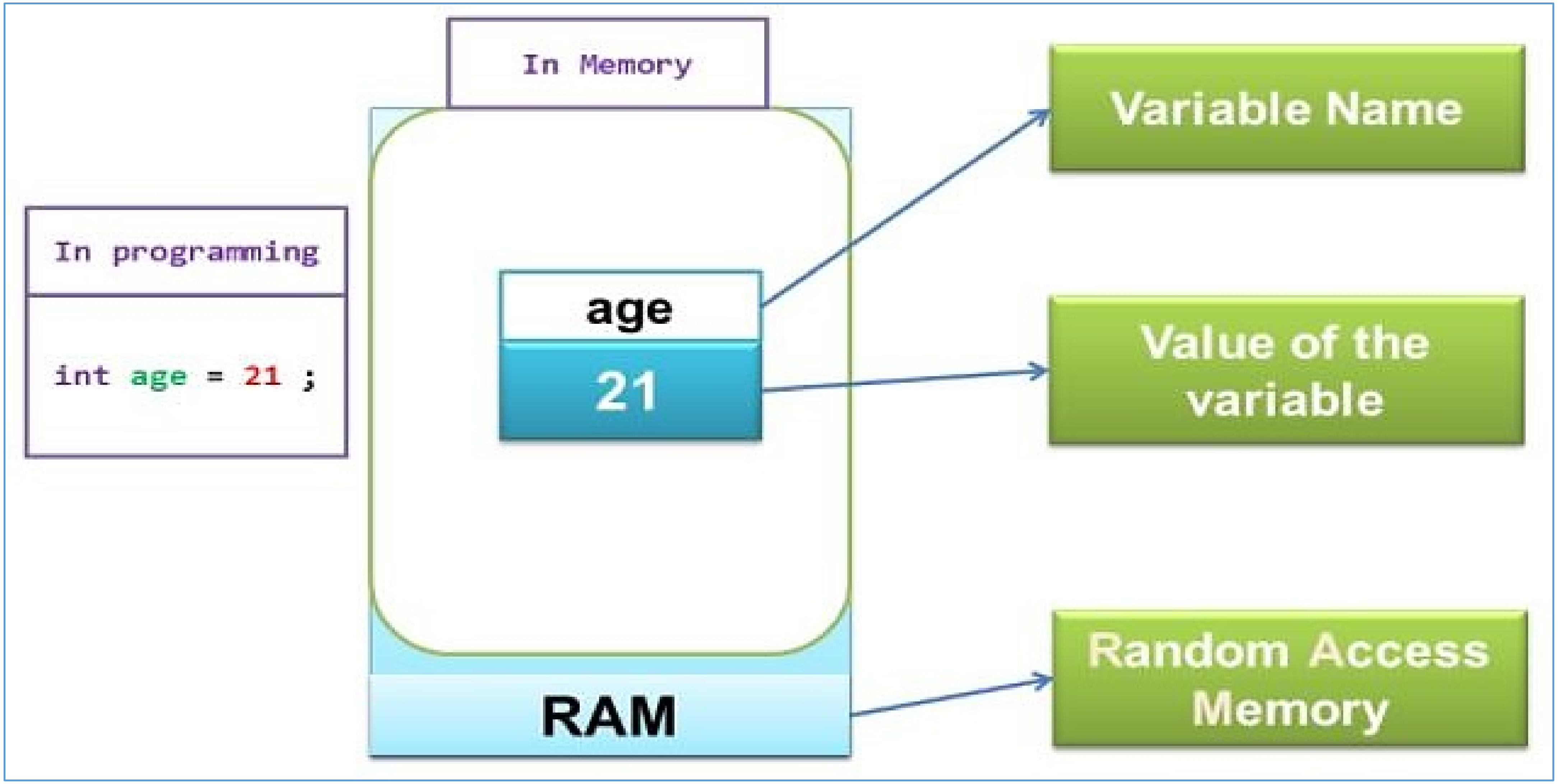


2. Variable Initialization

Syntax : data_type variable_name = value;

Eg:
```
int a=2,b=4,c=6;
float pi = 3.14f;
double val = 20.22d;
char a = ’v’;
```

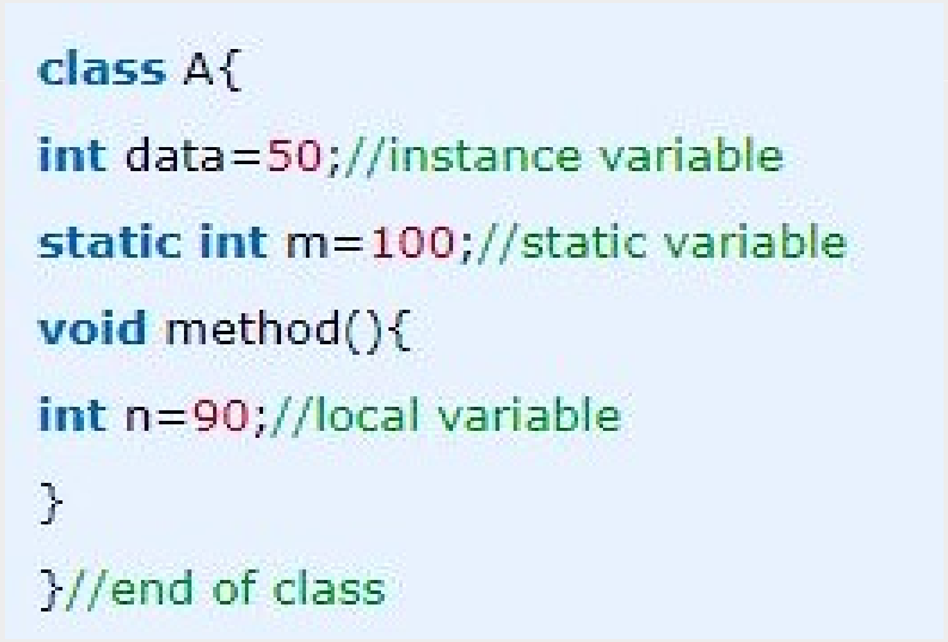
int num = 45.66;


In Memory
In programming
int age = 21 ;
age
21
RAM
Variable Name
Value of the variable
Random Access Memory

Types of variables

1. **Local variables -** declared inside the method.
2. **Instance Variable -** declared inside the class but outside the method.
3. **Static variable -** declared as with static keyword.

Example:



```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Java Type Casting or Type Conversion

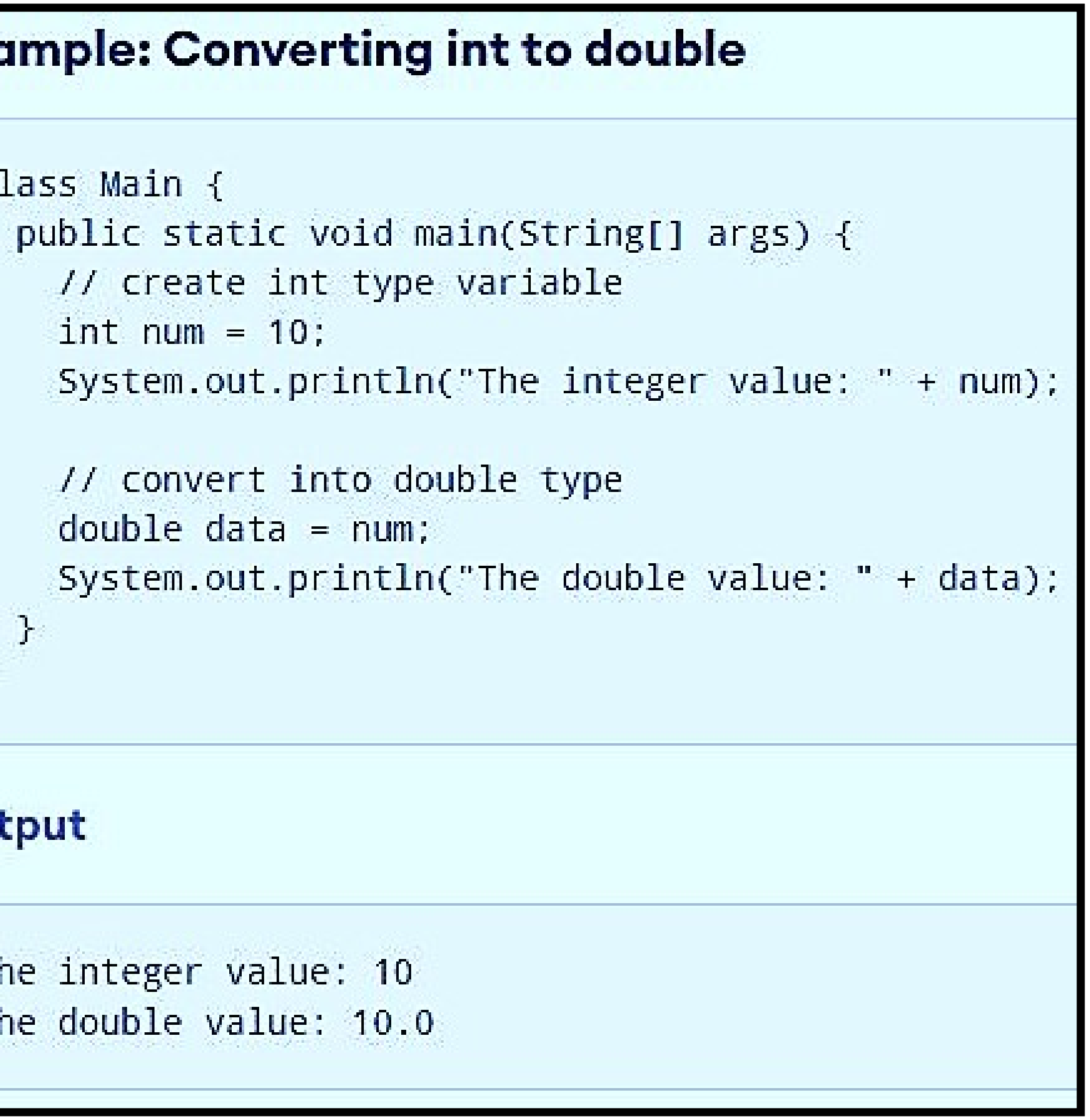
- Type casting is when you assign a value of one primitive data type to another type.
- In Java, there are two types of casting:

1. Widening Casting (automatically) – converting a smaller type to a larger type size (called Type Conversion)

   byte -> short -> char -> int -> long -> float -> double

2. Narrowing Casting (manually) – converting a larger type to a smaller size type (called Type Casting)

   double -> float -> long -> int -> char -> short -> byte



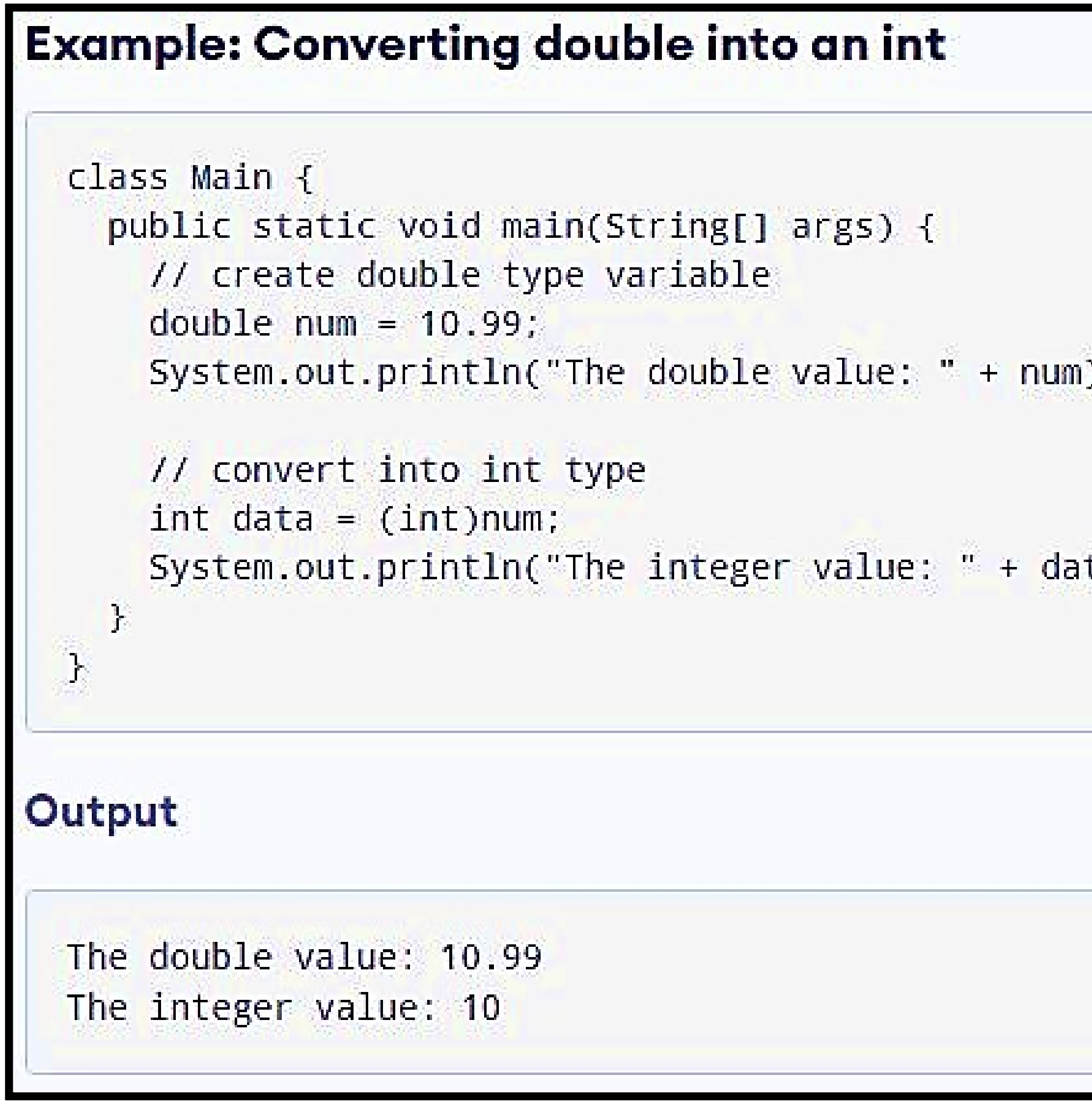
ample: Converting int to double

```
lass Main {
 public static void main(String[] args) {
   // create int type variable
   int num = 10;
   System.out.println("The integer value: " + num);

   // convert into double type
   double data = num;
   System.out.println("The double value: " + data);
 }
```

tput

```
he integer value: 10
he double value: 10.0
```

Widening



Example: Converting double into an int

```
class Main {
  public static void main(String[] args) {
    // create double type variable
    double num = 10.99;
    System.out.println("The double value: " + num

    // convert into int type
    int data = (int)num;
    System.out.println("The integer value: " + dat
  }
}
```

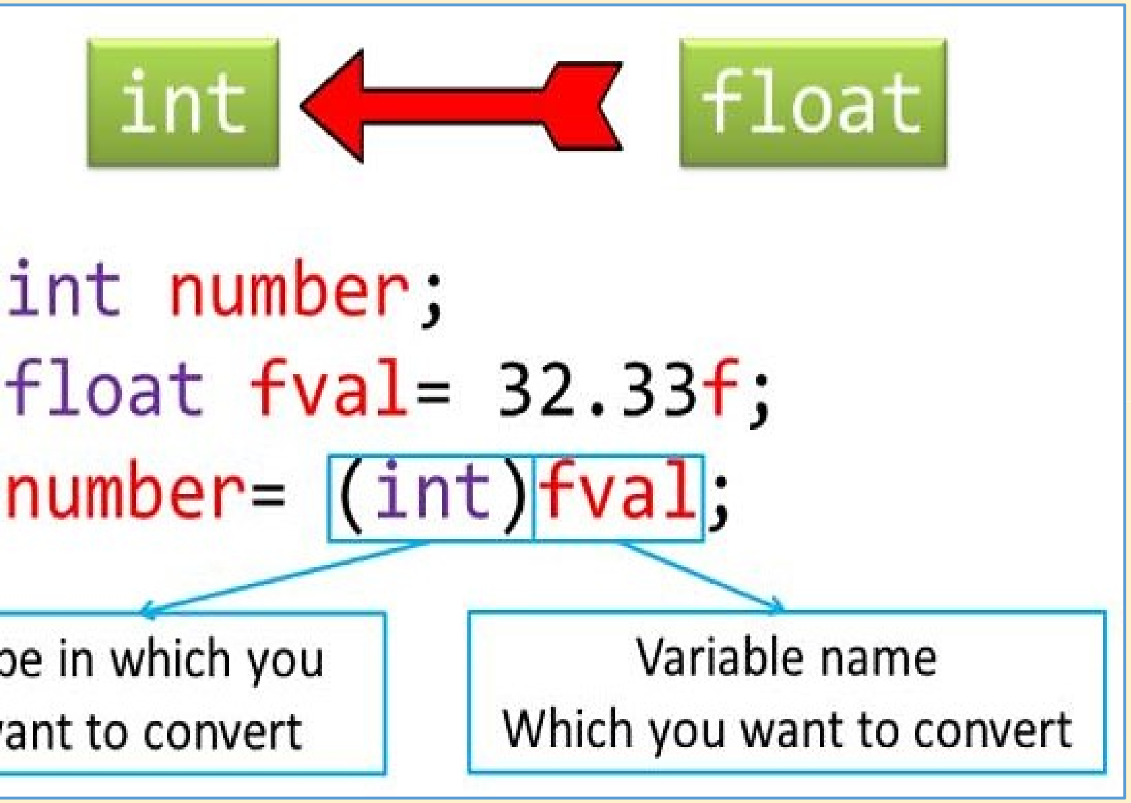
Output

```
The double value: 10.99
The integer value: 10
```

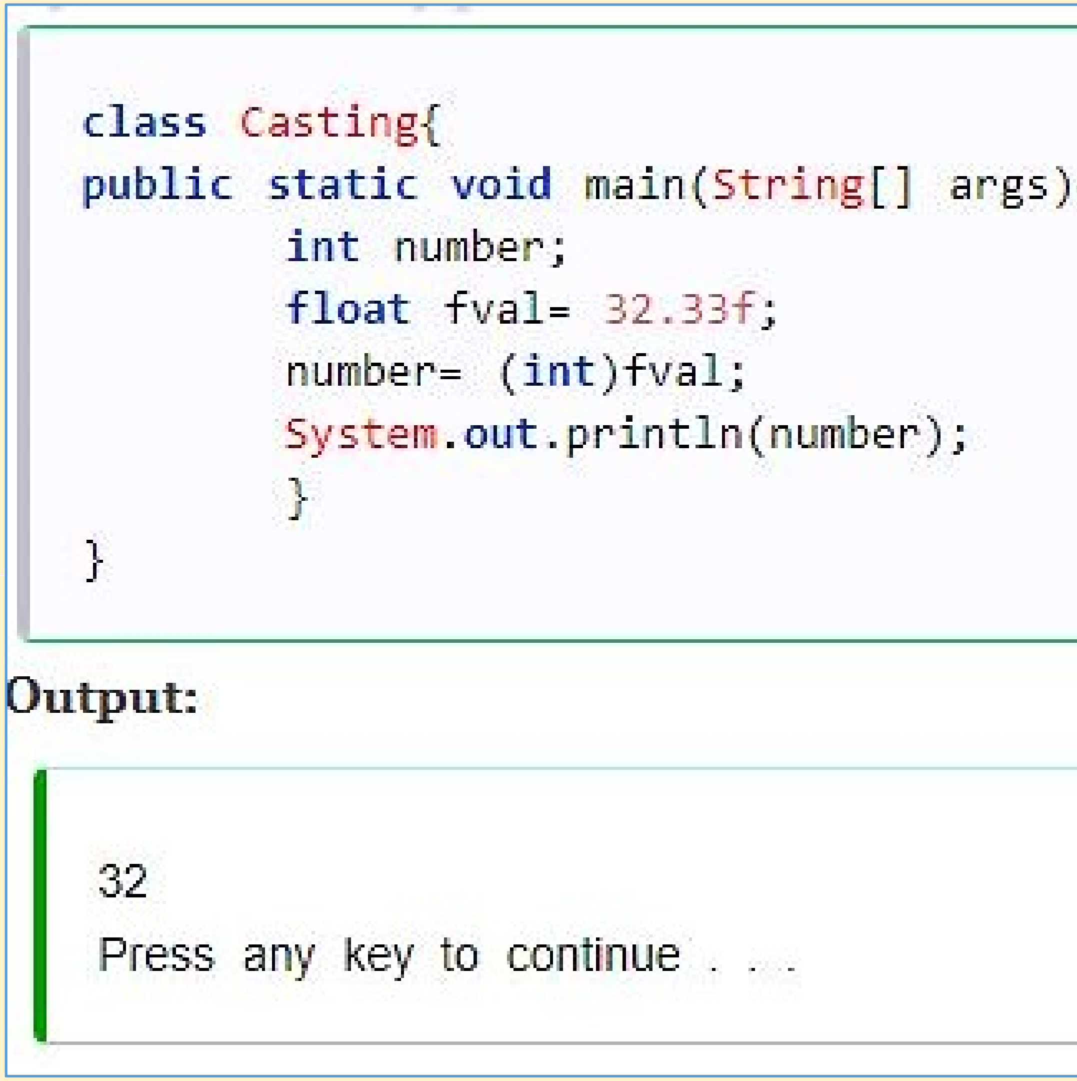
Narrowing

Truncation

- when a floating-point value is assigned to an integer typ truncation takes place, As you know, integers do not hav fractional components
- Thus, when a floating-point value is assigned to an integer typ the fractional component is lost.
- For example, if the value 45.12 is assigned to an integer, th resulting value will simply be 45. The 0.12 will have bee truncated.
- No automatic conversions from the numeric types to char o boolean. Also, char and boolean are not compatible with eac other.



```
int number;
float fval= 32.33f;
number= (int)fval;
```



```
class Casting{
public static void main(String[] args)
        int number;
        float fval= 32.33f;
        number= (int)fval;
        System.out.println(number);
        }
}
```

Output:

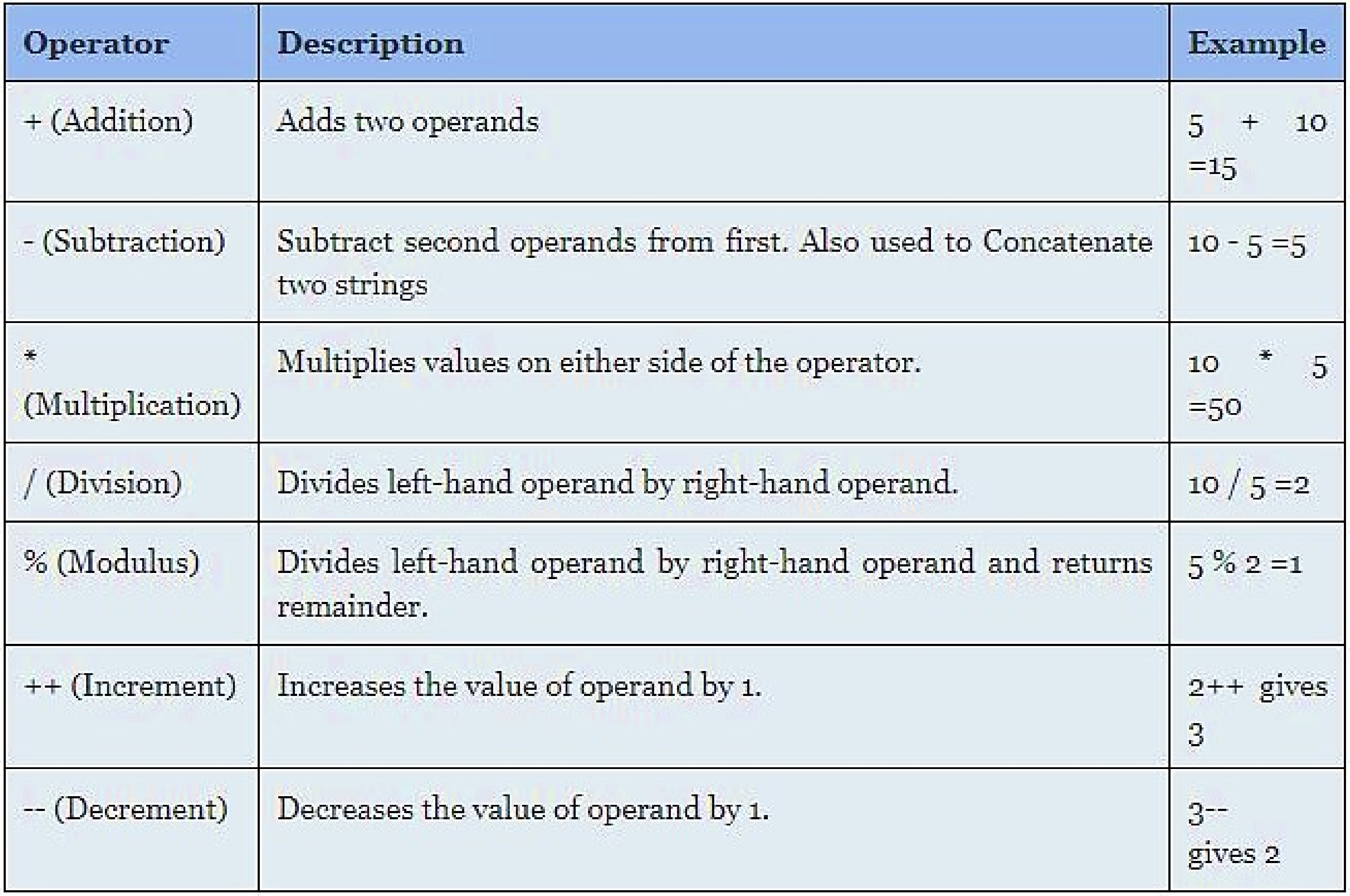
```
32
Press any key to continue . . .
```

OPERATORS

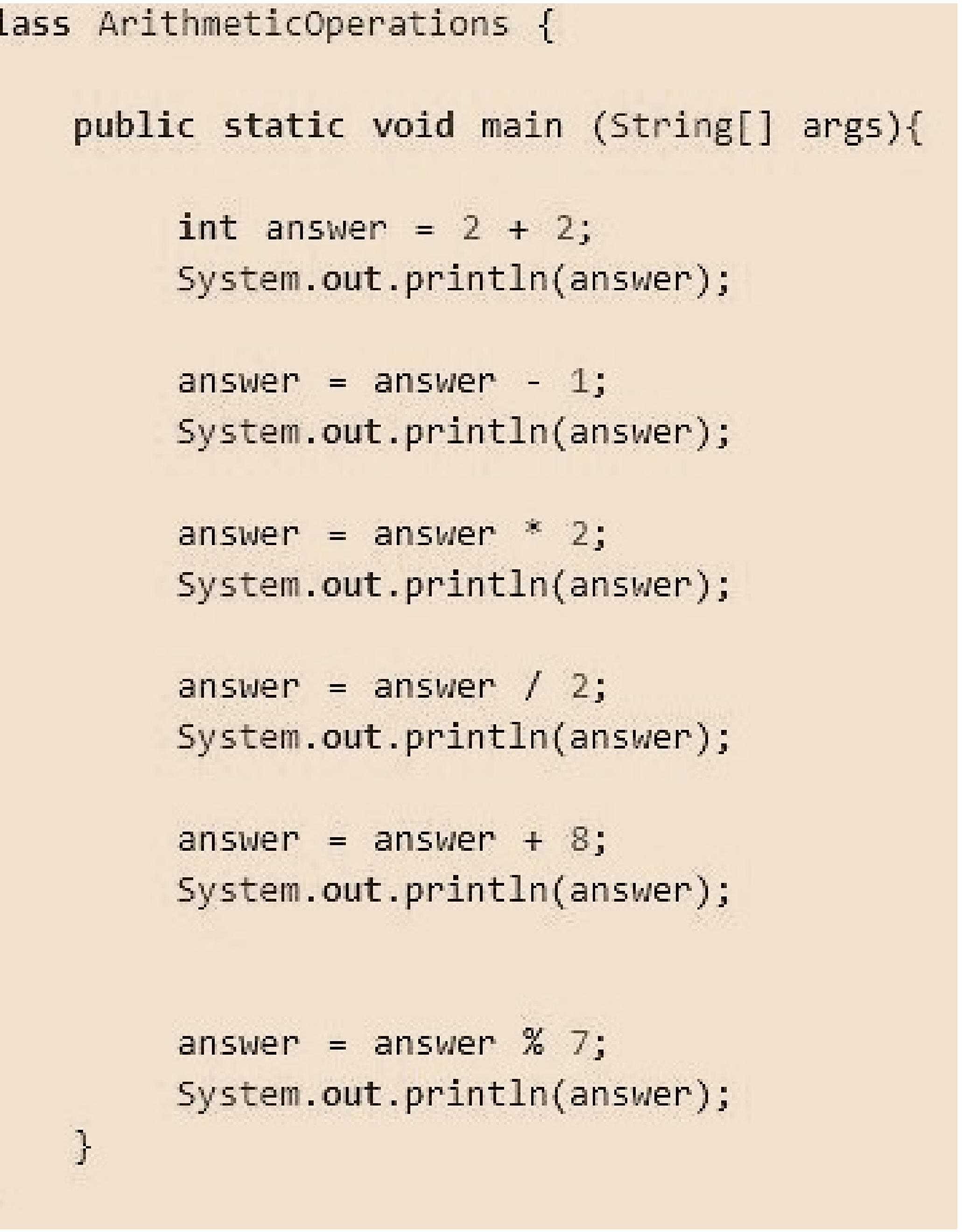
- An operator is a symbol that tells the computer to perforr certain mathematical or logical manipulation.
- Java operators can be divided into following categories:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- conditional operator (Ternary)

❖Arithmetic Operators



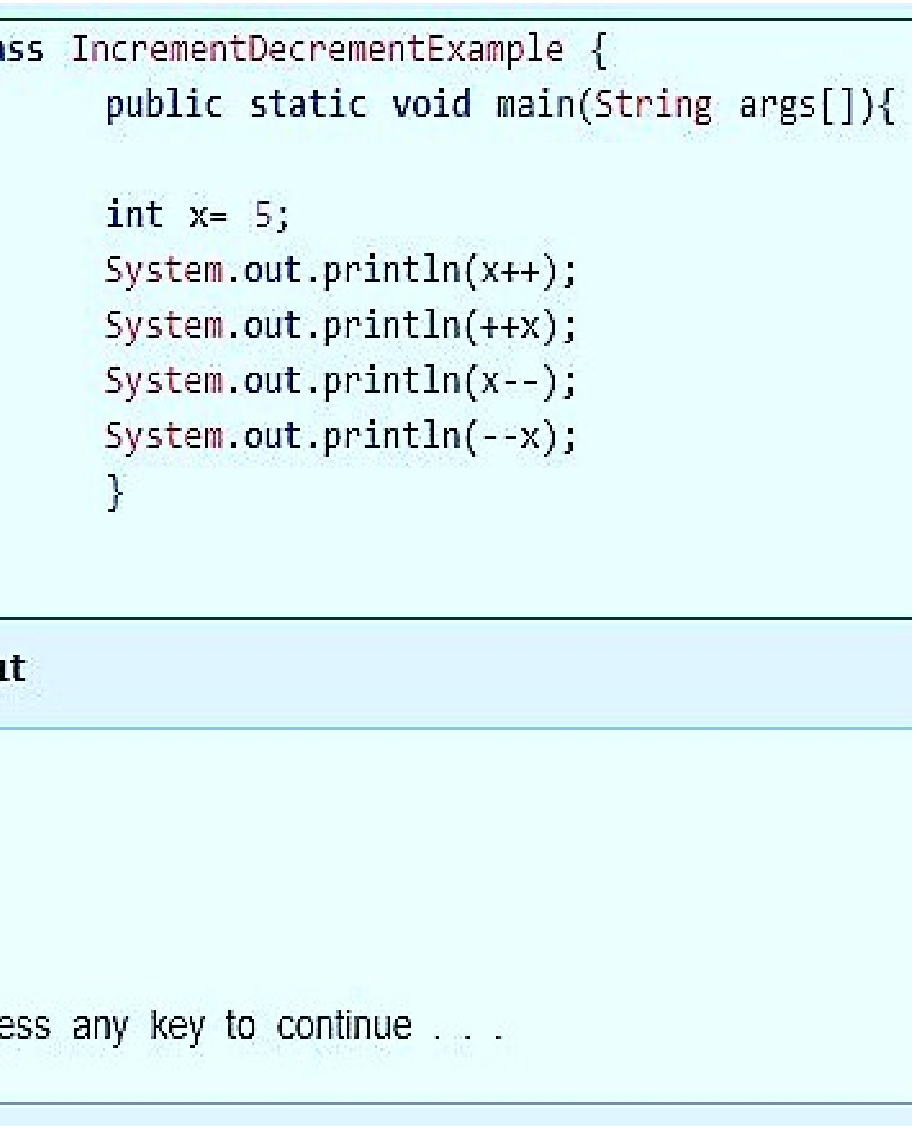
Operator	Description	Example
+ (Addition)	Adds two operands	5 + 10 =15
- (Subtraction)	Subtract second operands from first. Also used to Concatenate two strings	10 - 5 =5
* (Multiplication)	Multiplies values on either side of the operator.	10 * 5 =50
/ (Division)	Divides left-hand operand by right-hand operand.	10 / 5 =2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	5 % 2 =1
++ (Increment)	Increases the value of operand by 1.	2++ gives 3
-- (Decrement)	Decreases the value of operand by 1.	3-- gives 2



```
lass ArithmeticOperations {

    public static void main (String[] args){

        int answer = 2 + 2;
        System.out.println(answer);

        answer = answer - 1;
        System.out.println(answer);

        answer = answer * 2;
        System.out.println(answer);

        answer = answer / 2;
        System.out.println(answer);

        answer = answer + 8;
        System.out.println(answer);


        answer = answer % 7;
        System.out.println(answer);
    }
```

Output

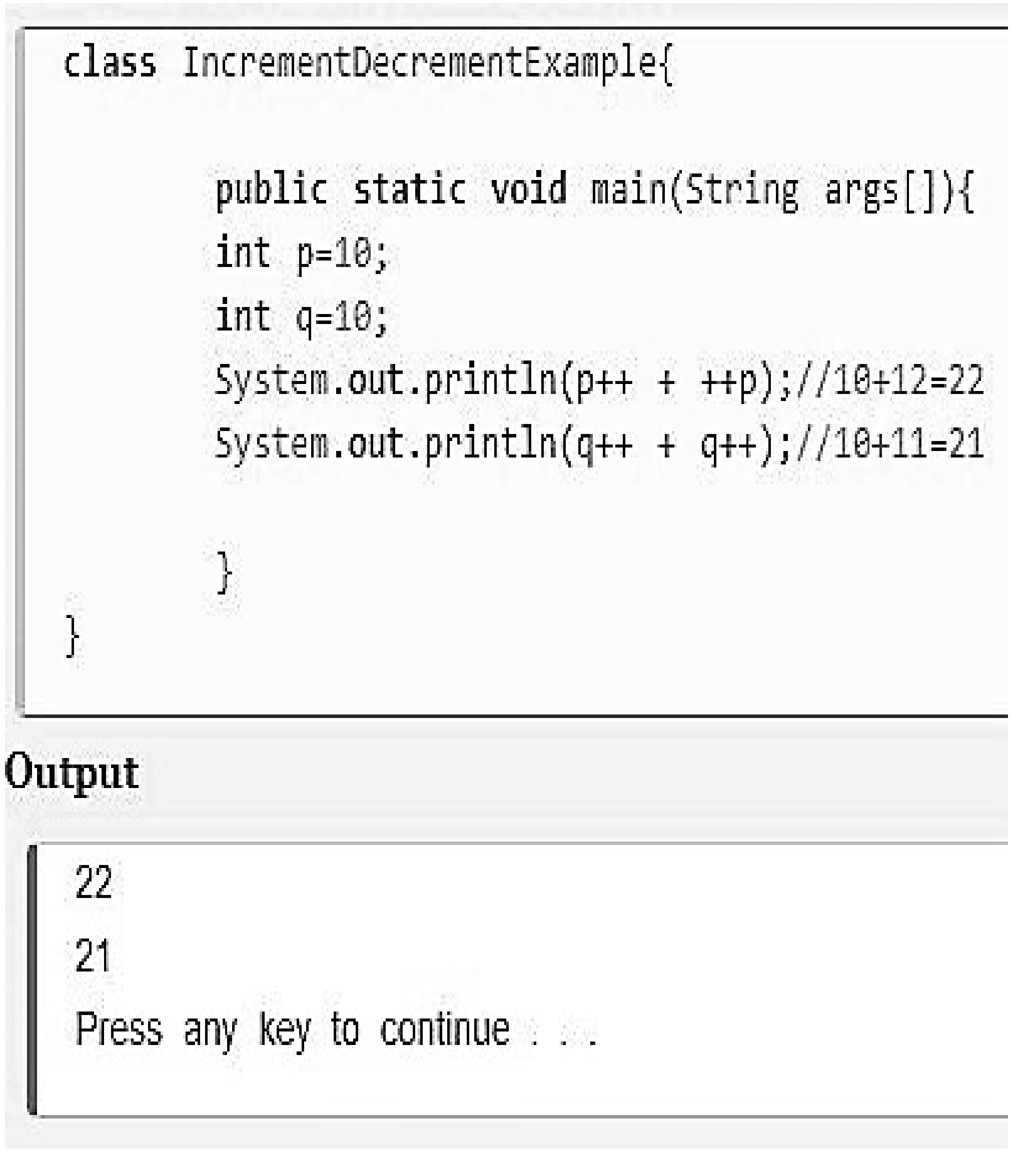
```
4
3
6
3
11
4
```



```
ss IncrementDecrementExample {
        public static void main(String args[]){

        int x= 5;
        System.out.println(x++);
        System.out.println(++x);
        System.out.println(x--);
        System.out.println(--x);
        }
```

ut

```
ess any key to continue . . .
```



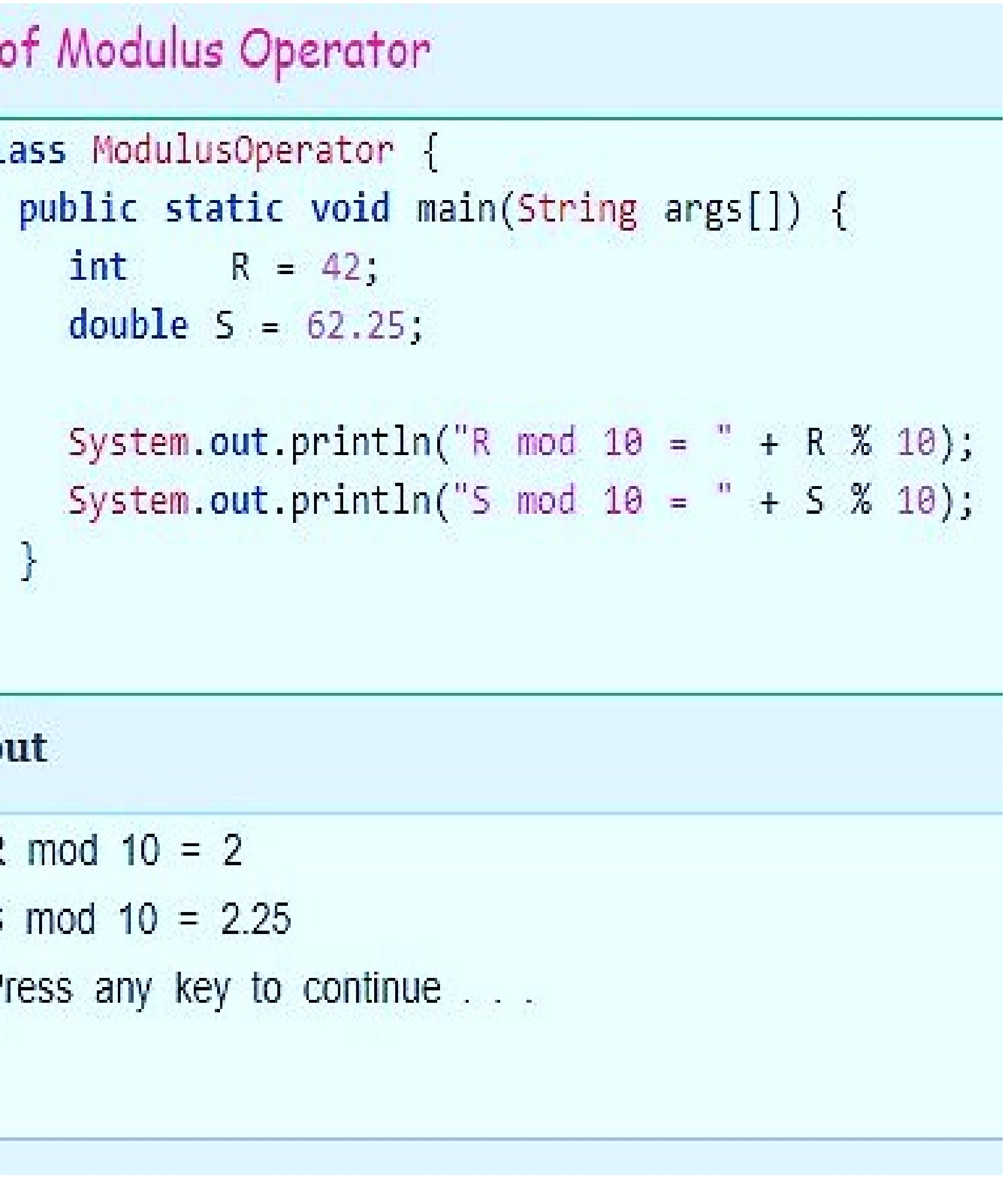
```
class IncrementDecrementExample{

        public static void main(String args[]){
        int p=10;
        int q=10;
        System.out.println(p++ + ++p);//10+12=22
        System.out.println(q++ + q++);//10+11=21

        }
}
```

Output

```
22
21
Press any key to continue . . .
```

+ is Use –Then - Change

++x is Change – Then - Use

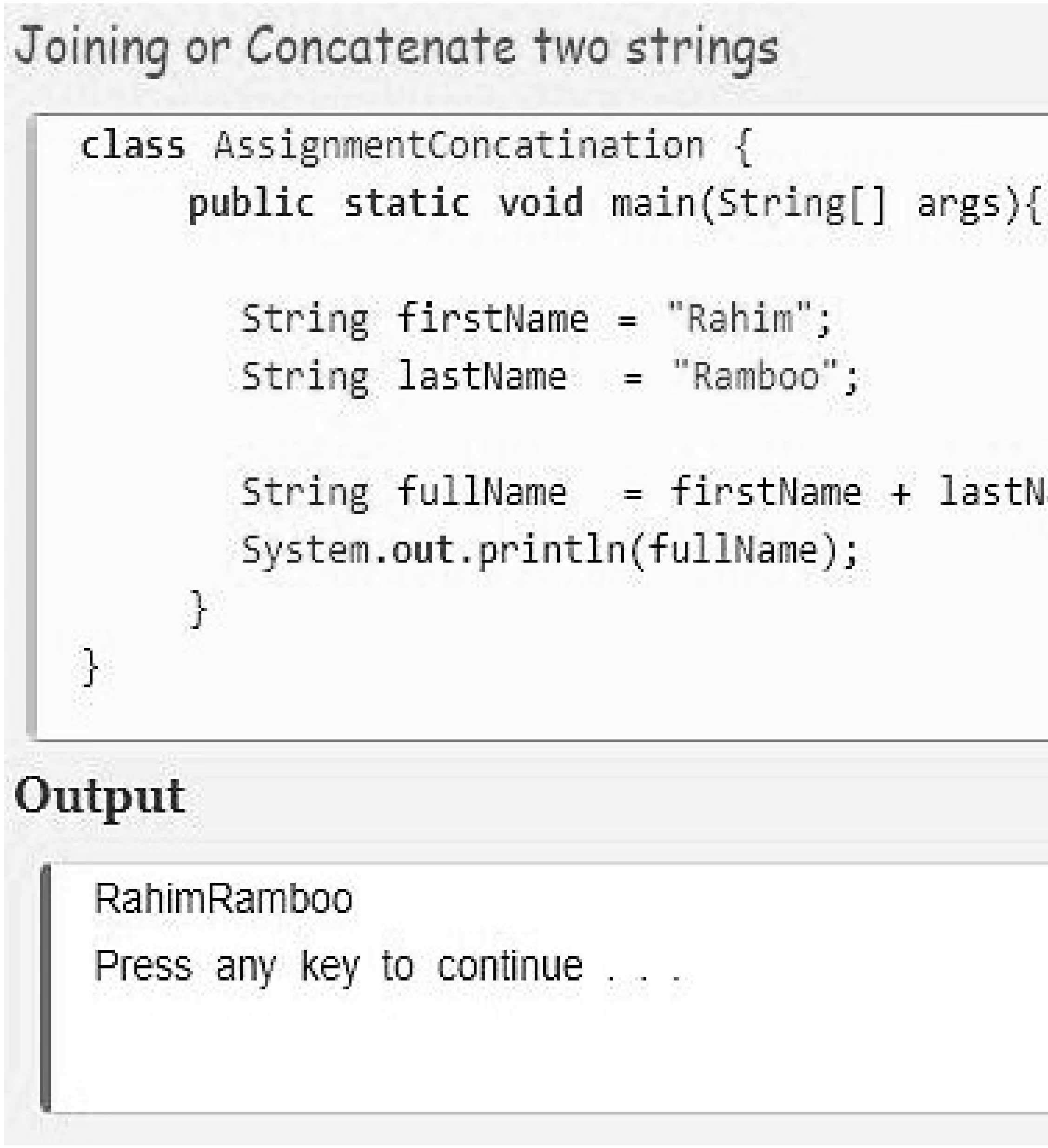


of Modulus Operator

```
lass ModulusOperator {
 public static void main(String args[]) {
   int     R = 42;
   double S = 62.25;

   System.out.println("R mod 10 = " + R % 10);
   System.out.println("S mod 10 = " + S % 10);
 }
```

ut

```
 mod 10 = 2
 mod 10 = 2.25
ress any key to continue . . .
```



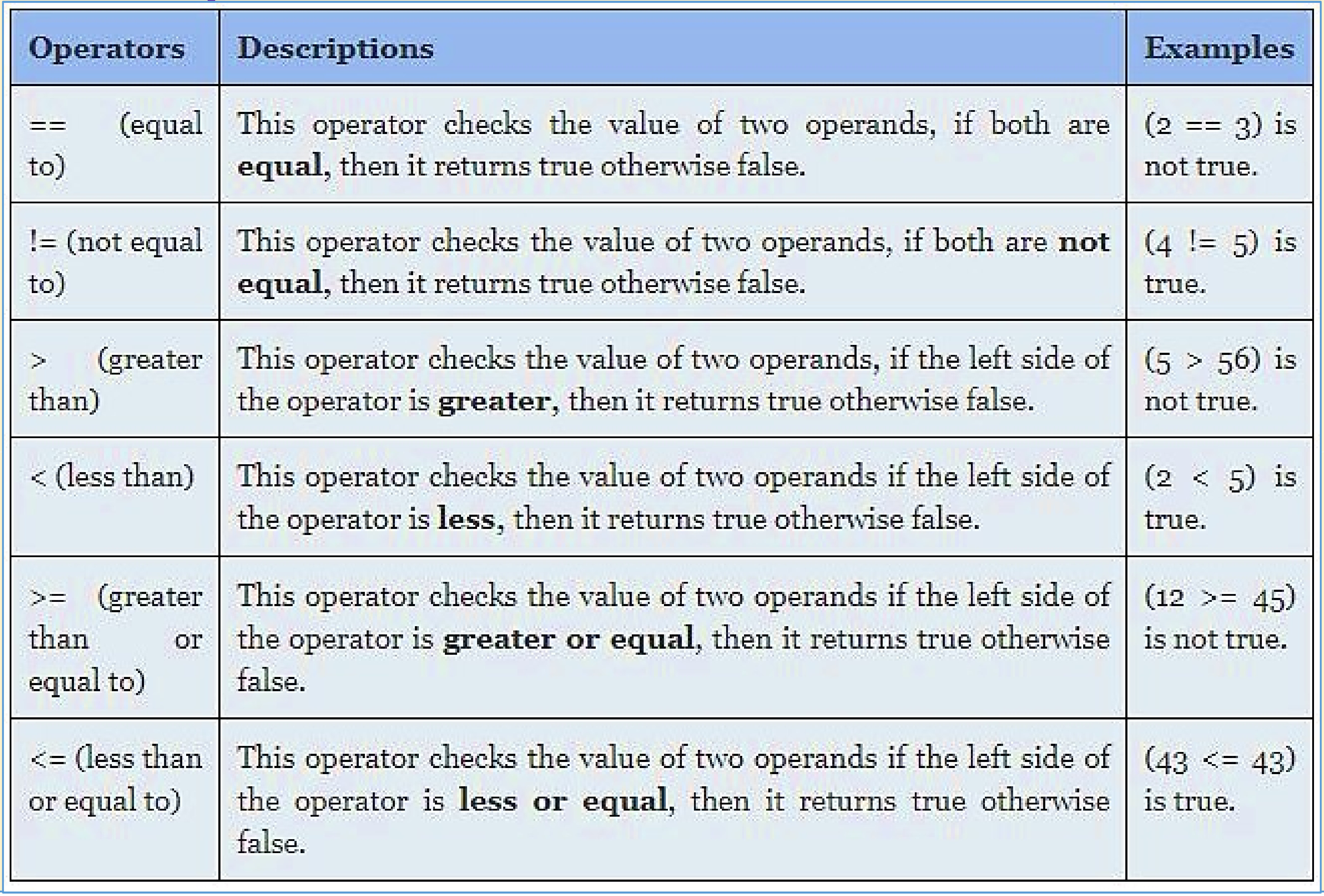
Joining or Concatenate two strings

```
class AssignmentConcatination {
     public static void main(String[] args){

       String firstName = "Rahim";
       String lastName  = "Ramboo";

       String fullName  = firstName + lastN
       System.out.println(fullName);
     }
}
```

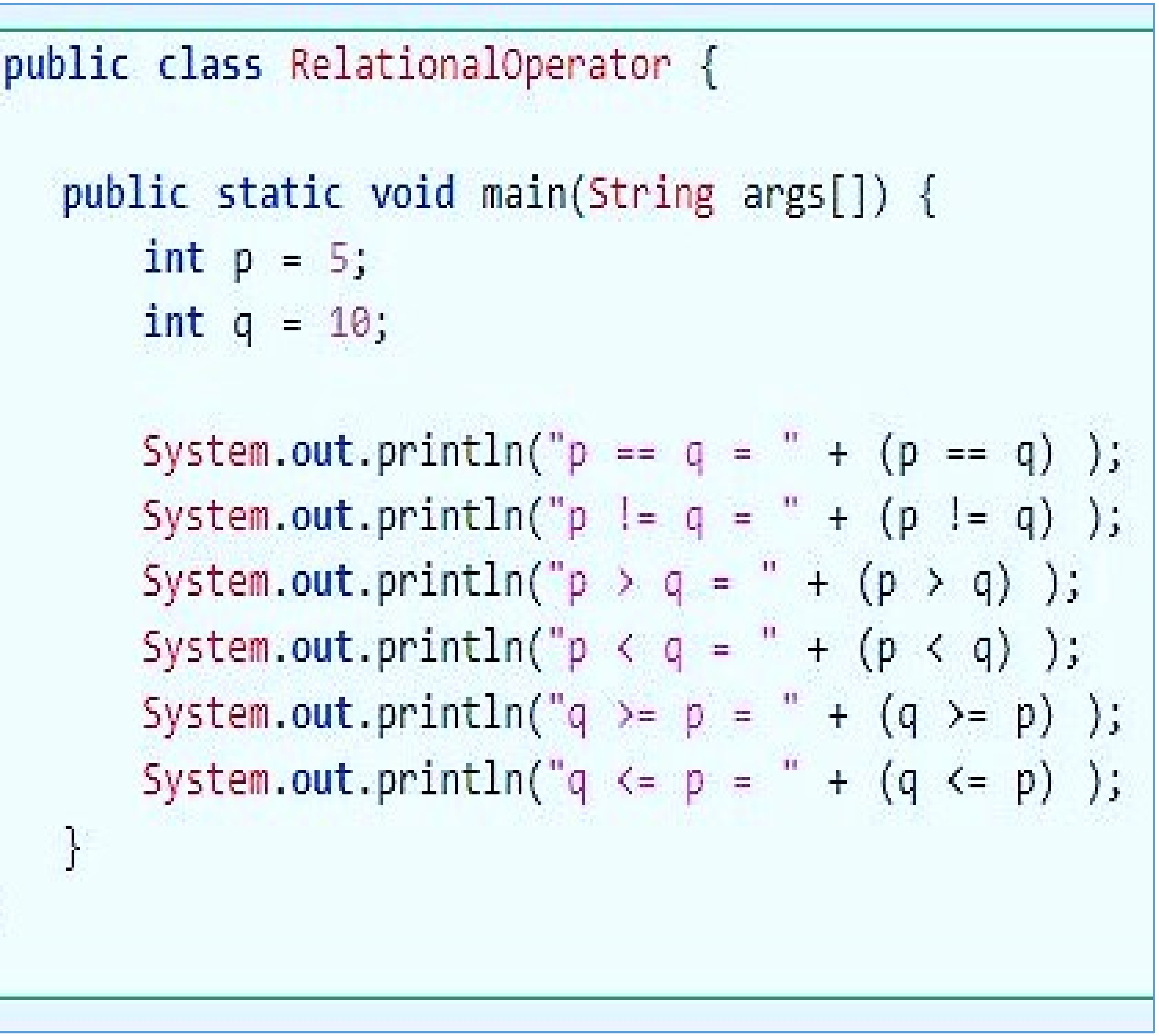
Output

```
RahimRamboo
Press any key to continue . . .
```

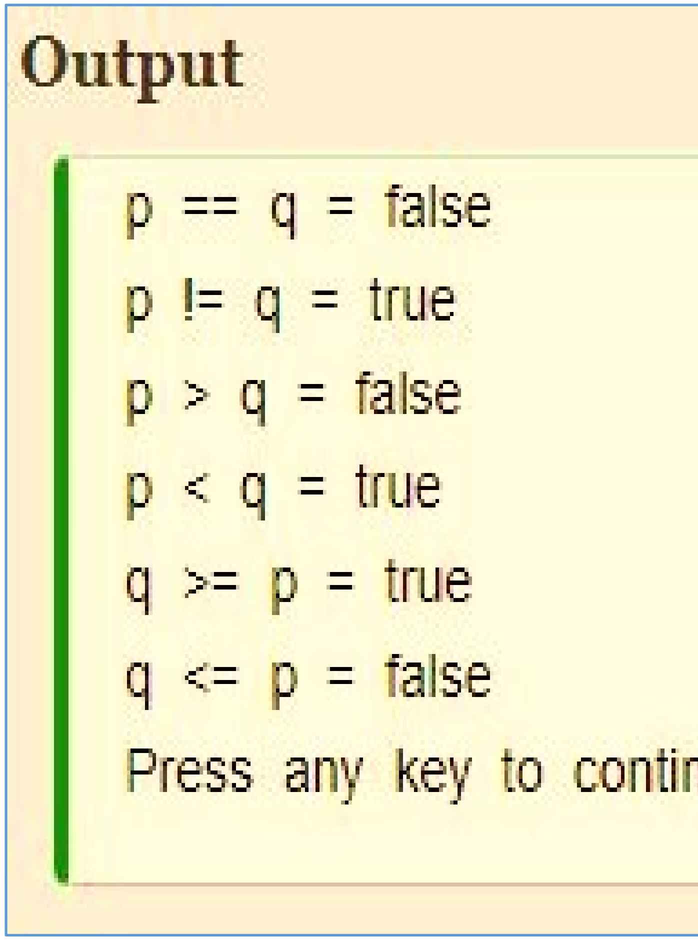
❖Relational Operators



Operators	Descriptions	Examples
== (equal to)	This operator checks the value of two operands, if both are **equal**, then it returns true otherwise false.	(2 == 3) is not true.
!= (not equal to)	This operator checks the value of two operands, if both are **not equal**, then it returns true otherwise false.	(4 != 5) is true.
> (greater than)	This operator checks the value of two operands, if the left side of the operator is **greater**, then it returns true otherwise false.	(5 > 56) is not true.
< (less than)	This operator checks the value of two operands if the left side of the operator is **less**, then it returns true otherwise false.	(2 < 5) is true.
>= (greater than or equal to)	This operator checks the value of two operands if the left side of the operator is **greater or equal**, then it returns true otherwise false.	(12 >= 45) is not true.
<= (less than or equal to)	This operator checks the value of two operands if the left side of the operator is **less or equal**, then it returns true otherwise false.	(43 <= 43) is true.



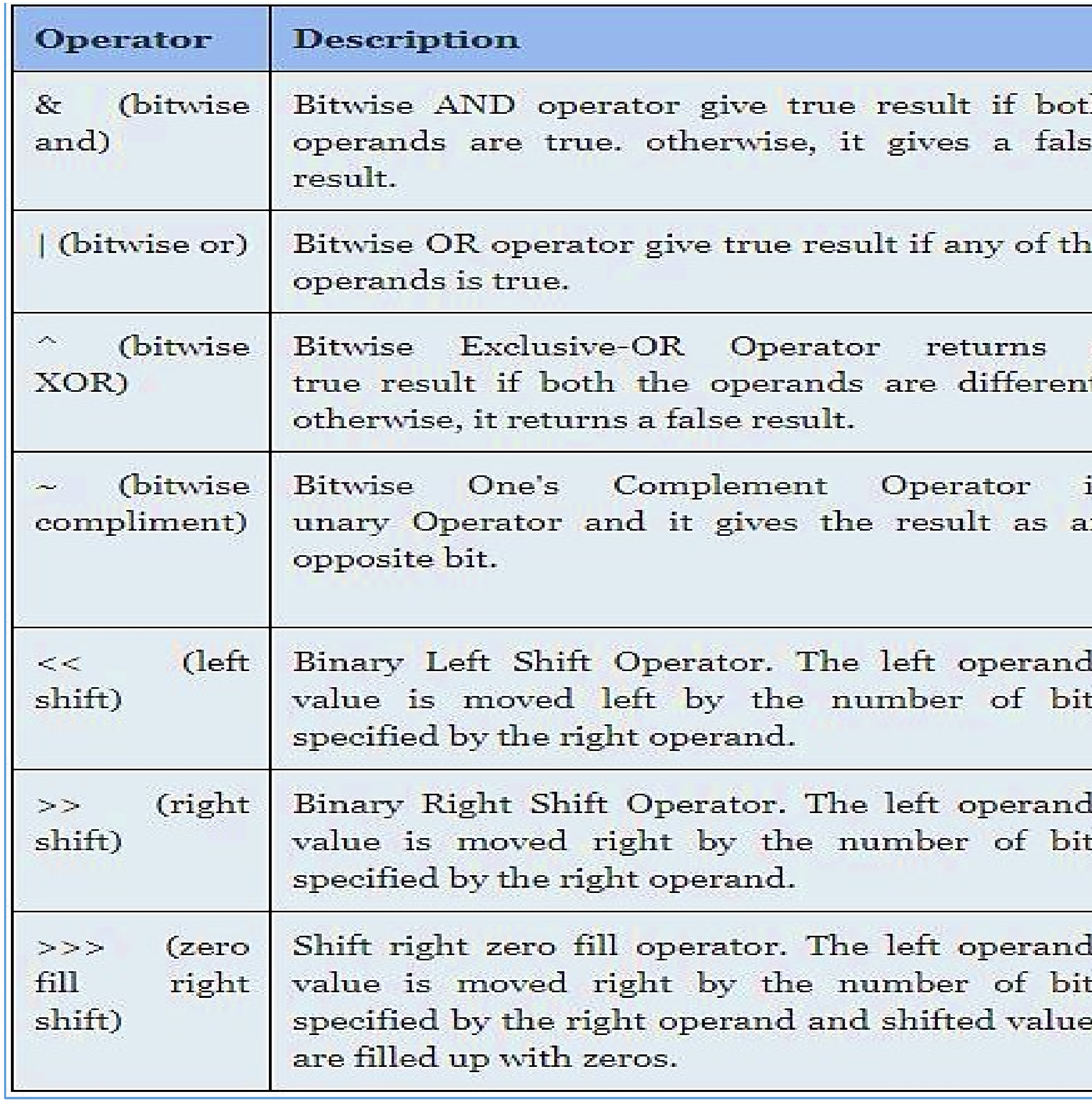
```java
public class RelationalOperator {

  public static void main(String args[]) {
     int p = 5;
     int q = 10;

     System.out.println("p == q = " + (p == q) );
     System.out.println("p != q = " + (p != q) );
     System.out.println("p > q = " + (p > q) );
     System.out.println("p < q = " + (p < q) );
     System.out.println("q >= p = " + (q >= p) );
     System.out.println("q <= p = " + (q <= p) );
  }
```



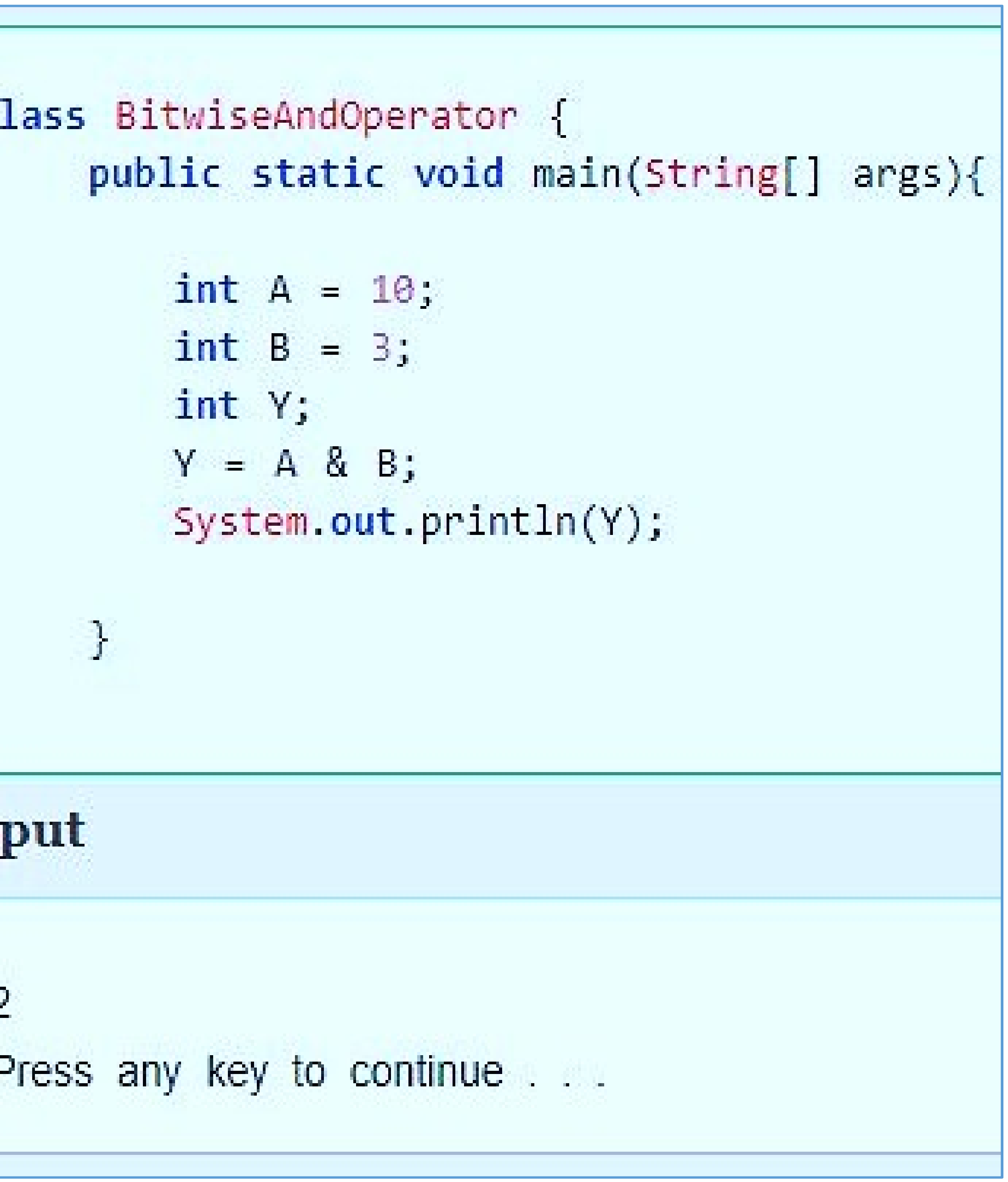
Output

```
p == q = false
p != q = true
p > q = false
p < q = true
q >= p = true
q <= p = false
Press any key to contin
```

Bitwise Operators



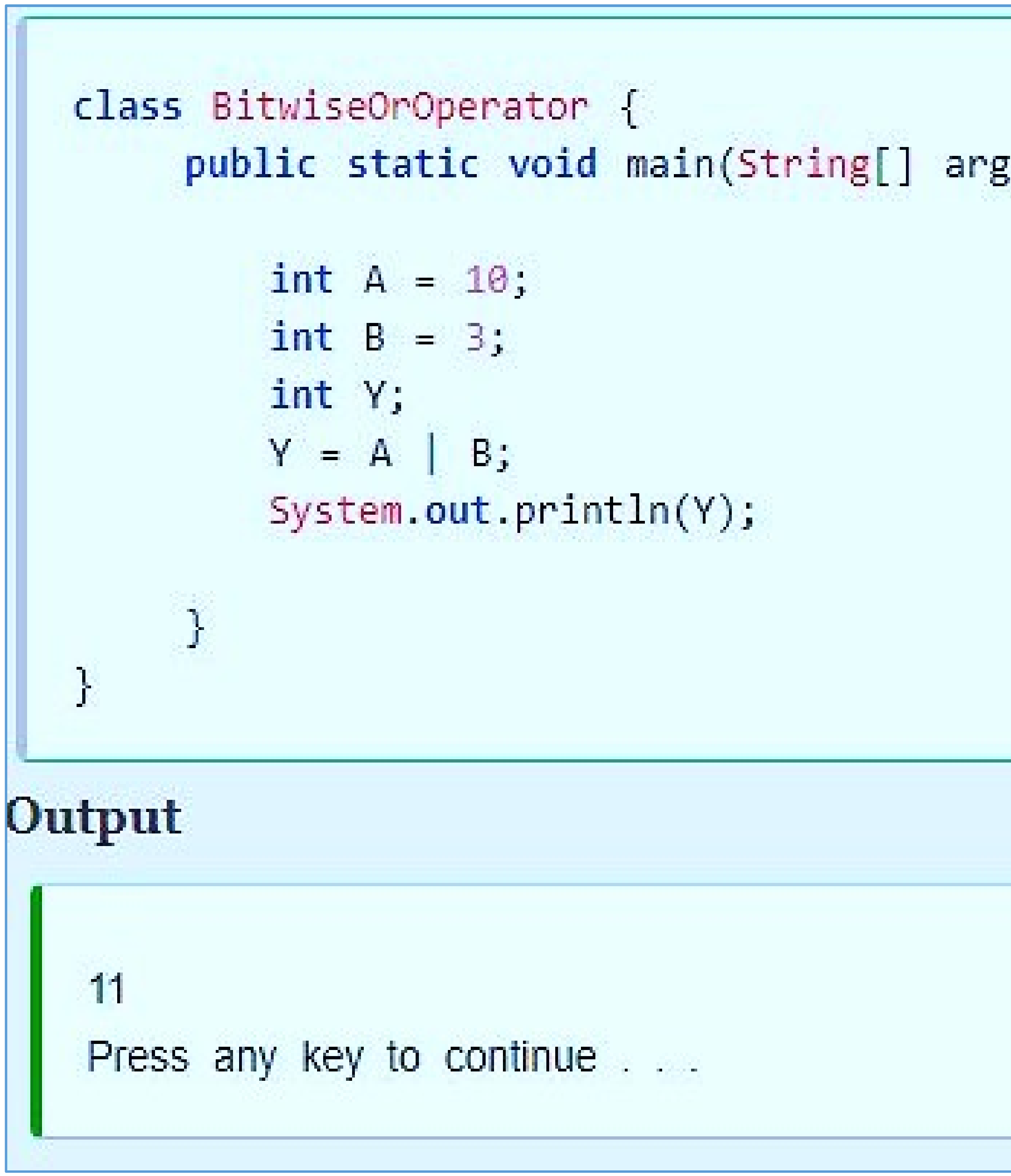
Operator	Description
& (bitwise and)	Bitwise AND operator give true result if bot operands are true. otherwise, it gives a fals result.
| (bitwise or)	Bitwise OR operator give true result if any of th operands is true.
^ (bitwise XOR)	Bitwise Exclusive-OR Operator returns true result if both the operands are differen otherwise, it returns a false result.
~ (bitwise compliment)	Bitwise One's Complement Operator i unary Operator and it gives the result as a opposite bit.
<< (left shift)	Binary Left Shift Operator. The left operand value is moved left by the number of bit specified by the right operand.
>> (right shift)	Binary Right Shift Operator. The left operand value is moved right by the number of bit specified by the right operand.
>>> (zero fill right shift)	Shift right zero fill operator. The left operand value is moved right by the number of bit specified by the right operand and shifted value are filled up with zeros.



```
lass BitwiseAndOperator {
    public static void main(String[] args){

        int A = 10;
        int B = 3;
        int Y;
        Y = A & B;
        System.out.println(Y);

    }
```

put

```
2
Press any key to continue . . .
```

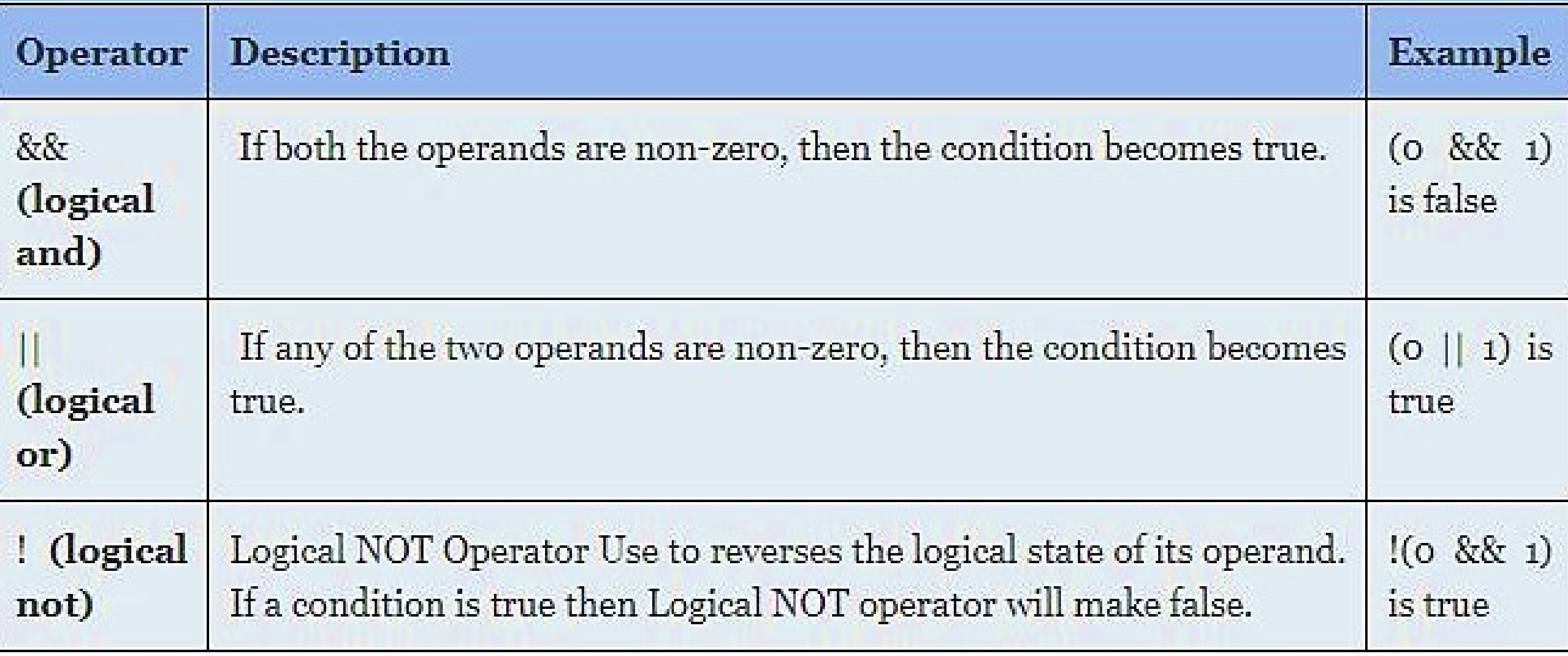


```
class BitwiseOrOperator {
    public static void main(String[] arg

        int A = 10;
        int B = 3;
        int Y;
        Y = A | B;
        System.out.println(Y);

    }
}
```

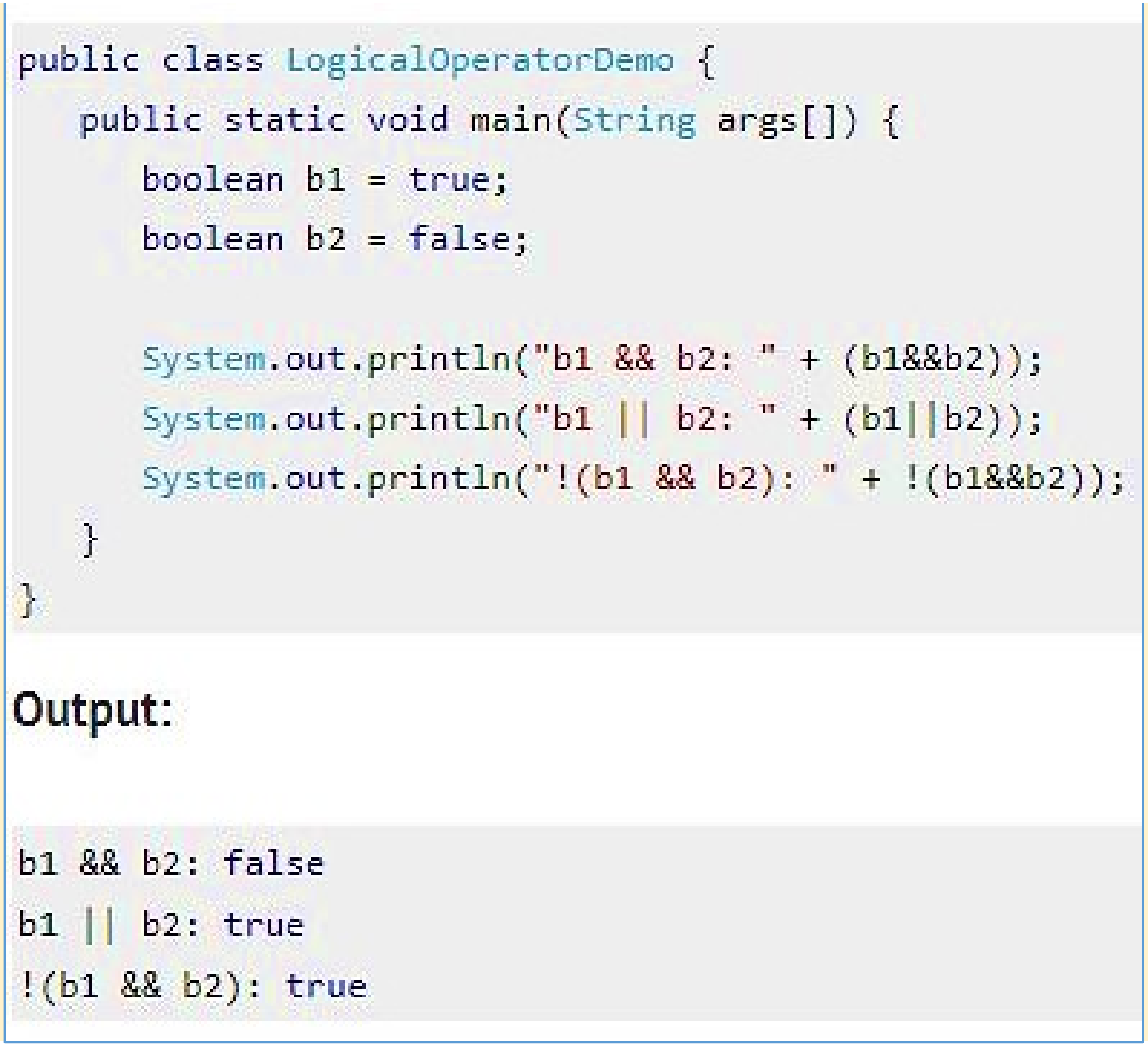
Output

```
11
Press any key to continue . . .
```

❖Logical Operators



Operator	Description	Example
&& (logical and)	If both the operands are non-zero, then the condition becomes true.	(0 && 1) is false
|| (logical or)	If any of the two operands are non-zero, then the condition becomes true.	(0 || 1) is true
! (logical not)	Logical NOT Operator Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(0 && 1) is true

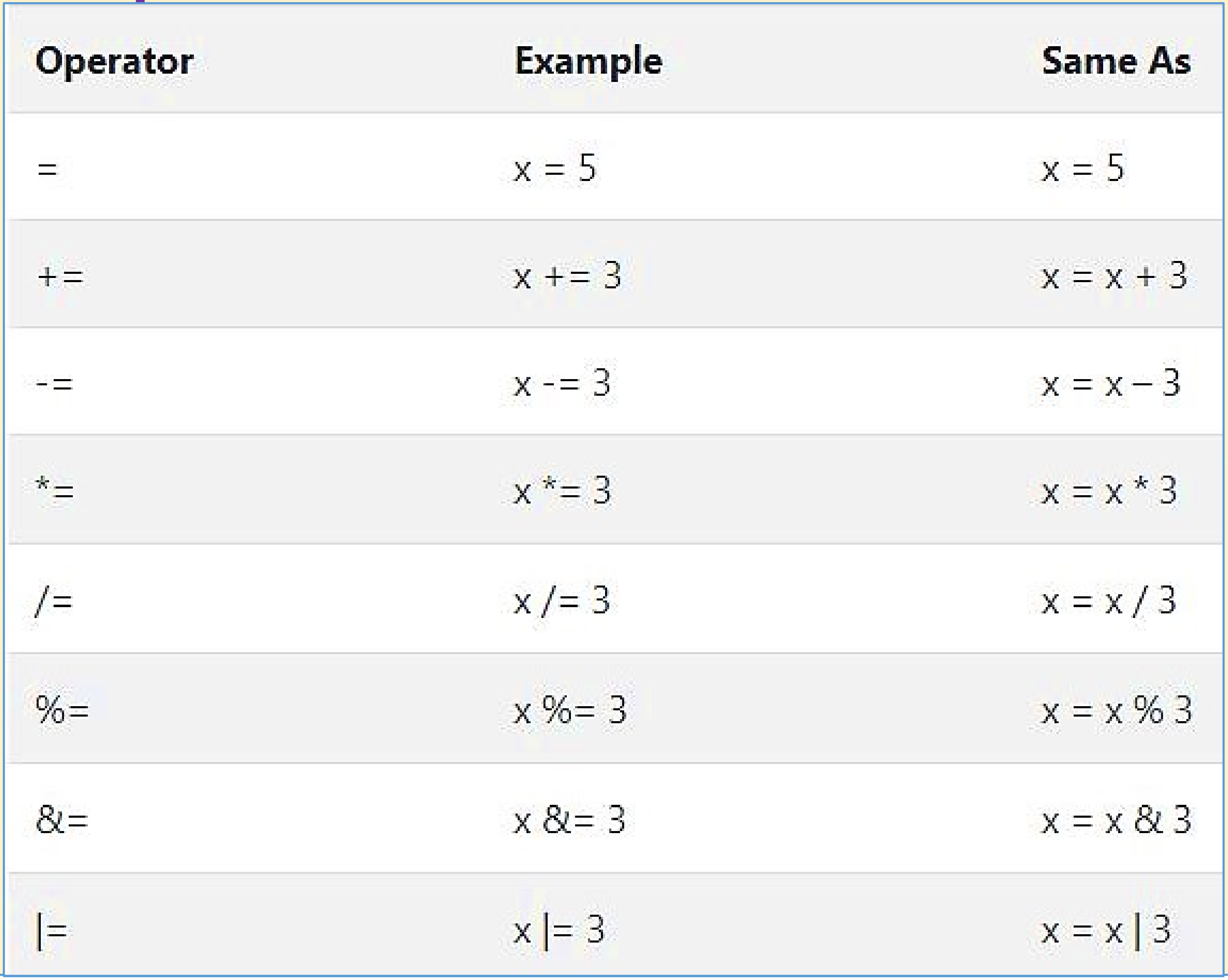


```java
public class LogicalOperatorDemo {
   public static void main(String args[]) {
      boolean b1 = true;
      boolean b2 = false;

      System.out.println("b1 && b2: " + (b1&&b2));
      System.out.println("b1 || b2: " + (b1||b2));
      System.out.println("!(b1 && b2): " + !(b1&&b2));
   }
}
```

Output:

```
b1 && b2: false
b1 || b2: true
!(b1 && b2): true
```

❖Assignment Operators

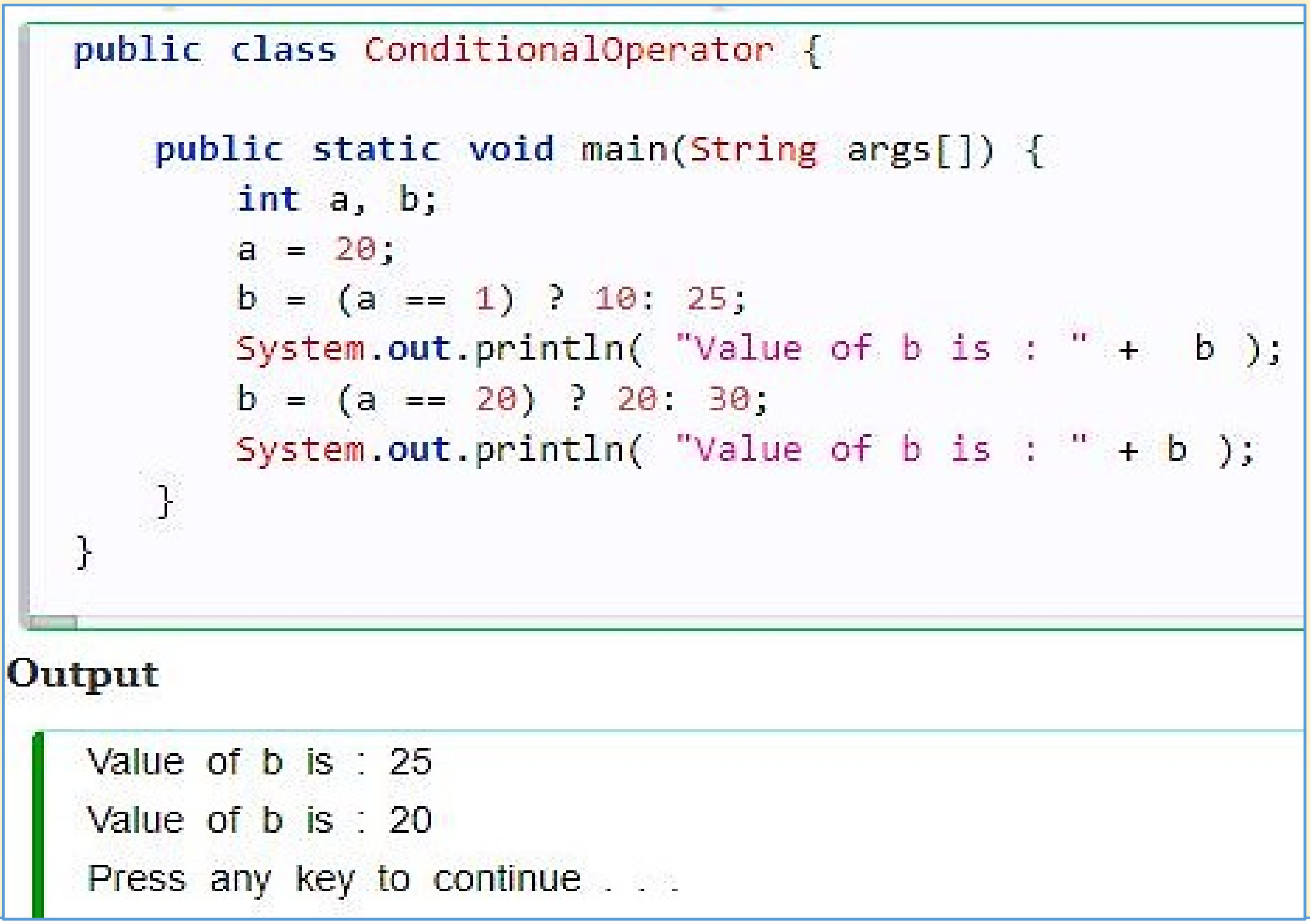


Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x – 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3

❖conditional Operator / Ternary Operator (? :)

Expression1 ? Expression2 : Expression3

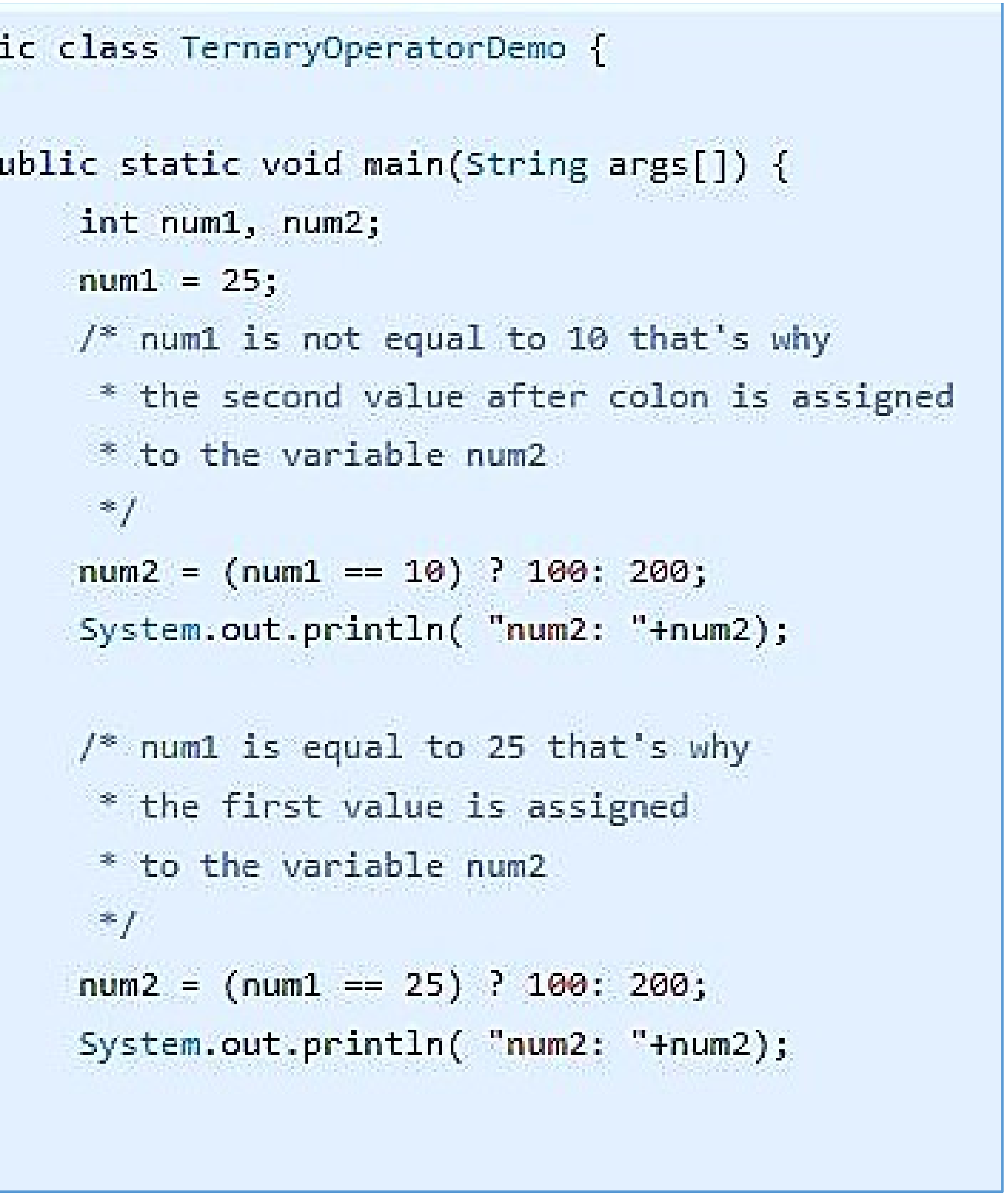
Expression ? value if true : value if false



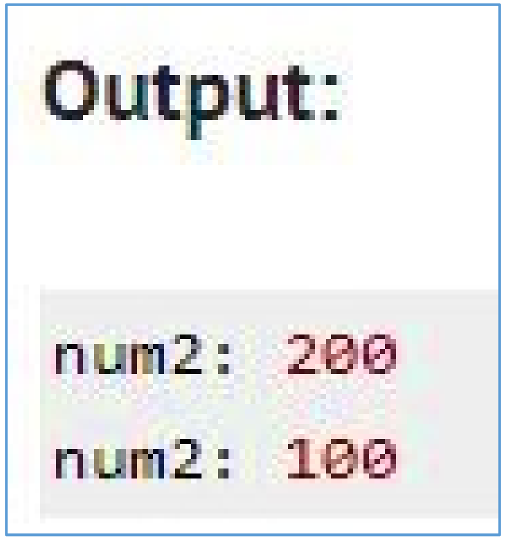
```
public class ConditionalOperator {

   public static void main(String args[]) {
      int a, b;
      a = 20;
      b = (a == 1) ? 10: 25;
      System.out.println( "Value of b is : " +  b );
      b = (a == 20) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

Output

```
Value of b is : 25
Value of b is : 20
Press any key to continue . . .
```



```
ic class TernaryOperatorDemo {

ublic static void main(String args[]) {
    int num1, num2;
    num1 = 25;
    /* num1 is not equal to 10 that's why
     * the second value after colon is assigned
     * to the variable num2
     */
    num2 = (num1 == 10) ? 100: 200;
    System.out.println( "num2: "+num2);

    /* num1 is equal to 25 that's why
     * the first value is assigned
     * to the variable num2
     */
    num2 = (num1 == 25) ? 100: 200;
    System.out.println( "num2: "+num2);
```



Output:

```
num2: 200
num2: 100
```

Operator Precedence

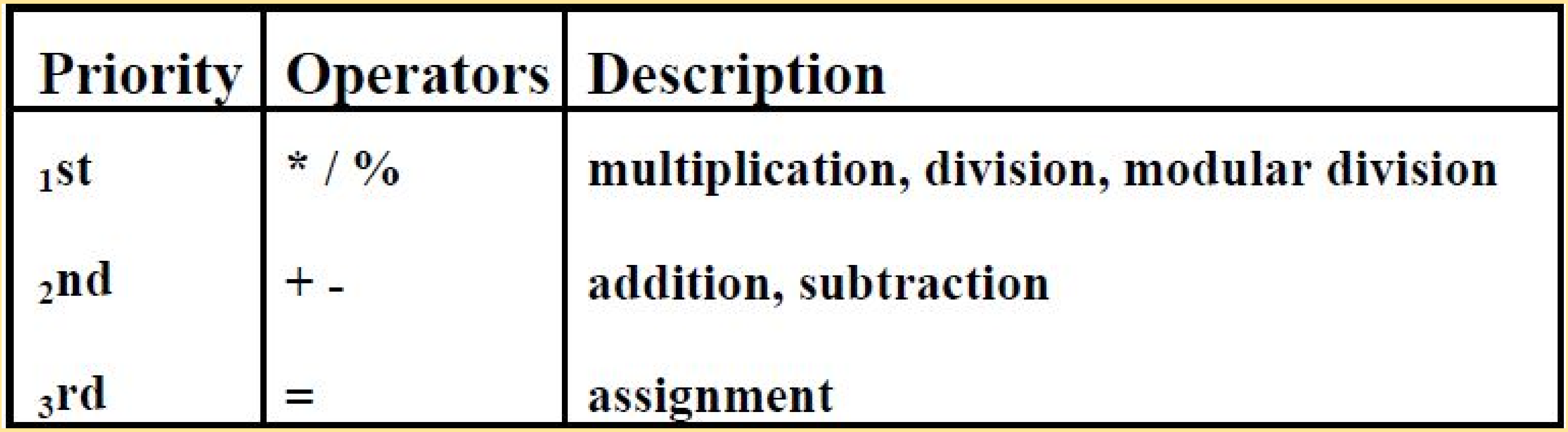
- Evaluate 2*x-3*y ?

  (2x)-(3y) or 2(x-3y) which one is correct??????

- Evaluate A / B * C

  A / (B * C) or (A / B) * C Which one is correct?????

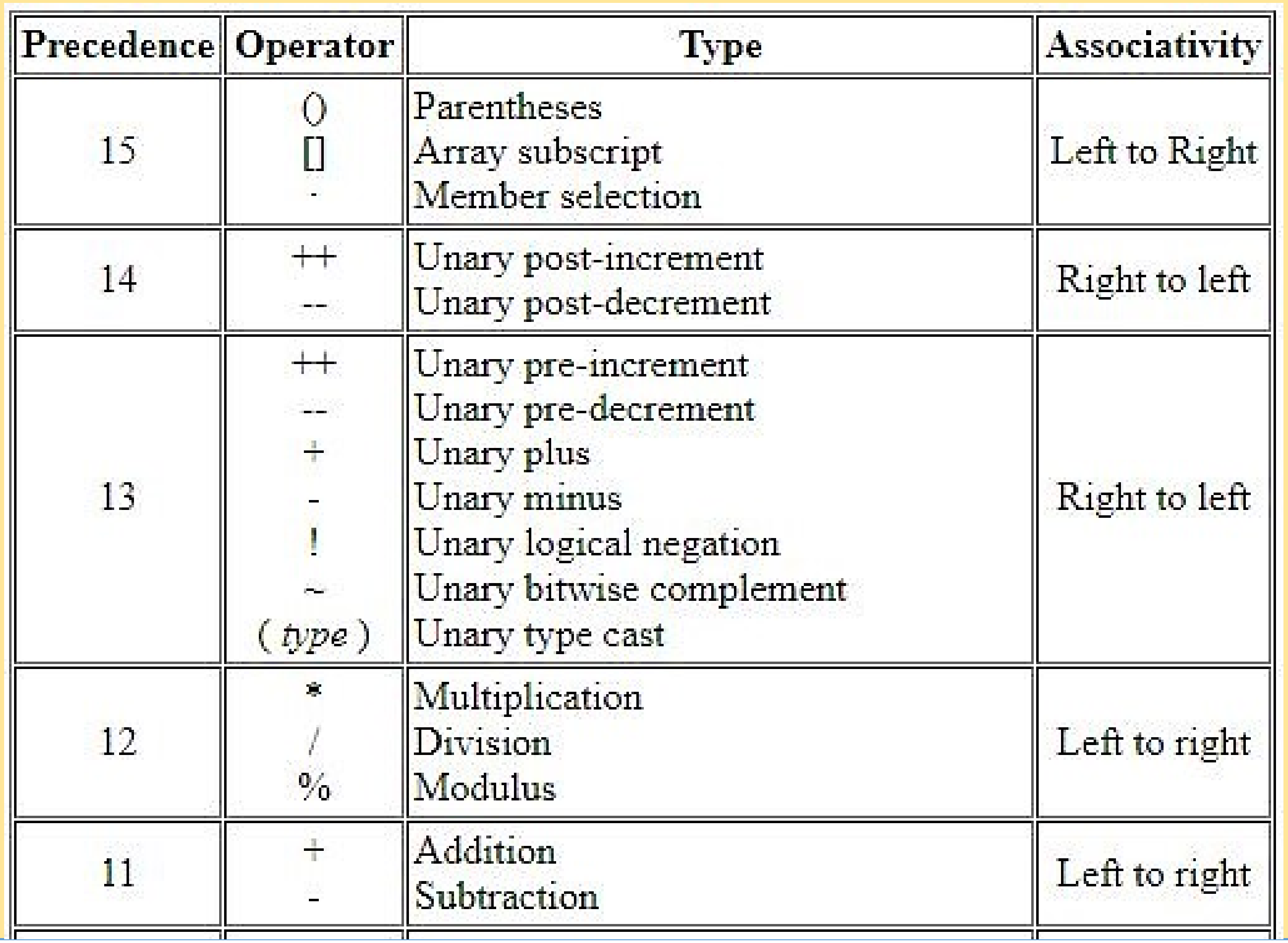
➢ To answer these questions satisfactorily one has to understand the priority or precedence of operations.



Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

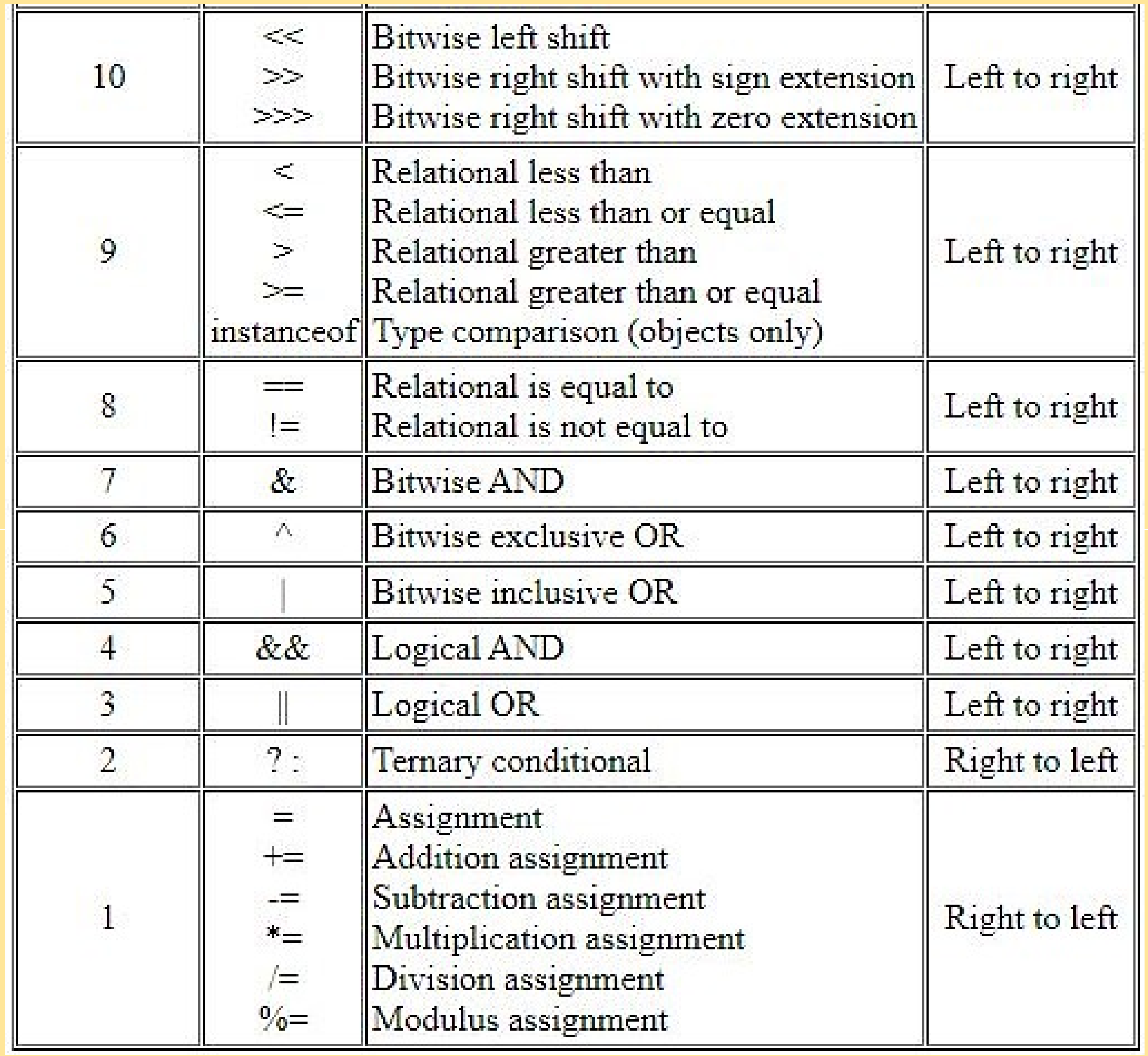
- **Precedence order -** When two operators share an operand th operator with the higher precedence goes first.
- **Associativity -** When an expression has two operators with th same precedence, the expression is evaluated according t its associativity.

➢Larger number means higher precedence



Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right

➢Larger number means higher precedence



10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Evaluate **i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8**

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8	**operation: ***
i = 1 + 4 / 4 + 8 - 2 + 5 / 8	**operation: /**
i = 1 + 1+ 8 - 2 + 5 / 8	**operation: /**
i = 1 + 1+ 8 - 2 + 0	**operation: /**
i = 2 + 8- 2 + 0	**operation: +**
i = 10 - 2 + 0	**operation: +**
i = 8 + 0	**operation : -**
i = 8	**operation: +**

SELECTION STATEMENTS

- Selection statements allow your program to choose differen paths of execution based upon the outcome of an expression o the state of a variable.
- Also called decision making statements
- Java supports various selection statements, like if, if-else anc switch
- There are various types of if statement in java.
- if statement
- if-else statement
- nested if statement
- if-else-if ladder

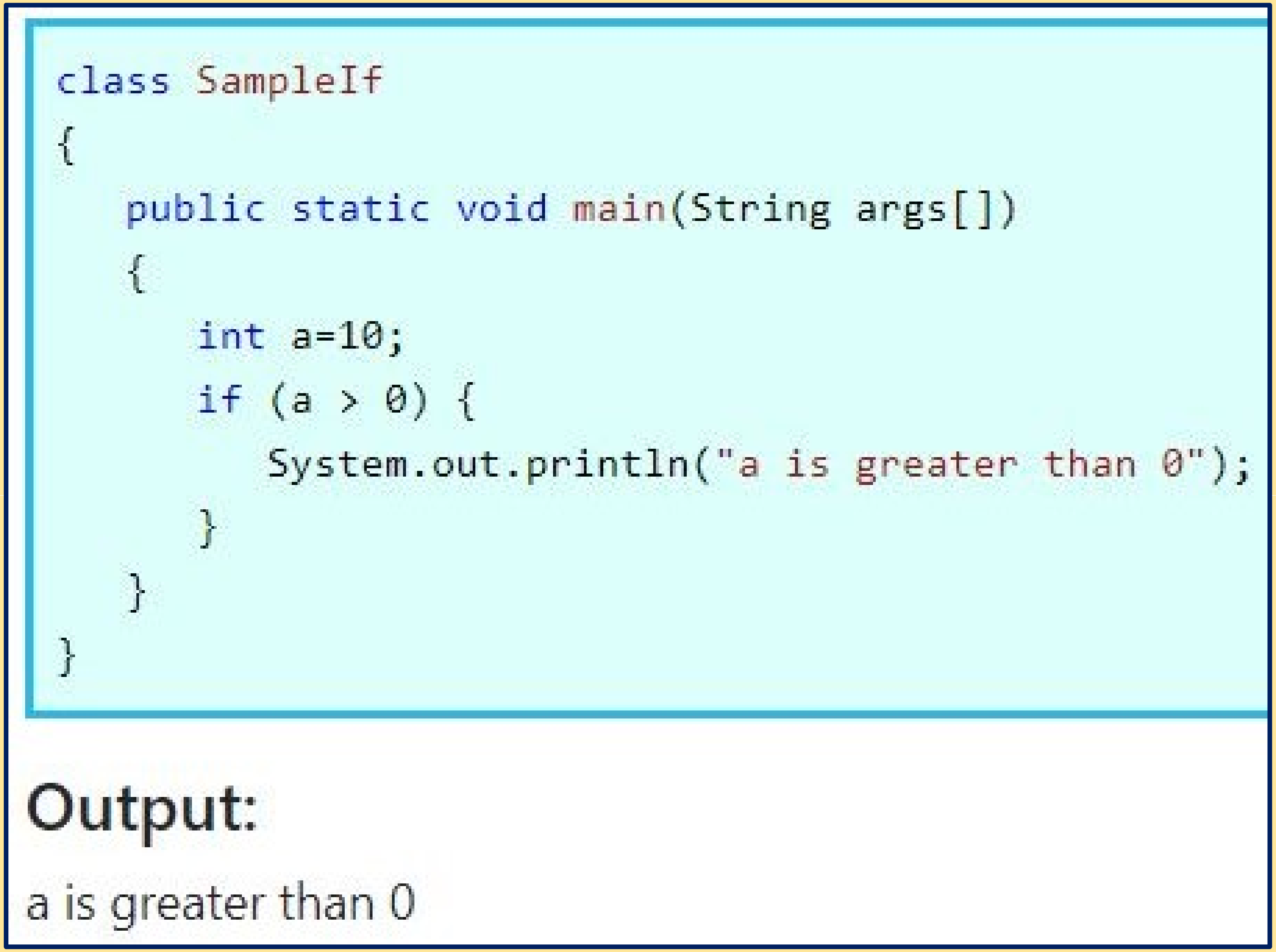
If statement

Use the if statement to specify a block of Java code to b
executed if a condition is true.

ntax

```
if (condition)
     {
          // block of code to be executed if the condition is true
      }
```

ample



```
class SampleIf
{
   public static void main(String args[])
   {
      int a=10;
      if (a > 0) {
         System.out.println("a is greater than 0");
      }
   }
}
```

Output:

```
a is greater than 0
```

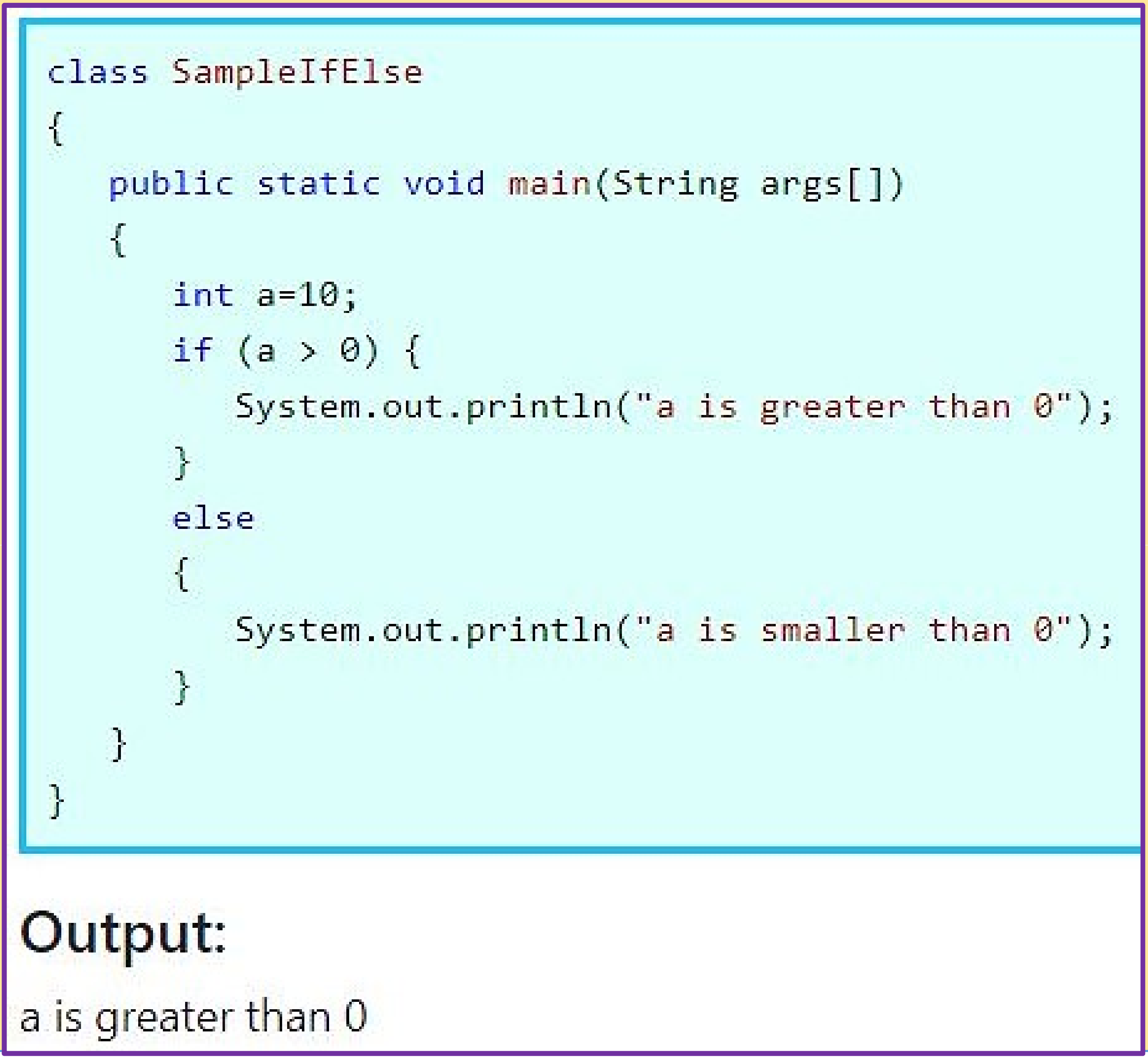
if-else Statement

If-else statement also tests the condition. It executes the if block f condition is true otherwise else block is executed.

ntax

```
 if (condition)
     {
         // block of code to be executed if the condition is true
     }
else
    {
        // block of code to be executed if the condition is false
    }
```

Example



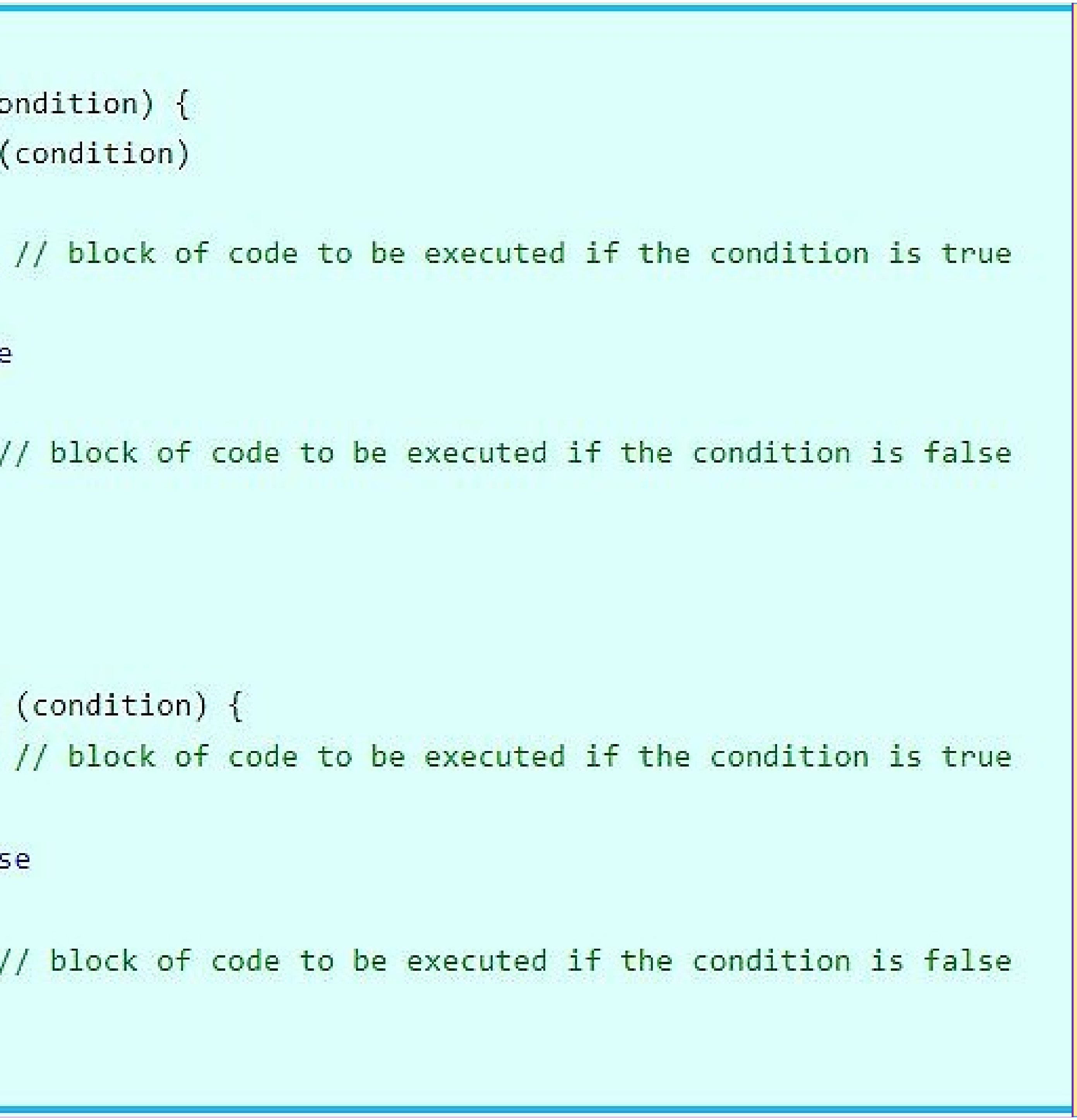
```
class SampleIfElse
{
   public static void main(String args[])
   {
      int a=10;
      if (a > 0) {
         System.out.println("a is greater than 0");
      }
      else
      {
         System.out.println("a is smaller than 0");
      }
   }
}
```

Output:

```
a is greater than 0
```

Nested if else Stateme

Syntax



```
ondition) {
(condition)

 // block of code to be executed if the condition is true

e

// block of code to be executed if the condition is false




 (condition) {
 // block of code to be executed if the condition is true

se

// block of code to be executed if the condition is false
```

```
s SampleNestedIfElse

ublic static void main(String args[])

  int a=10,b=20,c=30;
  if (a>b)
  {
     if (a>c)
     {
         System.out.println("a is greatest.");
     }
     else
     {
          System.out.println("c is greatest.");
     }
 }
 else
 {
    if (b>c)
    {
         System.out.println("b is greatest.");
    }
```

```
         else
         {
              System.out.println("c is greatest."
         }
      }
    }
}
```

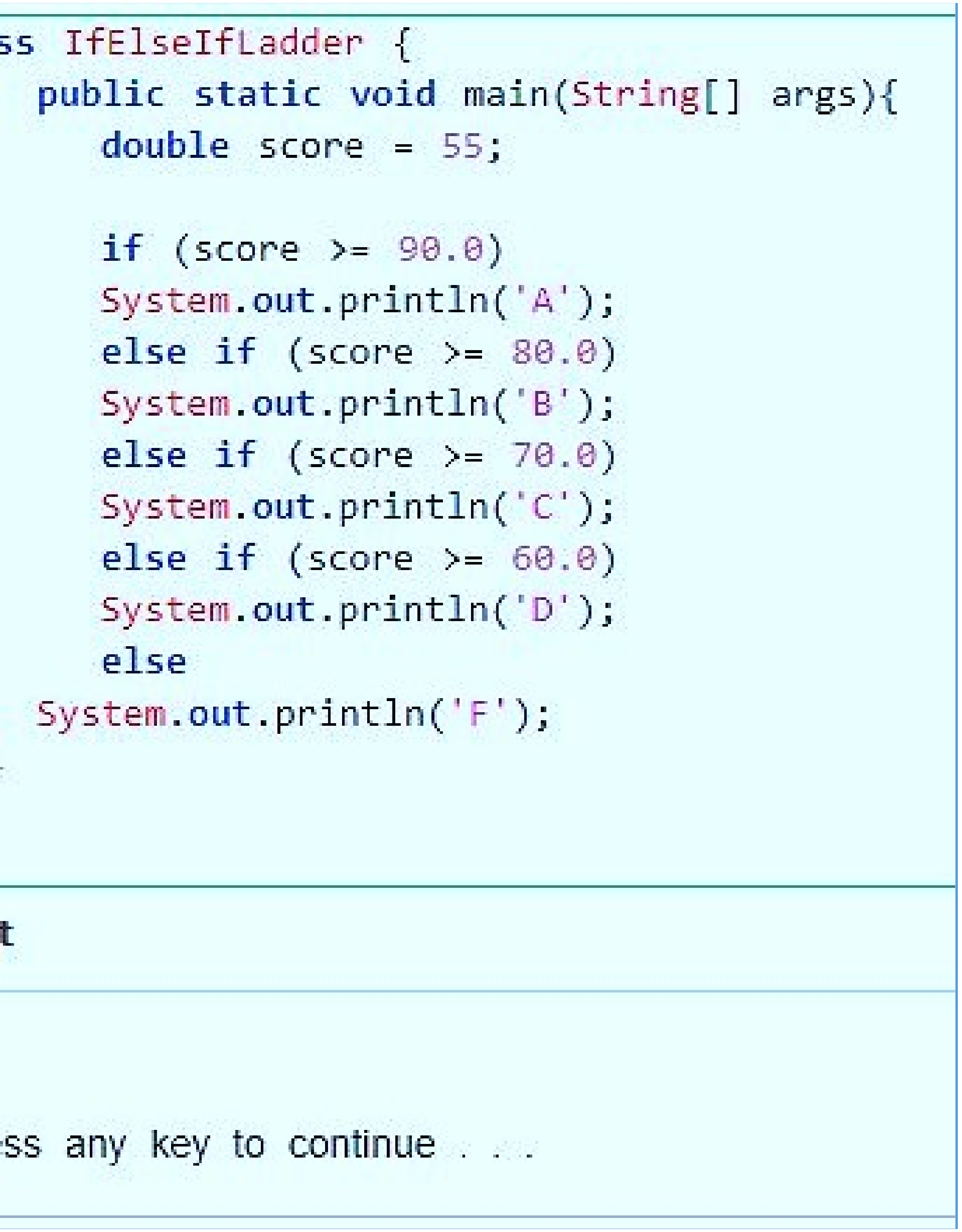
Output:

```
c is greatest.
```

f else if ladder

tax

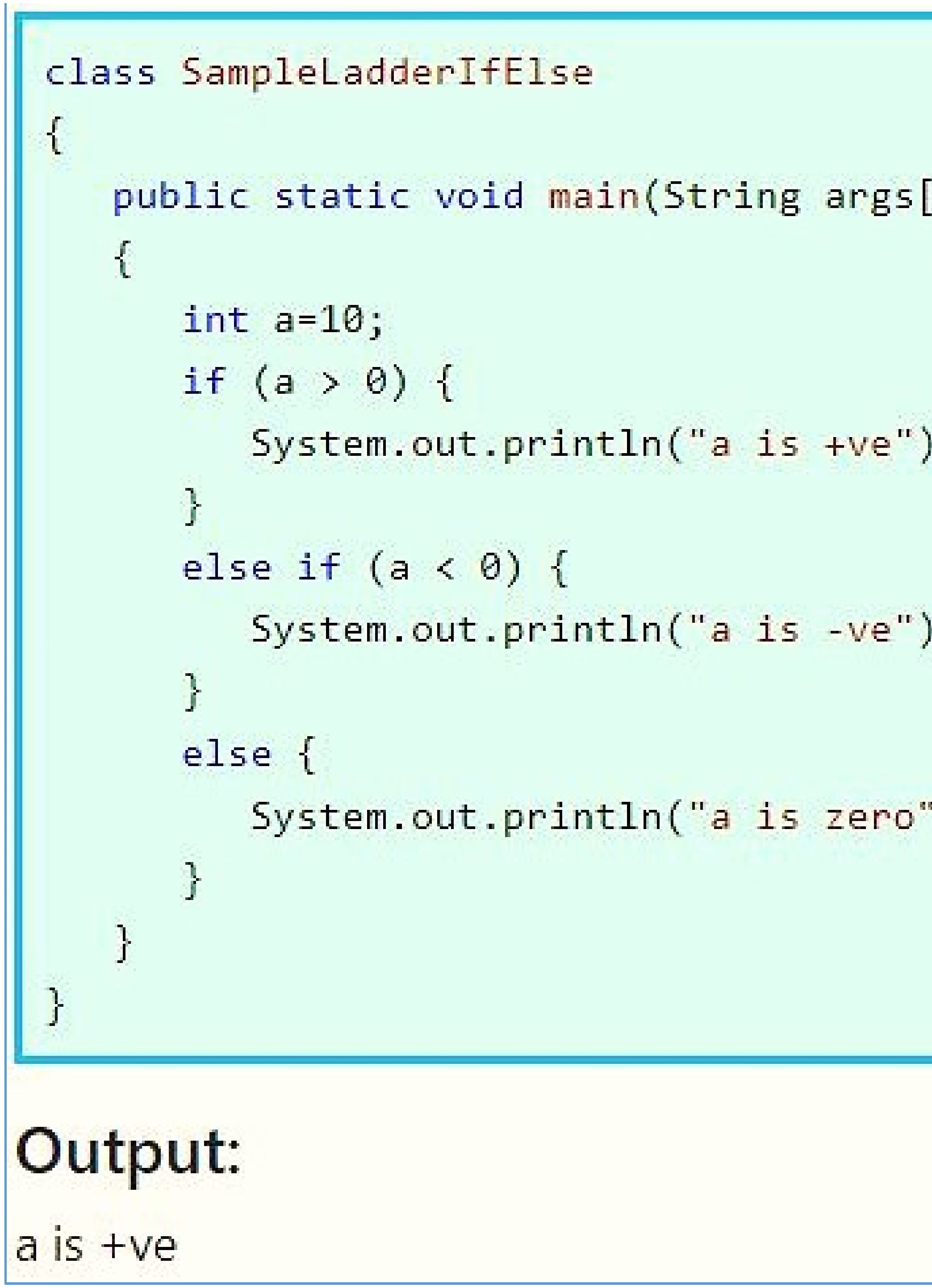
```
f (condition)
 {
   // block of code to be executed if the condition is true
 }
lse if (condition)
 {
   // block of code to be executed if the condition is true
 }
lse
{
  // block of code to be executed if the condition is true
}
```



```
ss IfElseIfLadder {
 public static void main(String[] args){
    double score = 55;

    if (score >= 90.0)
    System.out.println('A');
    else if (score >= 80.0)
    System.out.println('B');
    else if (score >= 70.0)
    System.out.println('C');
    else if (score >= 60.0)
    System.out.println('D');
    else
 System.out.println('F');
```

t

```
ss any key to continue . . .
```

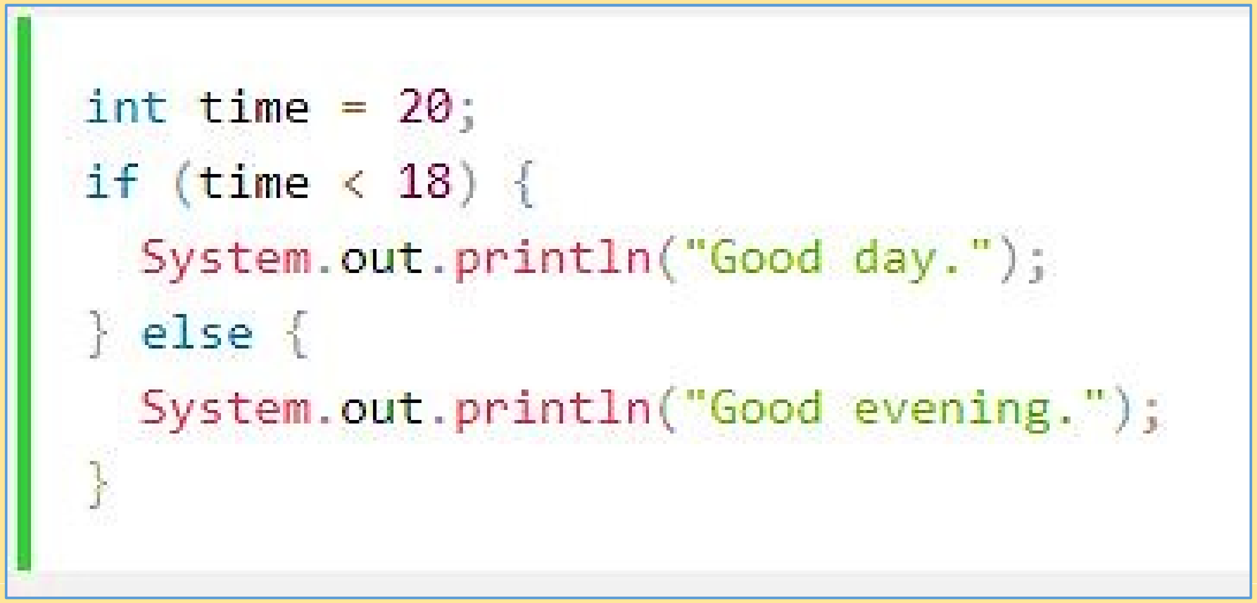


```
class SampleLadderIfElse
{
   public static void main(String args[
   {
      int a=10;
      if (a > 0) {
         System.out.println("a is +ve")
      }
      else if (a < 0) {
         System.out.println("a is -ve")
      }
      else {
         System.out.println("a is zero"
      }
   }
}
```

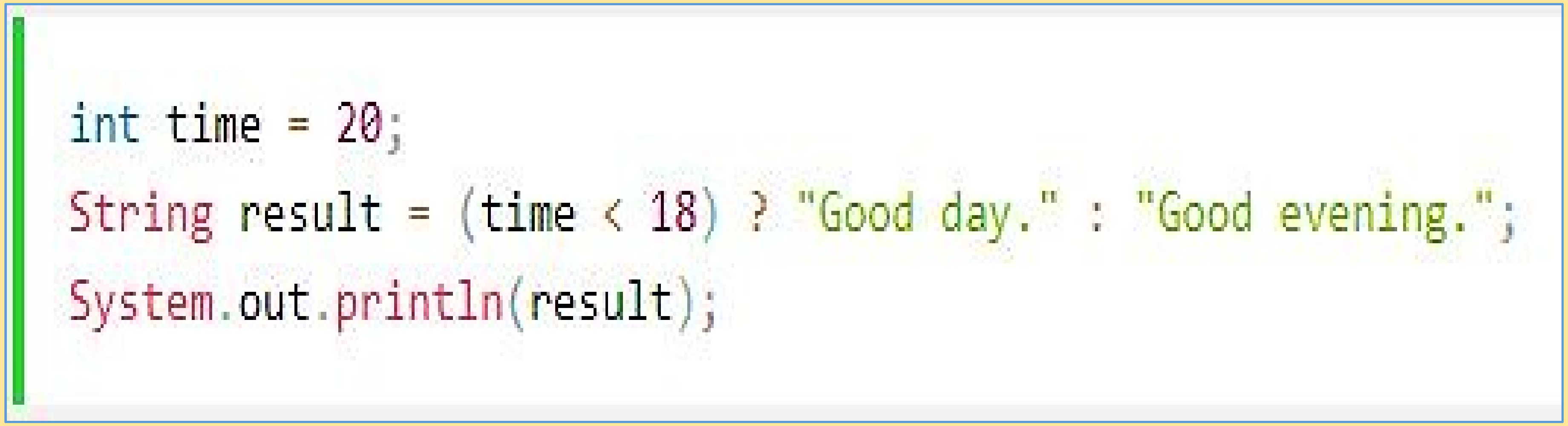
Output:

```
a is +ve
```

If...Else & Ternary Operator – A comparison



```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```



```
int time = 20;
String result = (time < 18) ? "Good day." : "Good evening.";
System.out.println(result);
```

switch case

The if statement in java, makes selections based on a single true
alse condition. But switch case have multiple choice for selecti
of the statements

It is like if-else-if ladder statement

How to Java switch works:

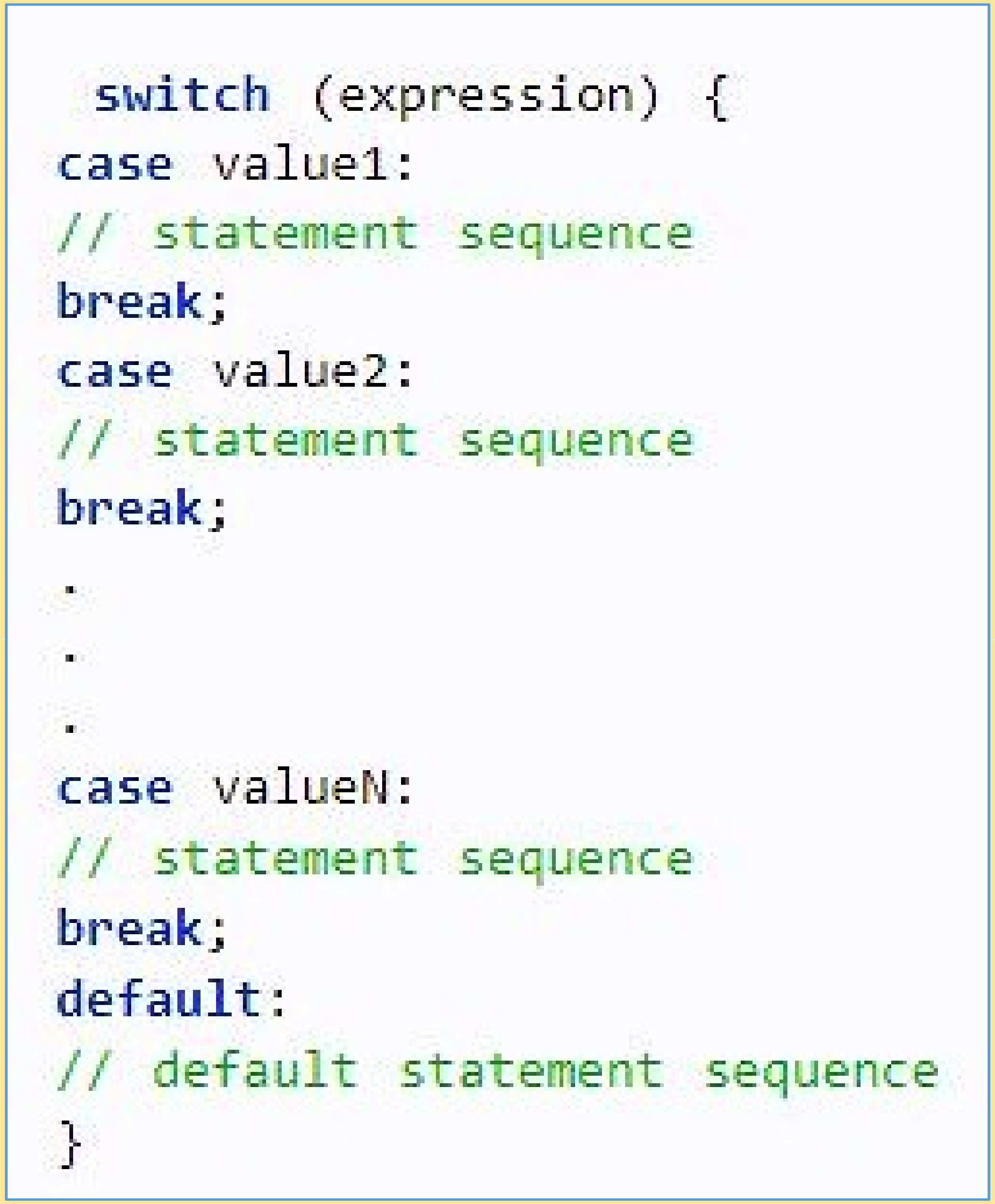
Matching each expression with case

Once it match, execute all case from where it matched.

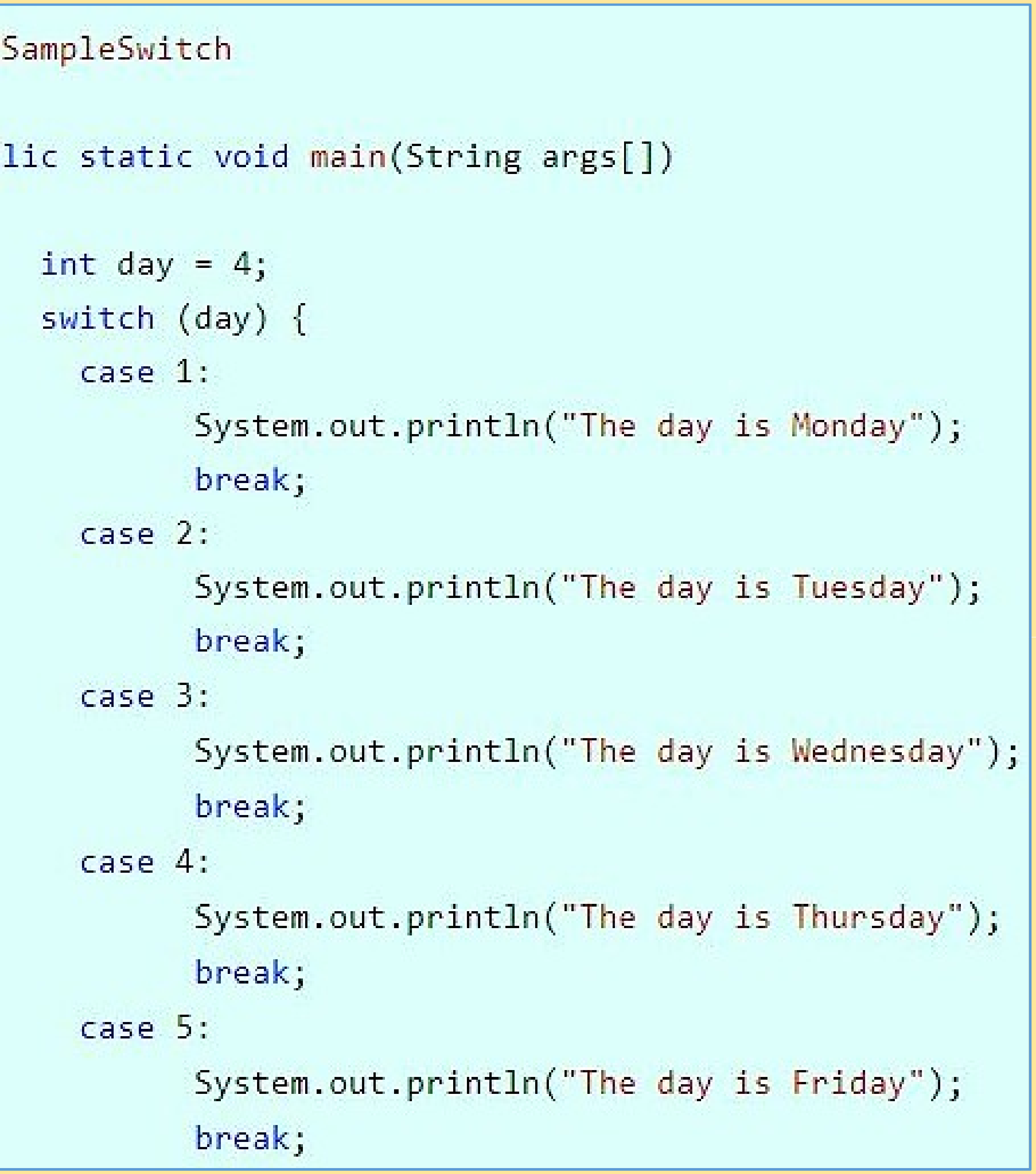
Use break to exit from switch

Use default when expression does not match with any case

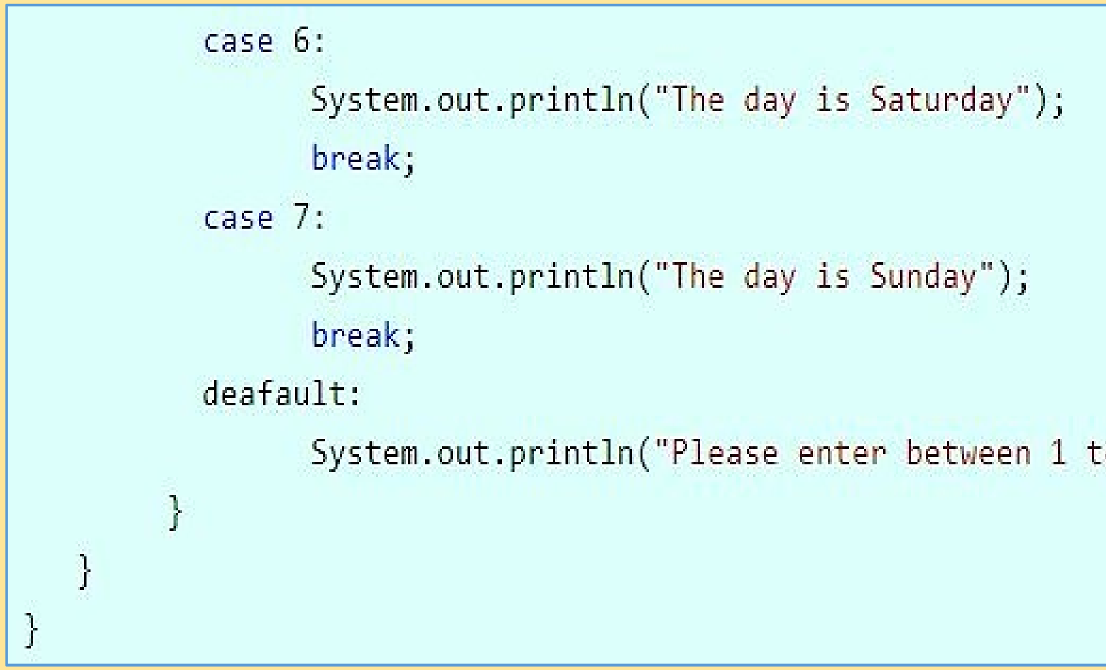
yntax



```
 switch (expression) {
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
.
.
.
case valueN:
// statement sequence
break;
default:
// default statement sequence
}
```



```
SampleSwitch

lic static void main(String args[])

  int day = 4;
  switch (day) {
    case 1:
          System.out.println("The day is Monday");
          break;
    case 2:
          System.out.println("The day is Tuesday");
          break;
    case 3:
          System.out.println("The day is Wednesday");
          break;
    case 4:
          System.out.println("The day is Thursday");
          break;
    case 5:
          System.out.println("The day is Friday");
          break;
```



```
          case 6:
               System.out.println("The day is Saturday");
               break;
          case 7:
               System.out.println("The day is Sunday");
               break;
          deafault:
               System.out.println("Please enter between 1 t
        }
    }
}
```

Output

The day is Thursday

Why break is necessary in switch statement ?

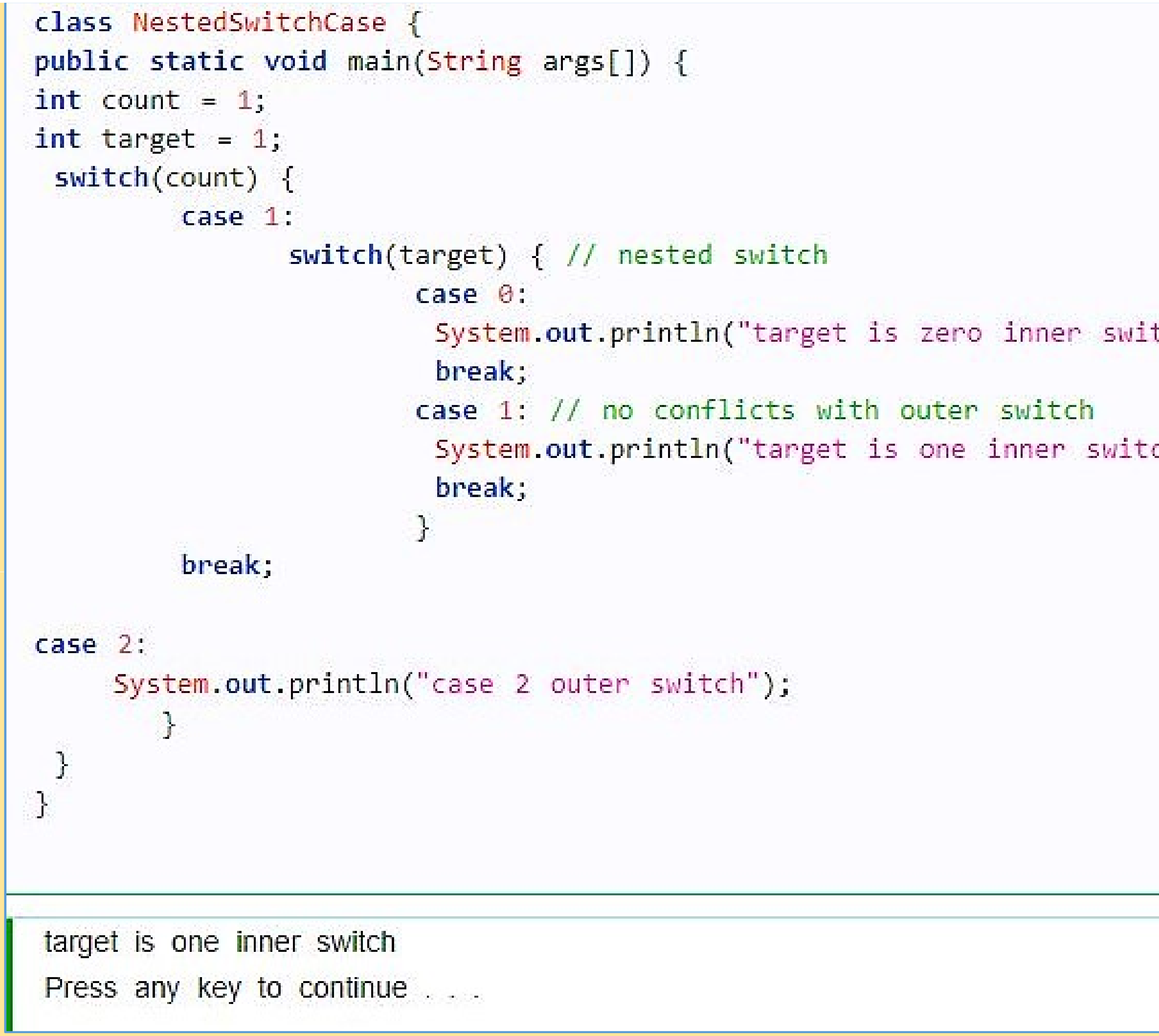
The break statement is used inside the switch to terminate statement sequence.

When a break statement is encountered, execution branches t the first line of code that follows the entire switch statement

This has the effect of jumping out of the switch.

The break statement is optional. If you omit the break, executio will continue on into the next case.

Nested Switch



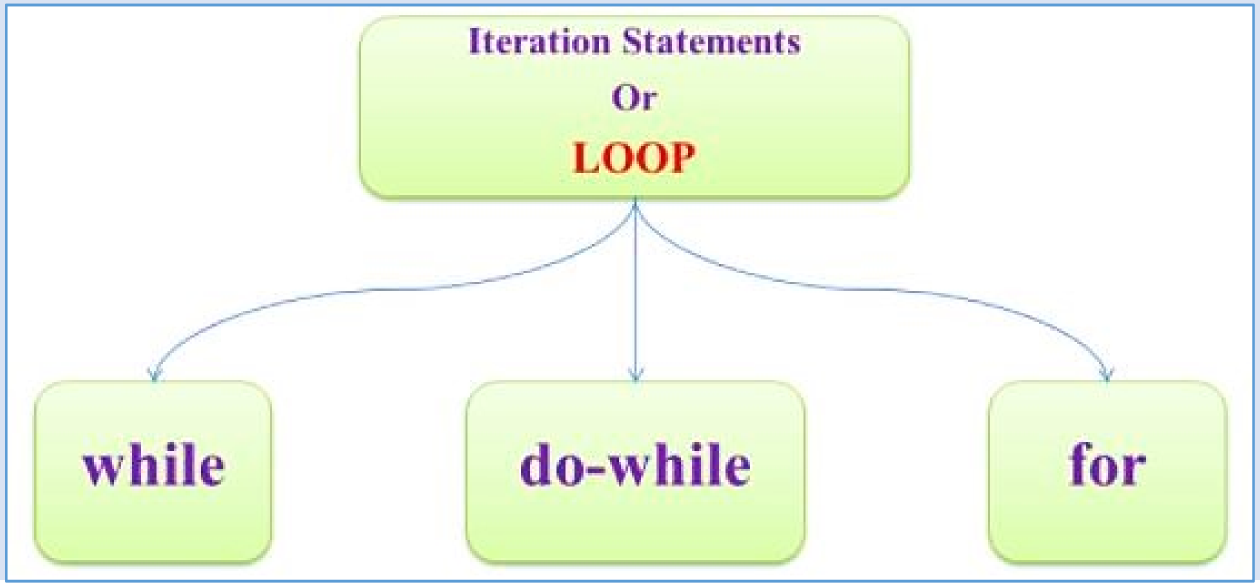
```
class NestedSwitchCase {
public static void main(String args[]) {
int count = 1;
int target = 1;
 switch(count) {
        case 1:
               switch(target) { // nested switch
                       case 0:
                        System.out.println("target is zero inner swit
                        break;
                       case 1: // no conflicts with outer switch
                        System.out.println("target is one inner switc
                        break;
                       }
        break;

case 2:
     System.out.println("case 2 outer switch");
        }
 }
}
```

```
target is one inner switch
Press any key to continue . . .
```

Iteration Statements (Loop)

A loop can be used to tell a program to execute statement repeatedly

A loop repeatedly executes the same set of instructions until termination condition is met.



❖ While Loop

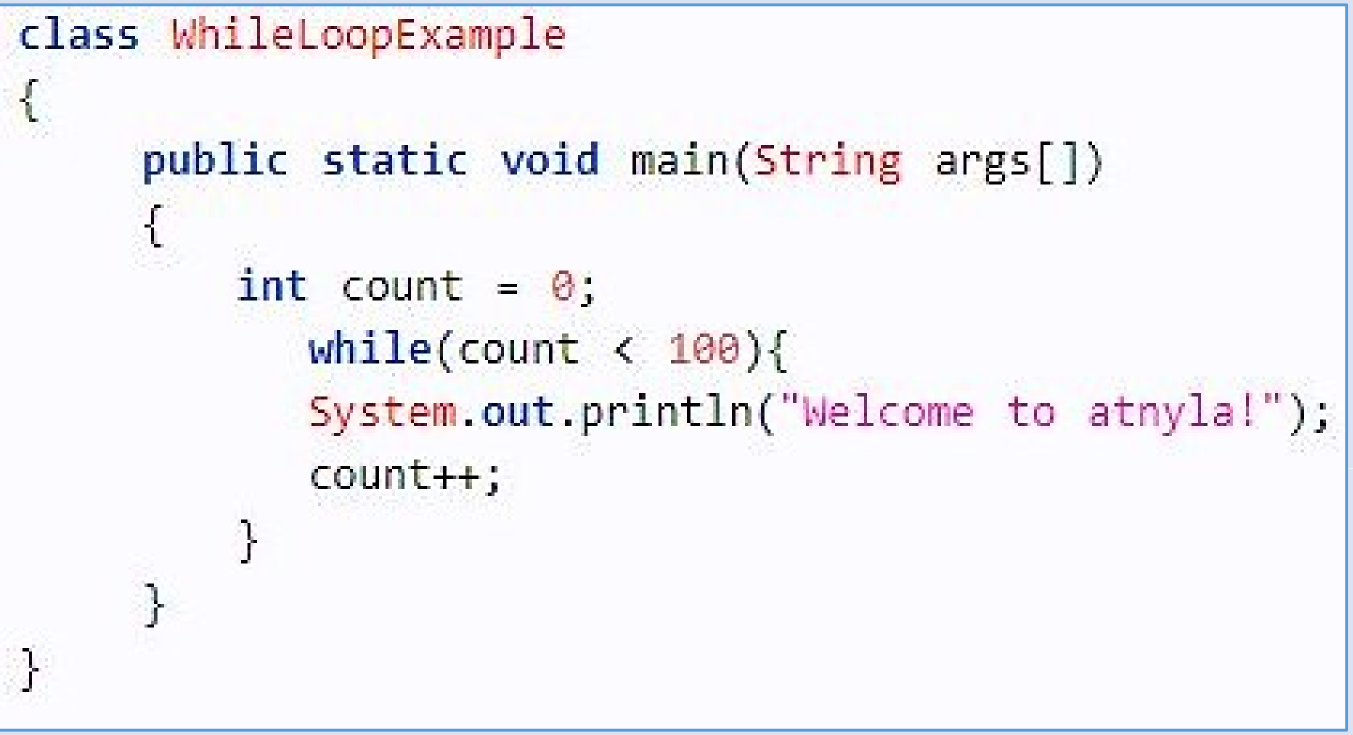
➢In **while loop** first checks the condition if the condition is tru
then control goes inside the loop body otherwise goes outside
the body.

yntax

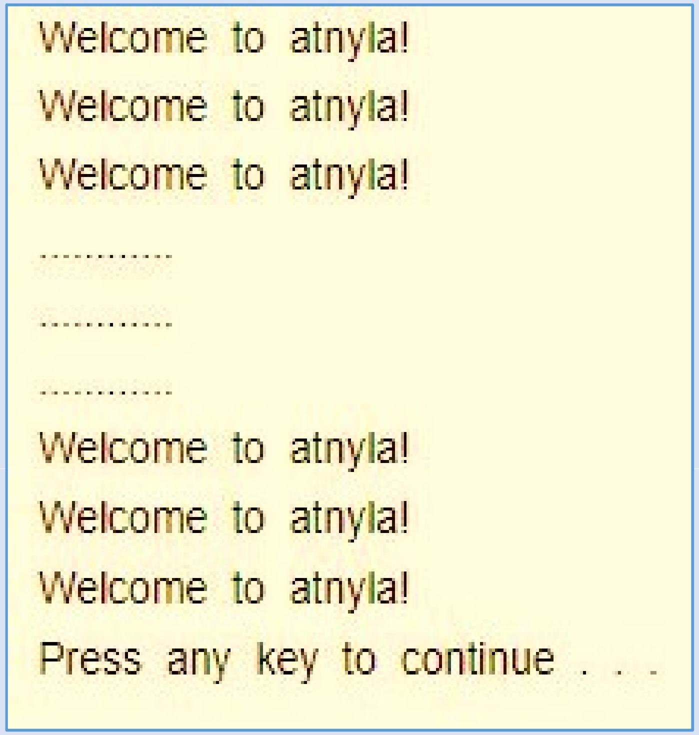
```
    while (condition)
     {
       // code block to be executed
     }
```

xample - 1

Output

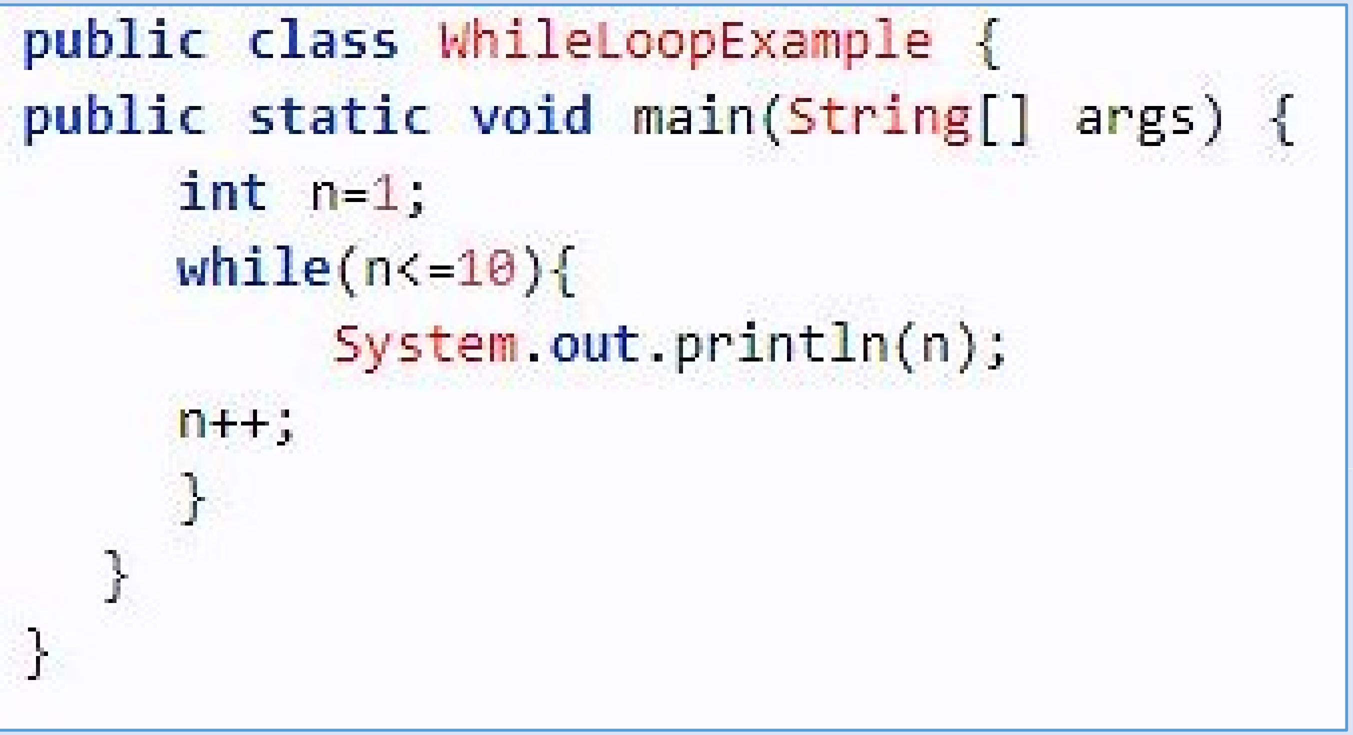


```
class WhileLoopExample
{
    public static void main(String args[])
    {
        int count = 0;
          while(count < 100){
          System.out.println("Welcome to atnyla!");
          count++;
        }
    }
}
```



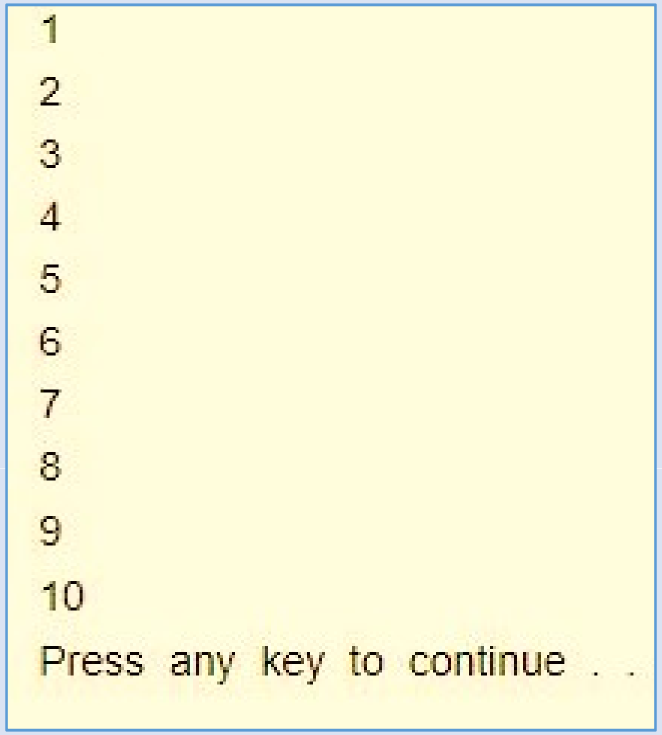
```
Welcome to atnyla!
Welcome to atnyla!
Welcome to atnyla!
...........
...........
...........
Welcome to atnyla!
Welcome to atnyla!
Welcome to atnyla!
Press any key to continue . . .
```

xample - 2



```
public class WhileLoopExample {
public static void main(String[] args) {
     int n=1;
     while(n<=10){
          System.out.println(n);
     n++;
     }
   }
}
```

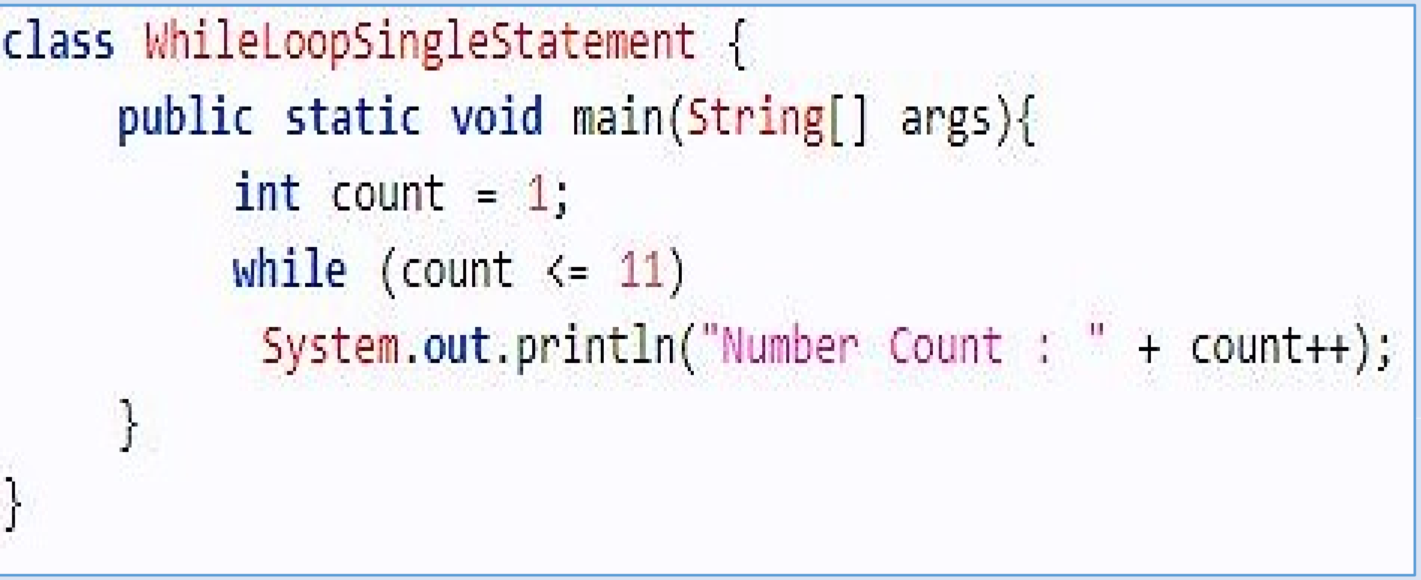
Output



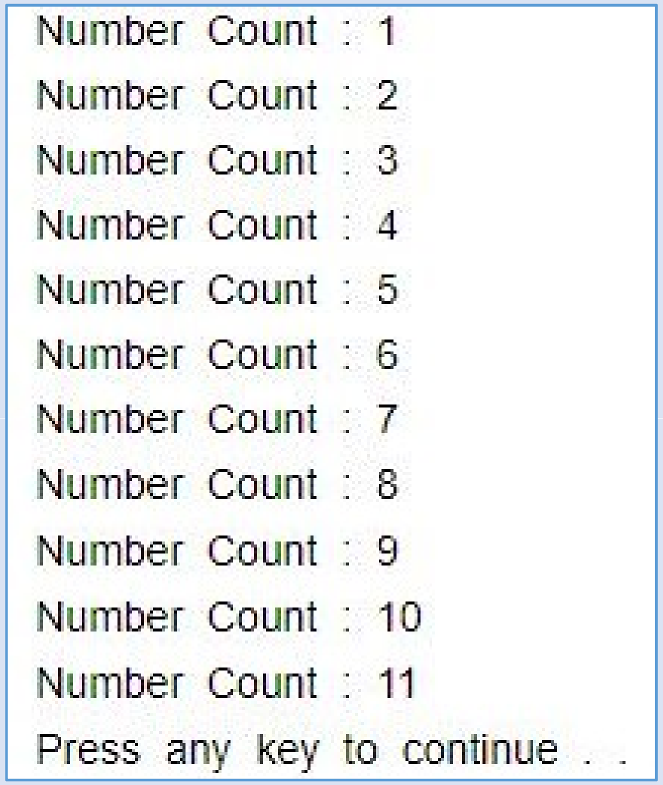
```
1
2
3
4
5
6
7
8
9
10
Press any key to continue . .
```

xample - 3

Output



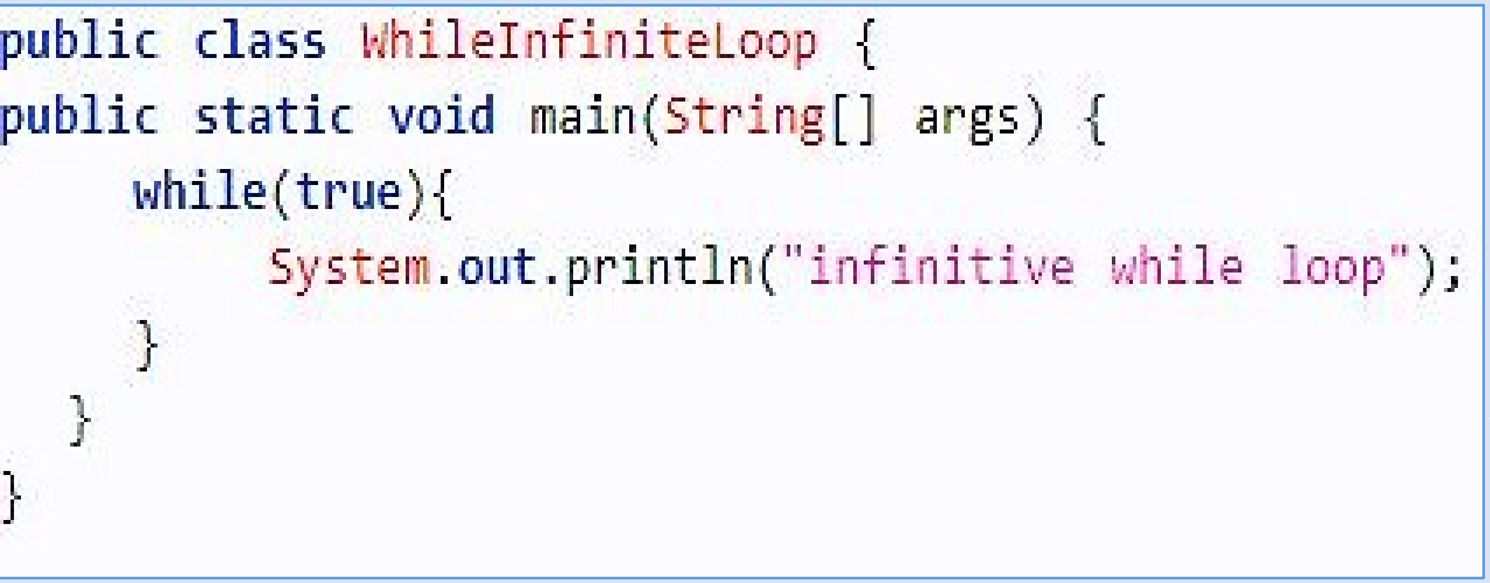
```
class WhileLoopSingleStatement {
    public static void main(String[] args){
        int count = 1;
        while (count <= 11)
         System.out.println("Number Count : " + count++);
    }
}
```



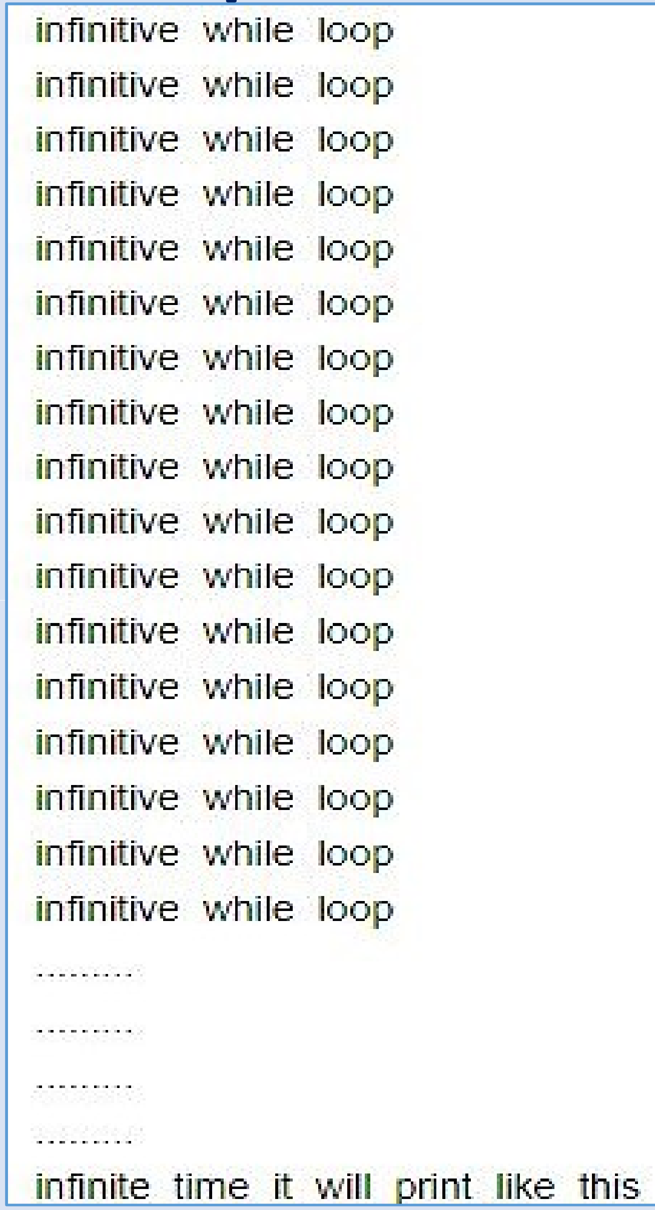
```
Number Count : 1
Number Count : 2
Number Count : 3
Number Count : 4
Number Count : 5
Number Count : 6
Number Count : 7
Number Count : 8
Number Count : 9
Number Count : 10
Number Count : 11
Press any key to continue . . .
```

xample - 4

Output

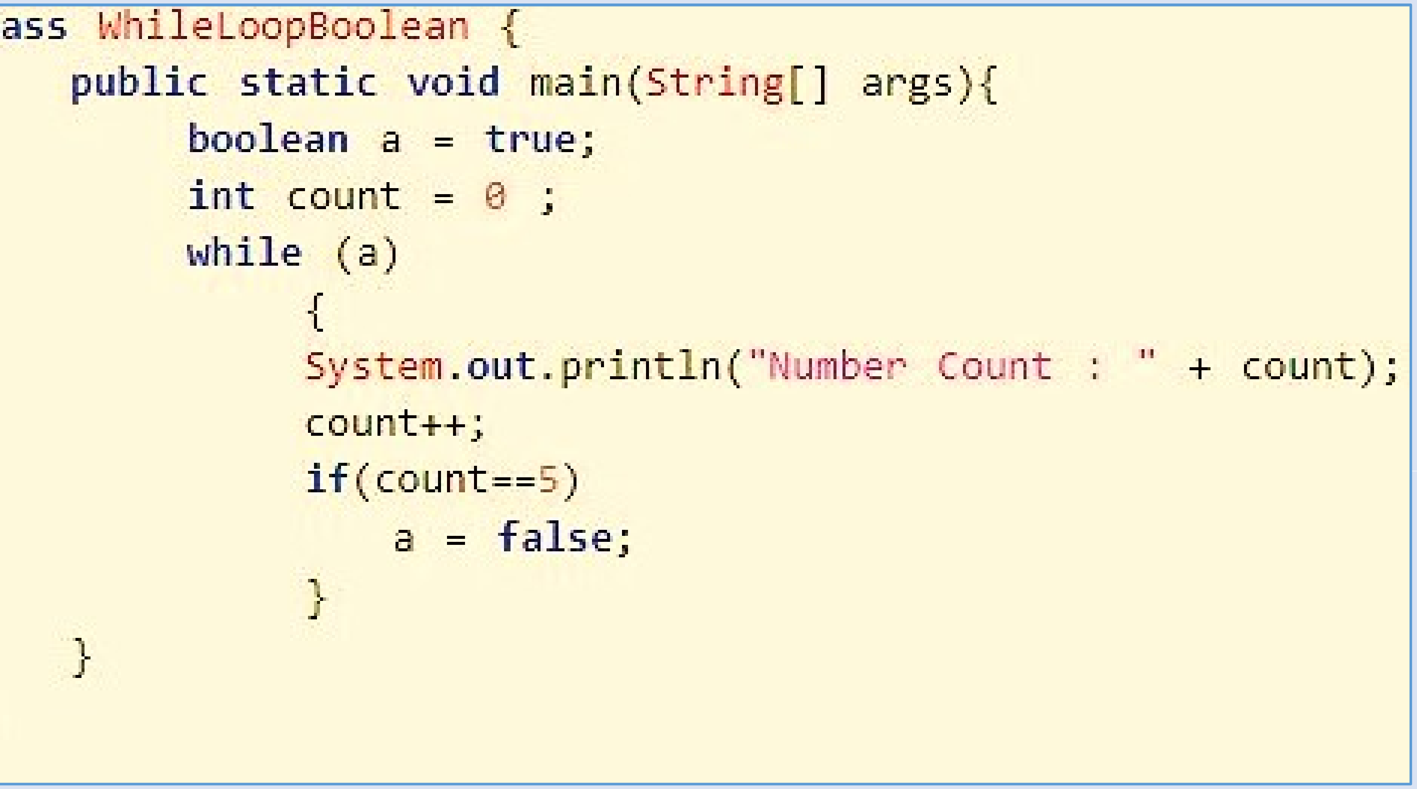


```
public class WhileInfiniteLoop {
public static void main(String[] args) {
    while(true){
        System.out.println("infinitive while loop");
    }
  }
}
```



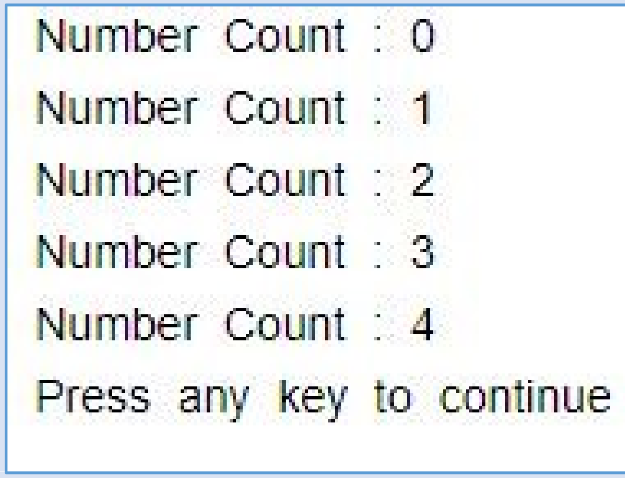
```
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
.........
.........
.........
.........
infinite time it will print like this
```

ample - 5 (Boolean Condition inside while loop)



```
ass WhileLoopBoolean {
    public static void main(String[] args){
          boolean a = true;
          int count = 0 ;
          while (a)
                {
                System.out.println("Number Count : " + count);
                count++;
                if(count==5)
                    a = false;
                }
    }
```

Output



```
Number Count : 0
Number Count : 1
Number Count : 2
Number Count : 3
Number Count : 4
Press any key to continue
```

do…while loop

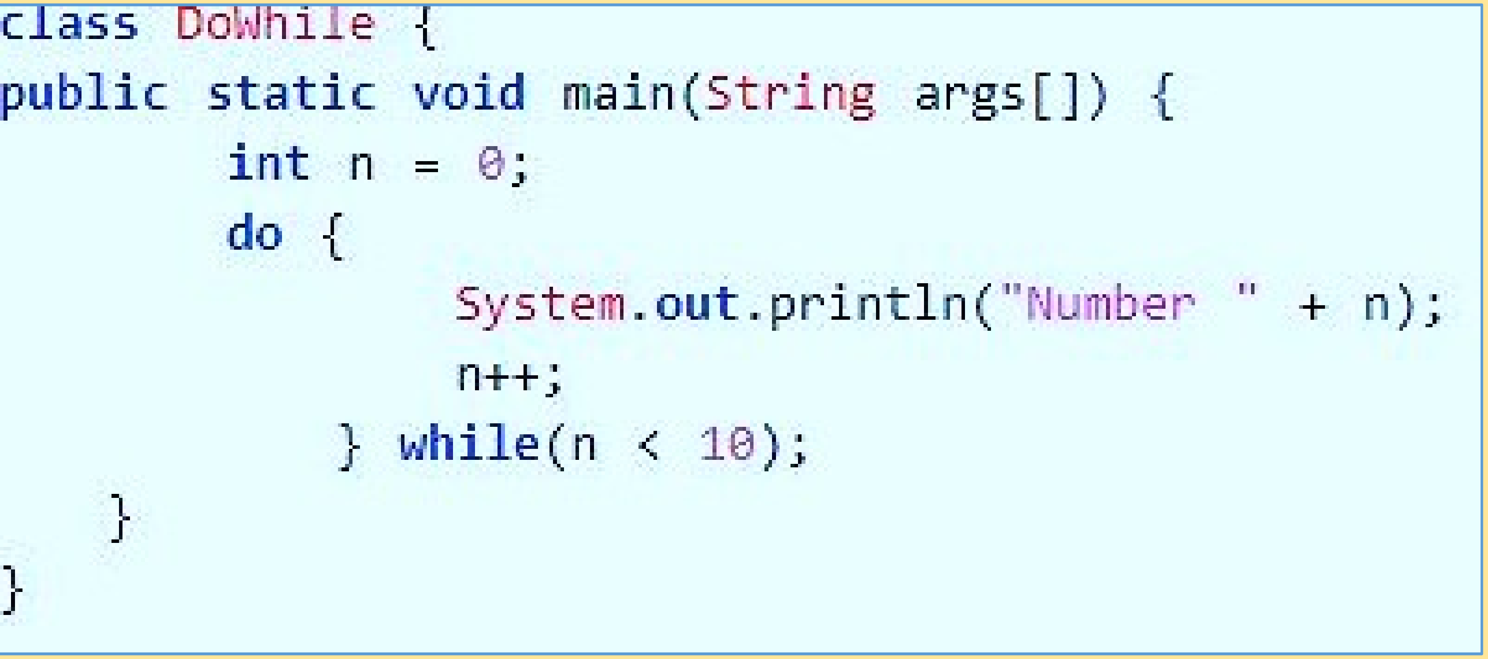
A do while loop is a control flow statement that executes a blo
of code at least once, and then repeatedly executes the block,
not, depending on a given condition at the end of the block (
while).

yntax

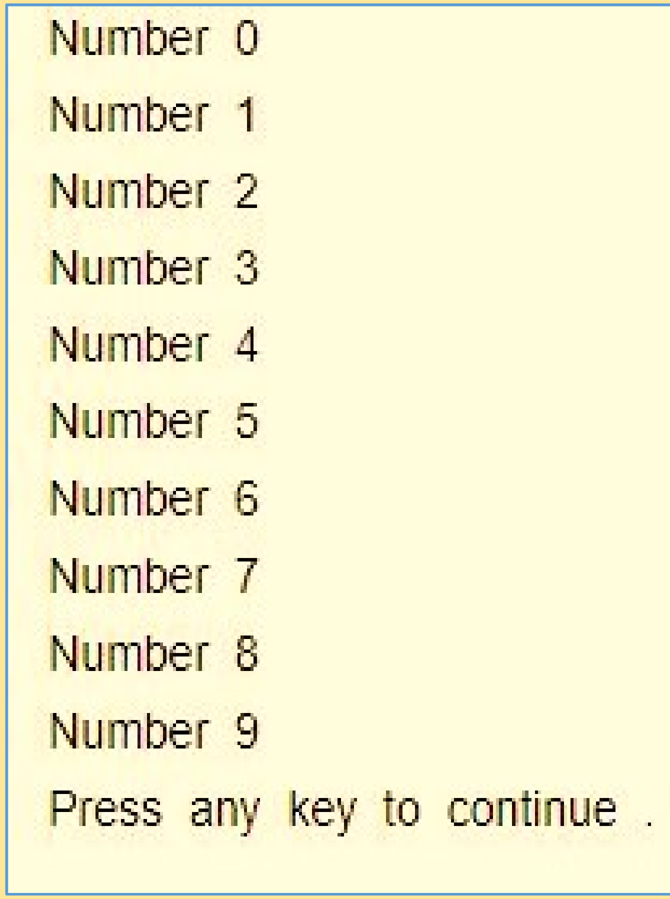
```
o {
      // code block to be executed
   } while (condition);
```

xample -1

Output

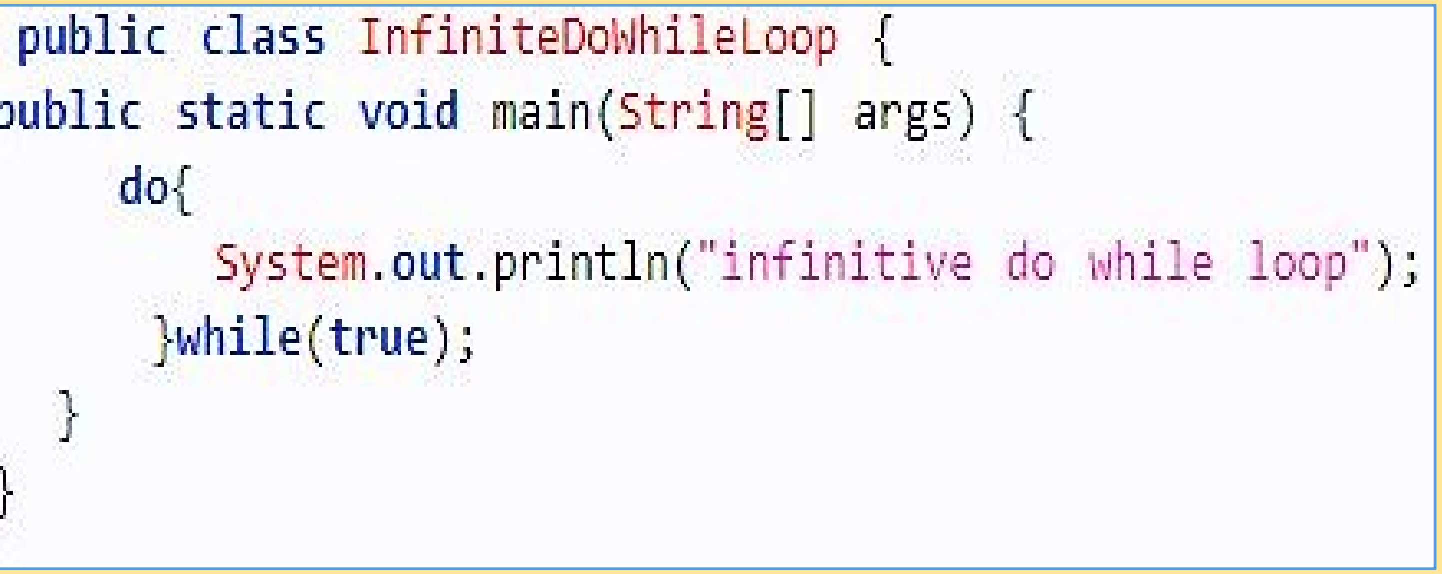


```
class DoWhile {
public static void main(String args[]) {
        int n = 0;
        do {
                System.out.println("Number " + n);
                n++;
            } while(n < 10);
    }
}
```



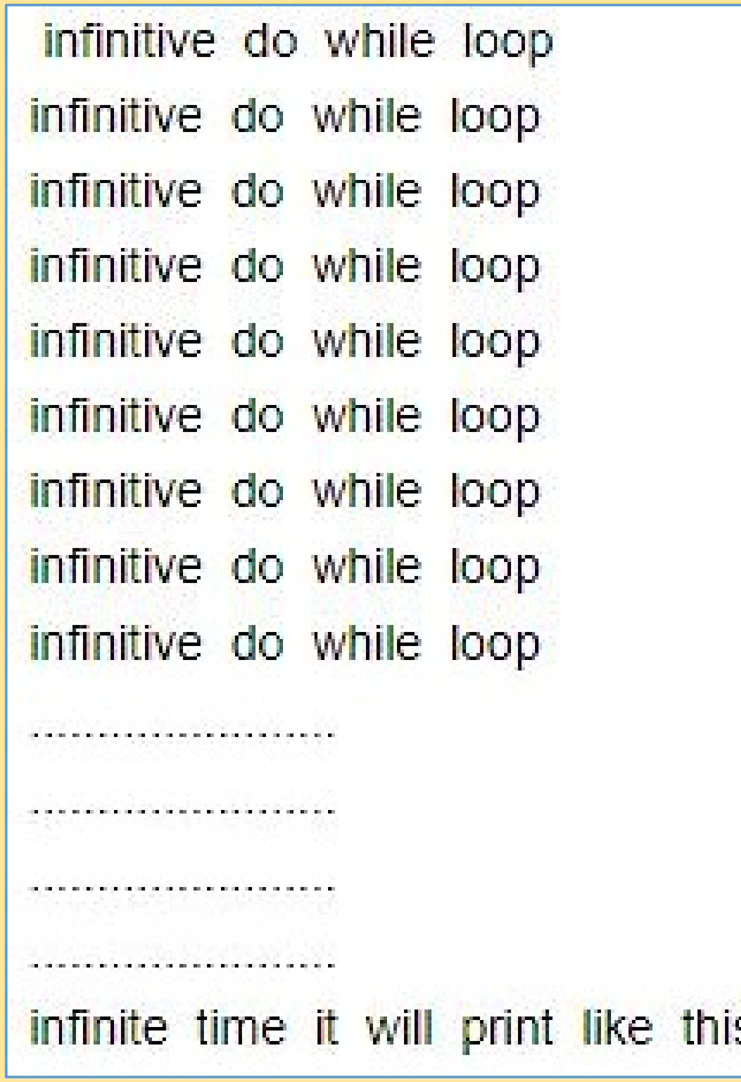
```
Number 0
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Press any key to continue .
```

xample -2 (Infinitive do-while Loop)



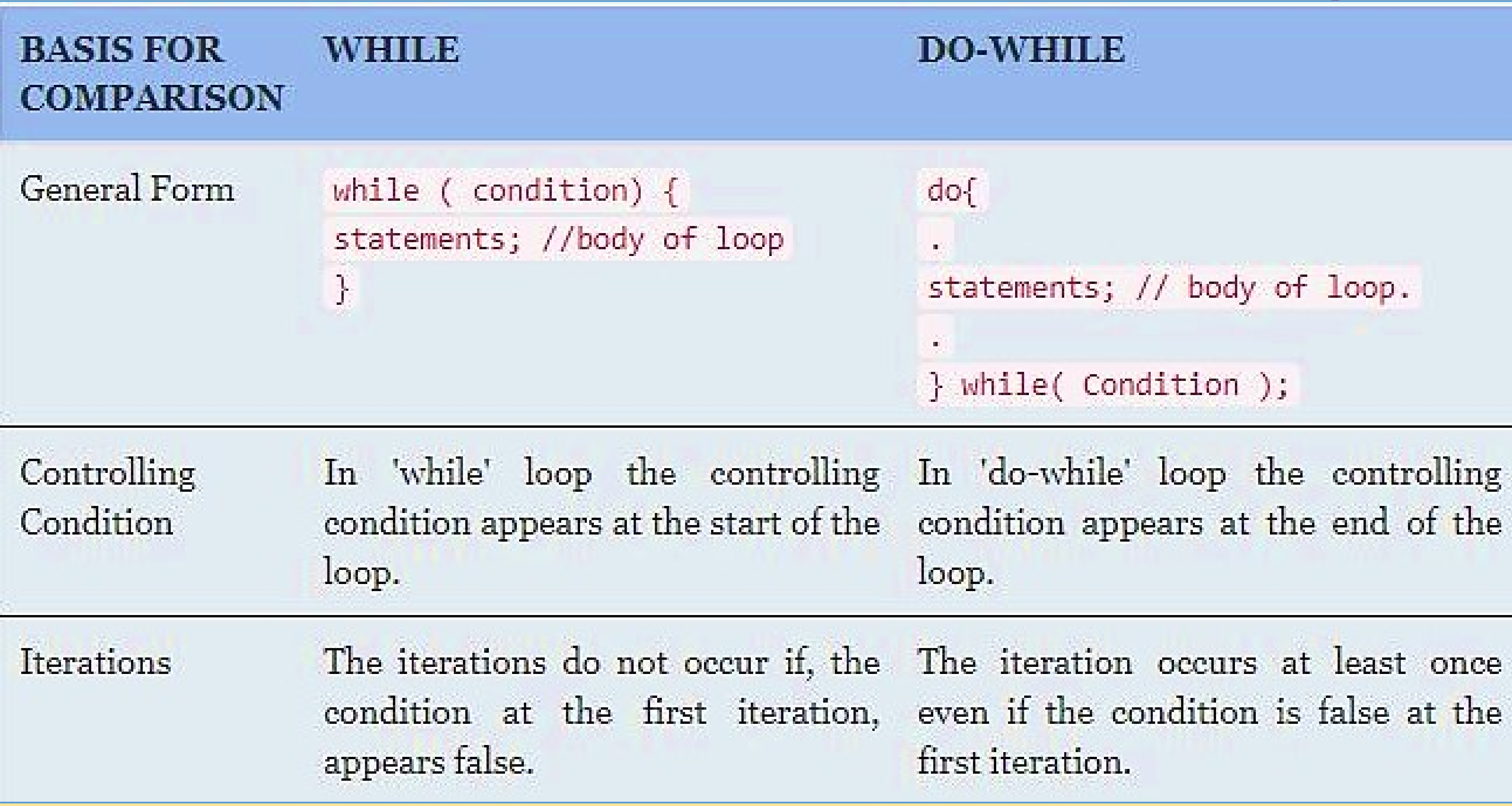
```
 public class InfiniteDoWhileLoop {
ublic static void main(String[] args) {
        do{
            System.out.println("infinitive do while loop");
         }while(true);
    }
}
```

Output



```
 infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
........................
........................
........................
........................
infinite time it will print like thi
```

Difference Between while and do-while Loop



BASIS FOR COMPARISON	WHILE	DO-WHILE

General Form	`while ( condition) {`
	`statements; //body of loop`
	`}`
		`do{`
		`.`
		`statements; // body of loop.`
		`.`
		`} while( Condition );`

Controlling Condition	In 'while' loop the controlling condition appears at the start of the loop.	In 'do-while' loop the controlling condition appears at the end of the loop.
Iterations	The iterations do not occur if, the condition at the first iteration, appears false.	The iteration occurs at least once even if the condition is false at the first iteration.

for loop

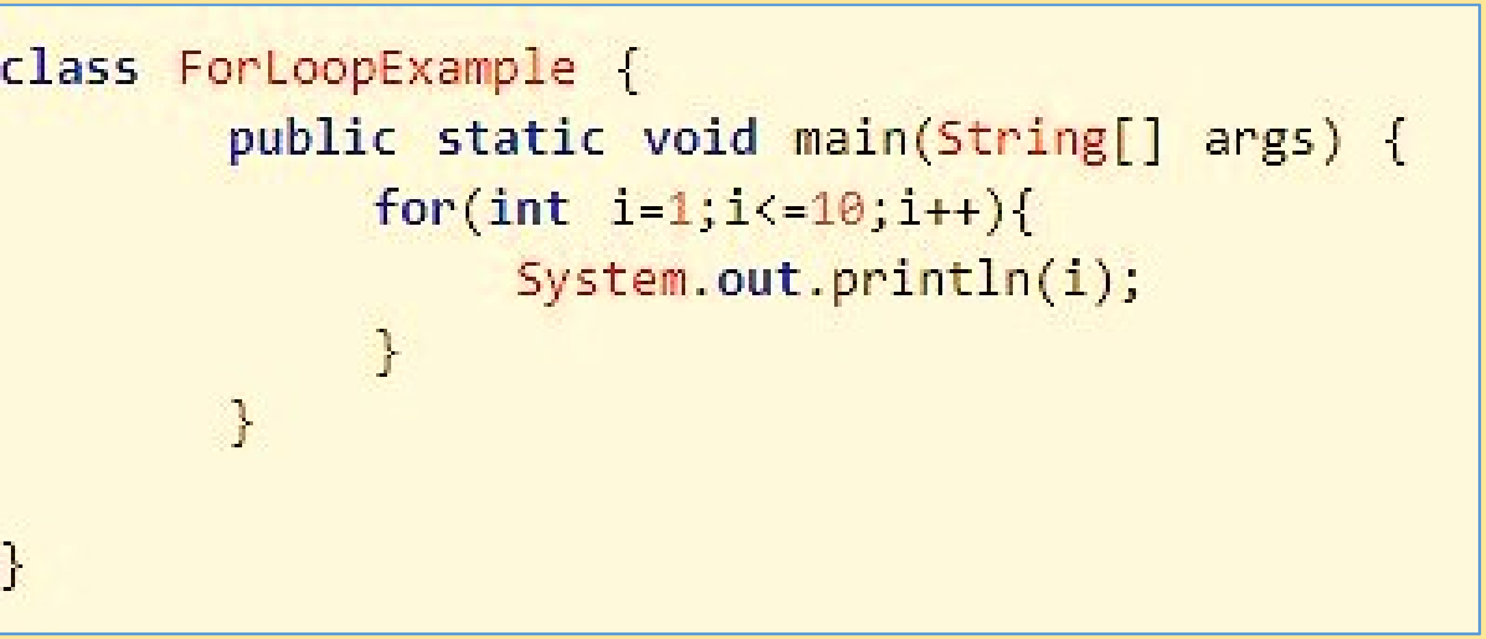
For Loop is used to execute set of statements repeatedly until the condition is true.

ntax

```
for (initialization; condition; increment/decrement)
    {
        // code block to be executed
    }
```

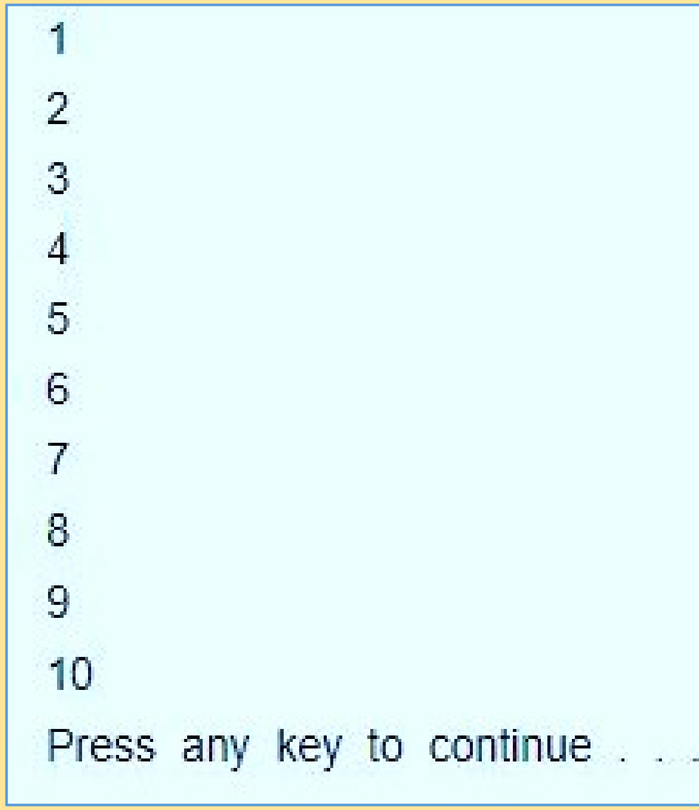
itialization : It executes at once.
ondition : This check until get true.
crement/Decrement: This is for increment or decrement.

xample 1



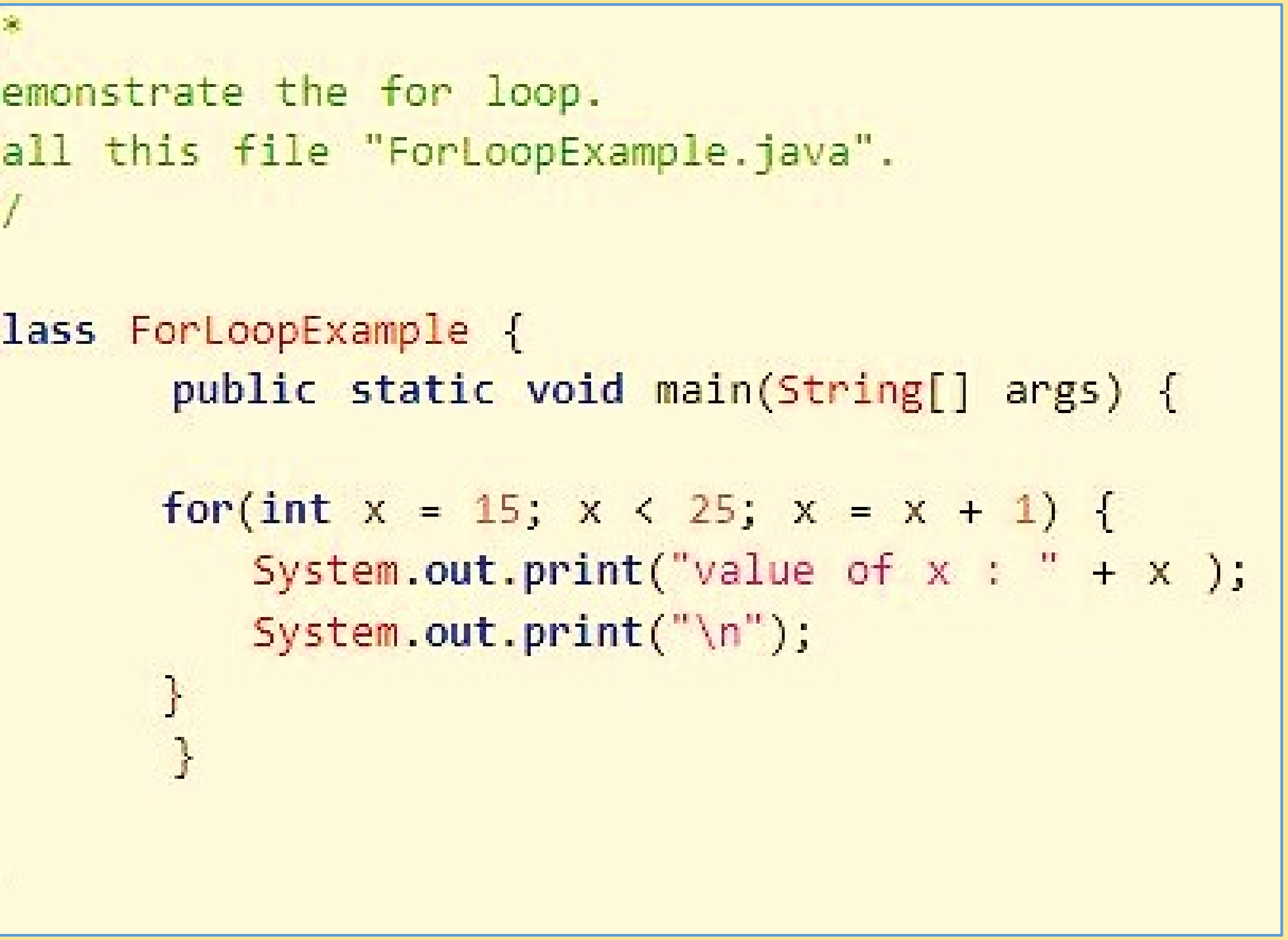
```
class ForLoopExample {
        public static void main(String[] args) {
                for(int i=1;i<=10;i++){
                        System.out.println(i);
                }
        }

}
```

Output



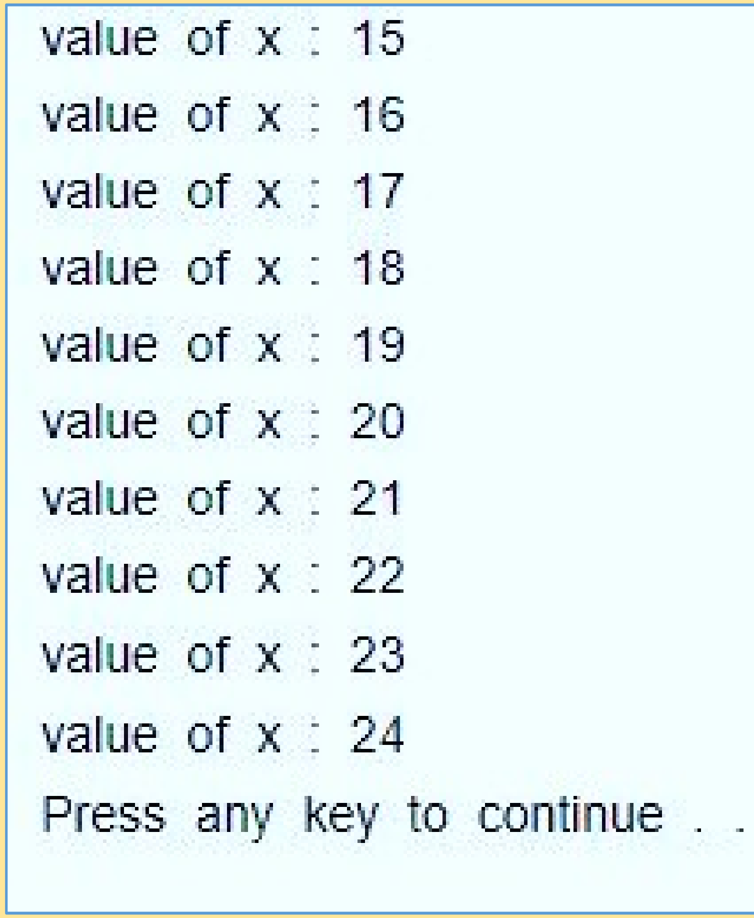
```
1
2
3
4
5
6
7
8
9
10
Press any key to continue . . .
```

xample 2



```
*
emonstrate the for loop.
all this file "ForLoopExample.java".
/

lass ForLoopExample {
        public static void main(String[] args) {

        for(int x = 15; x < 25; x = x + 1) {
            System.out.print("value of x : " + x );
            System.out.print("\n");
        }
         }
```

Output



```
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
value of x : 20
value of x : 21
value of x : 22
value of x : 23
value of x : 24
Press any key to continue . .
```

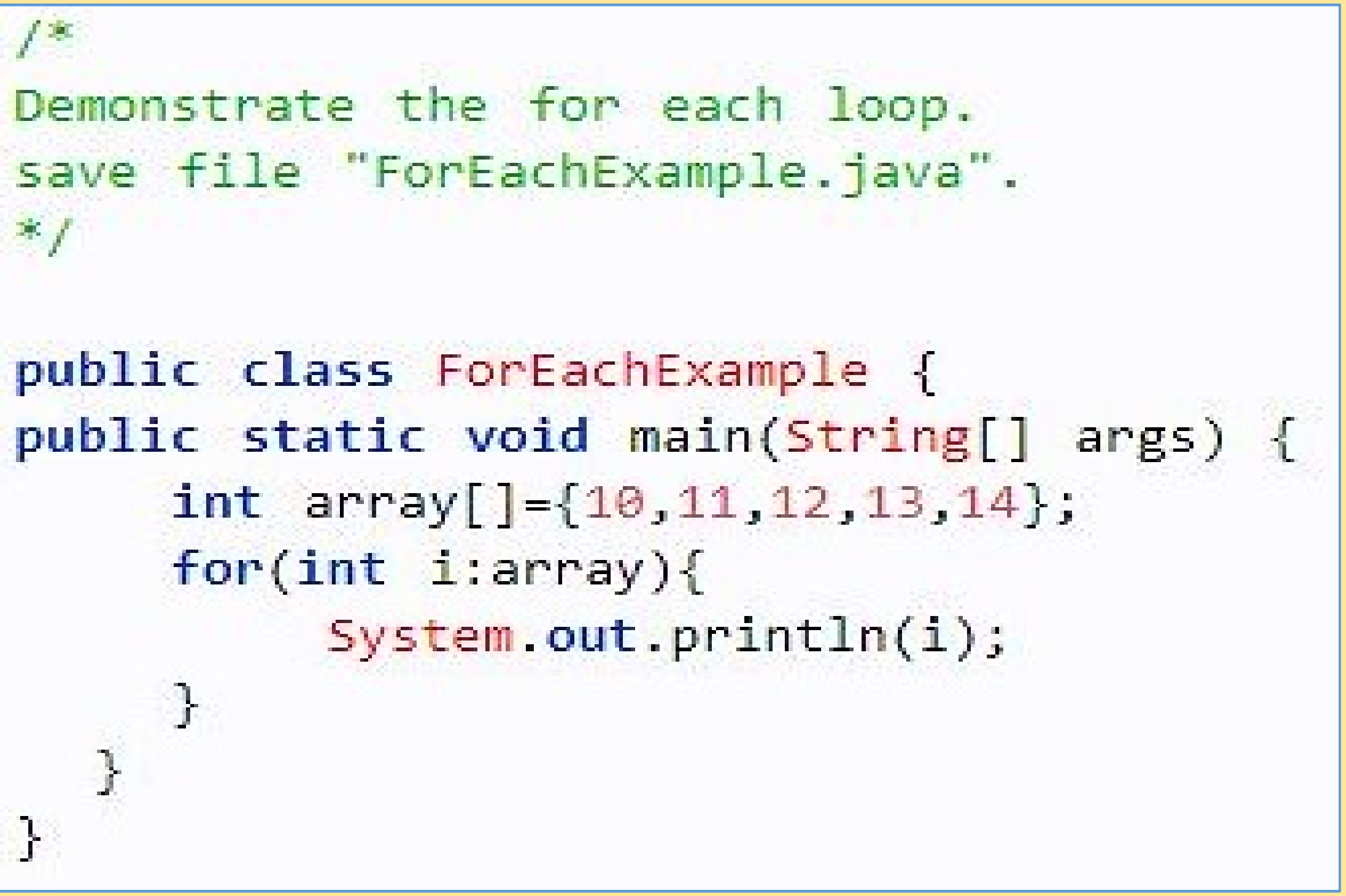
For-each or Enhanced For Loop

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

ntax

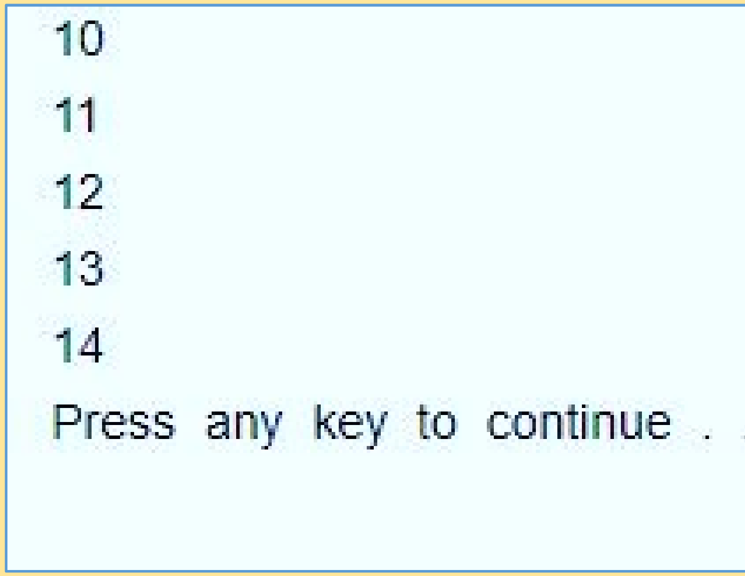
```
for (type variableName : arrayName)
 {
    // code block to be executed
 }
```

Example



```
/*
Demonstrate the for each loop.
save file "ForEachExample.java".
*/

public class ForEachExample {
public static void main(String[] args) {
      int array[]={10,11,12,13,14};
      for(int i:array){
            System.out.println(i);
      }
   }
}
```

Output



```
10
11
12
13
14
Press any key to continue .
```

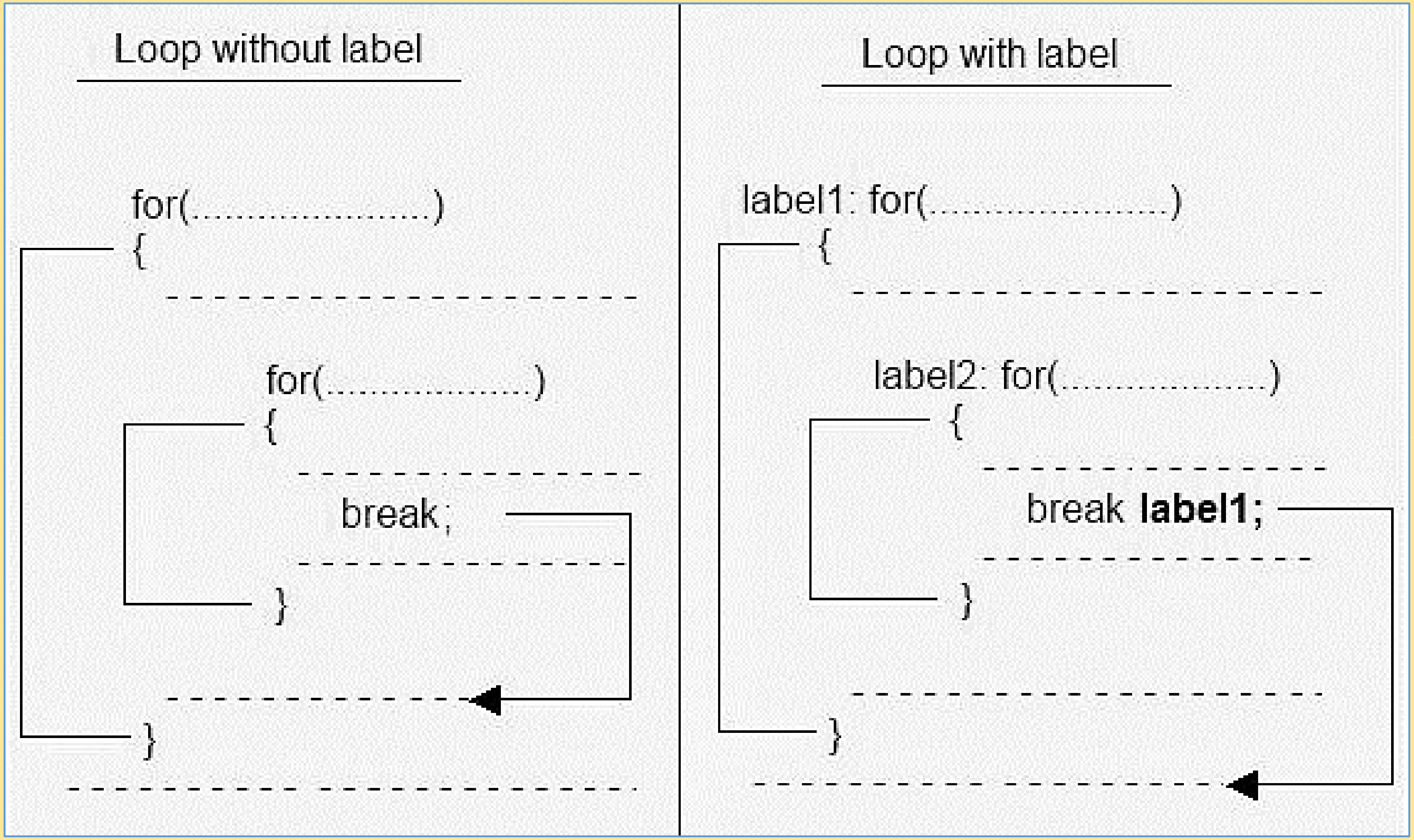
Labeled For Loop

According to nested loop, if we put break statement in inner loo
compiler will jump out from inner loop and continue the outer loo
gain.

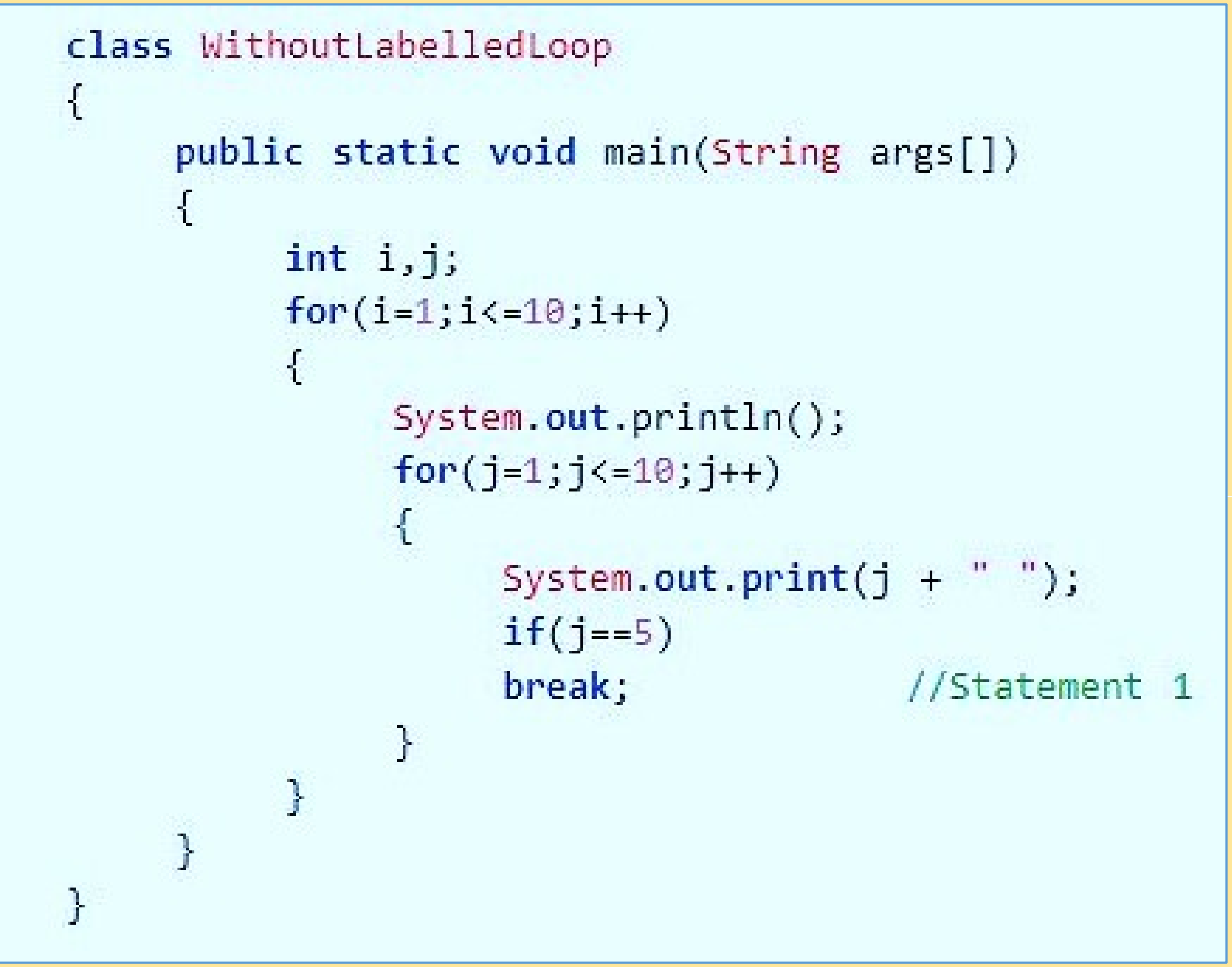
What if we need to jump out from the outer loop using brea
tatement given inside inner loop? The answer is, we shoul
define **label** along with colon(:) sign before loop.

ntax

```
belname:
r(initialization; condition; increment/decrement)

 //code to be executed
```

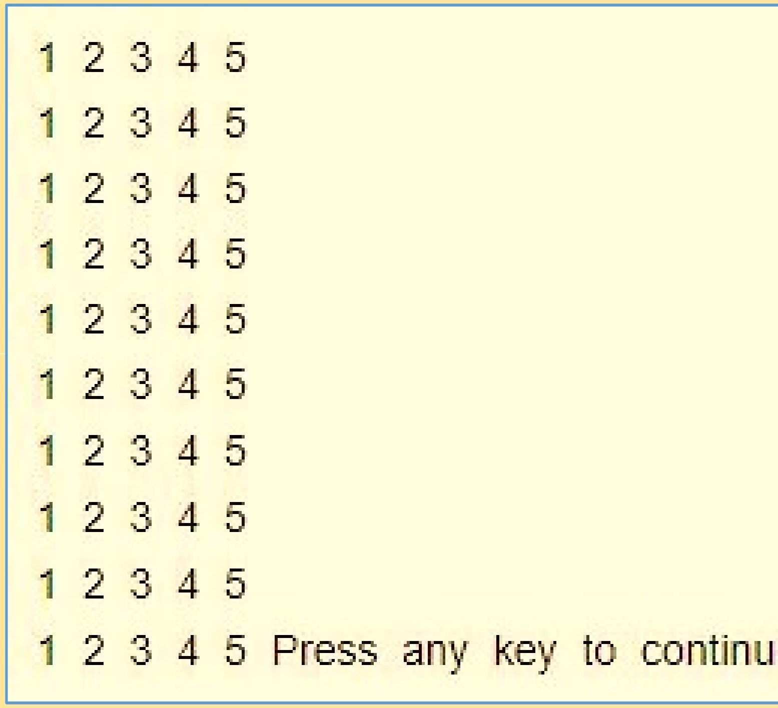

Loop without label
for(......................)
{
for(....................)
{
break;
}
}
Loop with label
label1: for(......................)
{
label2: for(...................)
{
break **label1;**
}
}

xample without labelled loop



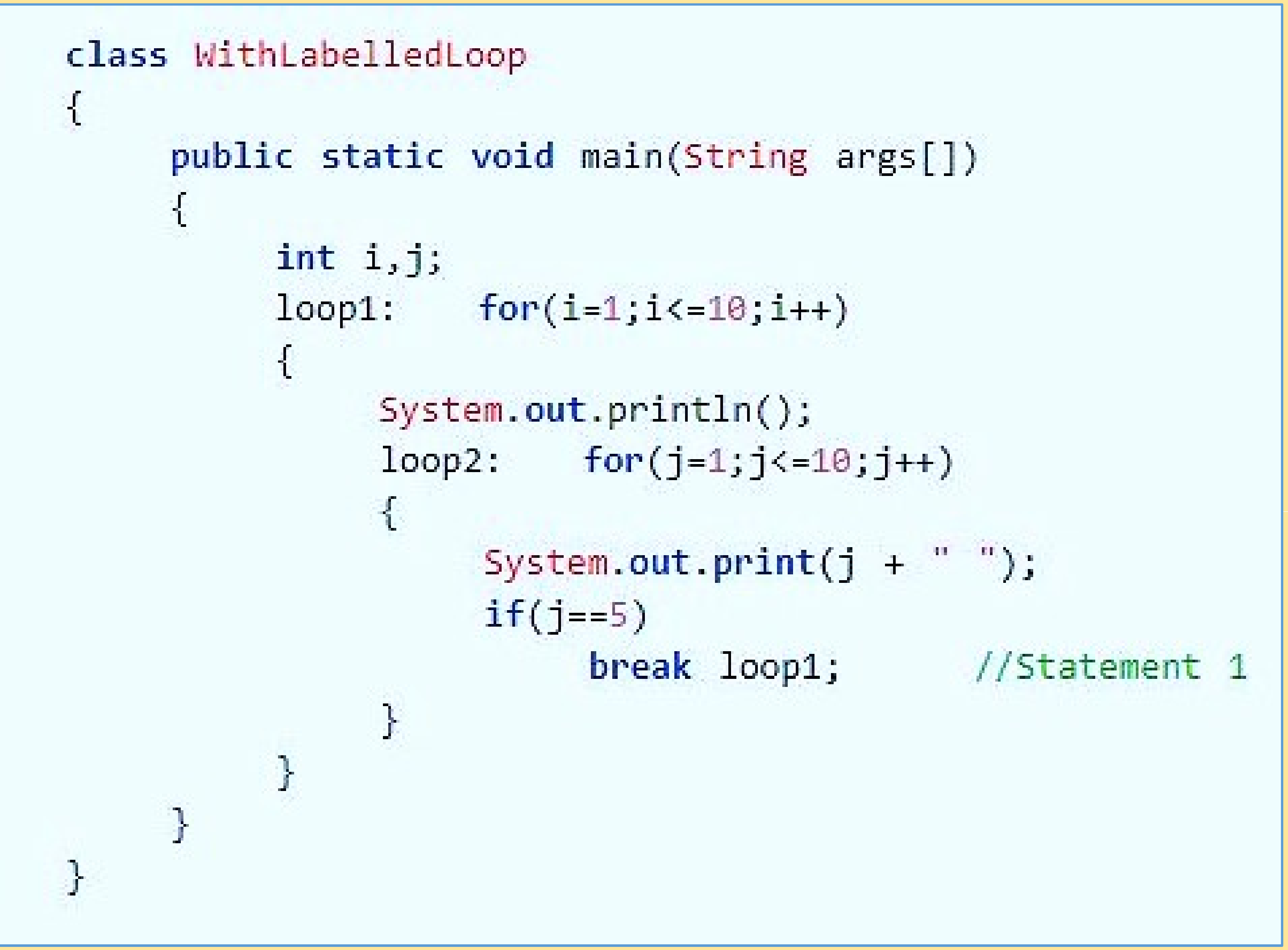
```
class WithoutLabelledLoop
{
    public static void main(String args[])
    {
        int i,j;
        for(i=1;i<=10;i++)
        {
            System.out.println();
            for(j=1;j<=10;j++)
            {
                System.out.print(j + " ");
                if(j==5)
                break;              //Statement 1
            }
        }
    }
}
```

Output



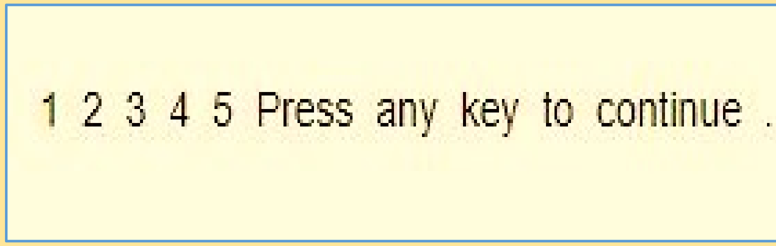
```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5 Press any key to continu
```

xample with labelled loop



```
class WithLabelledLoop
{
    public static void main(String args[])
    {
        int i,j;
        loop1:   for(i=1;i<=10;i++)
        {
            System.out.println();
            loop2:   for(j=1;j<=10;j++)
            {
                System.out.print(j + " ");
                if(j==5)
                    break loop1;      //Statement 1
            }
        }
    }
}
```

Output



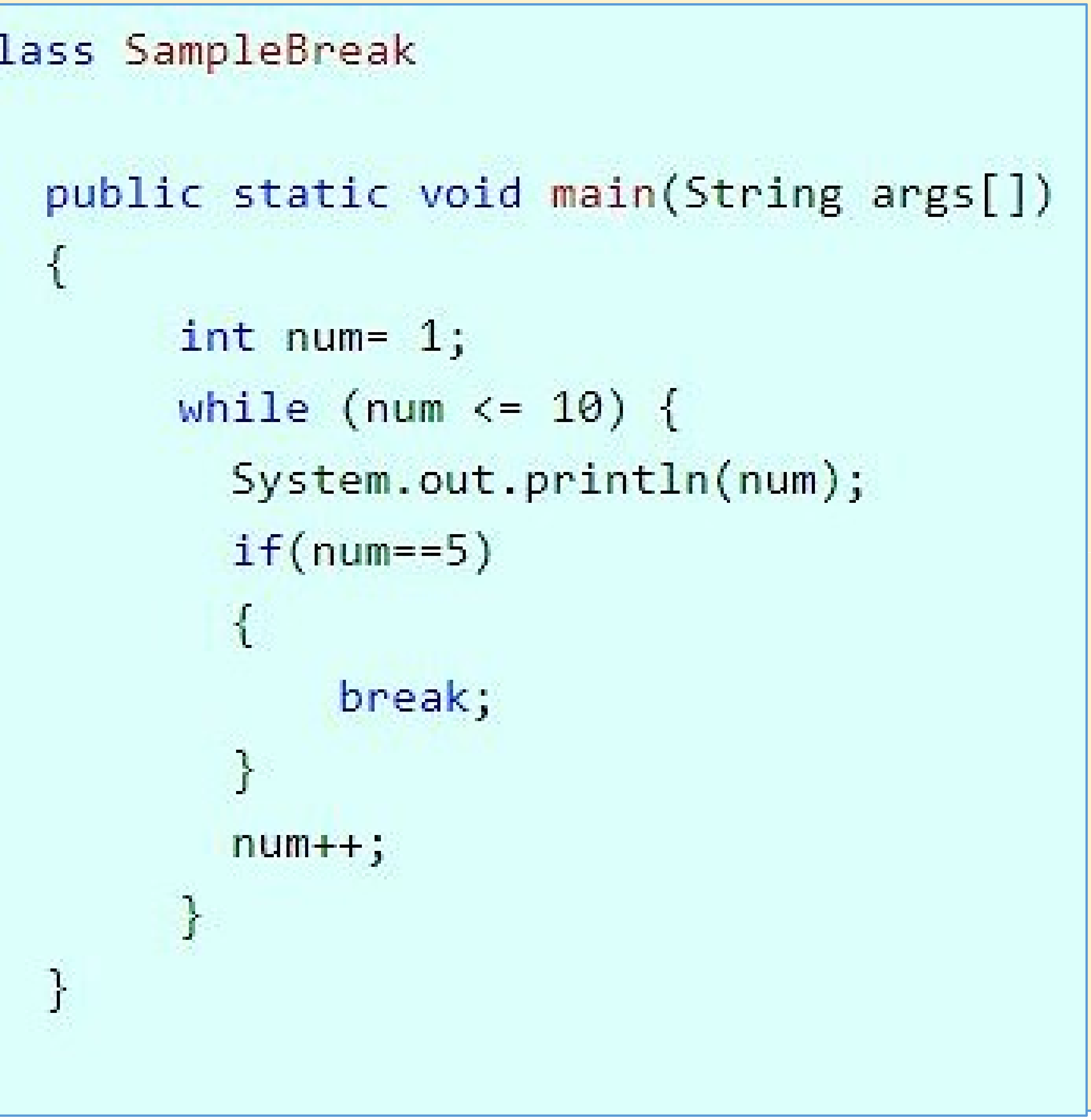
```
1 2 3 4 5 Press any key to continue .
```

Jump Statements

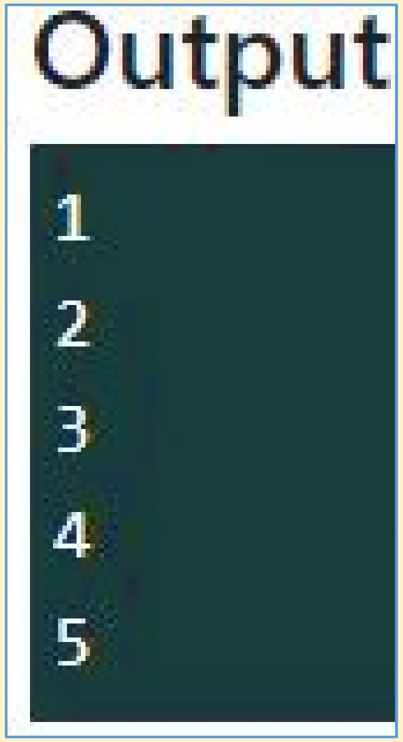
❖ Java Break Statement

- The Java break statement is used to break loop or switc statement
- It breaks the current flow of the program at specified condition
- When a break statement is encountered inside a loop, the loop i immediately terminated and the program control resumes at th next statement following the loop.
- In case of inner loop, it breaks only inner loop.

ample 1



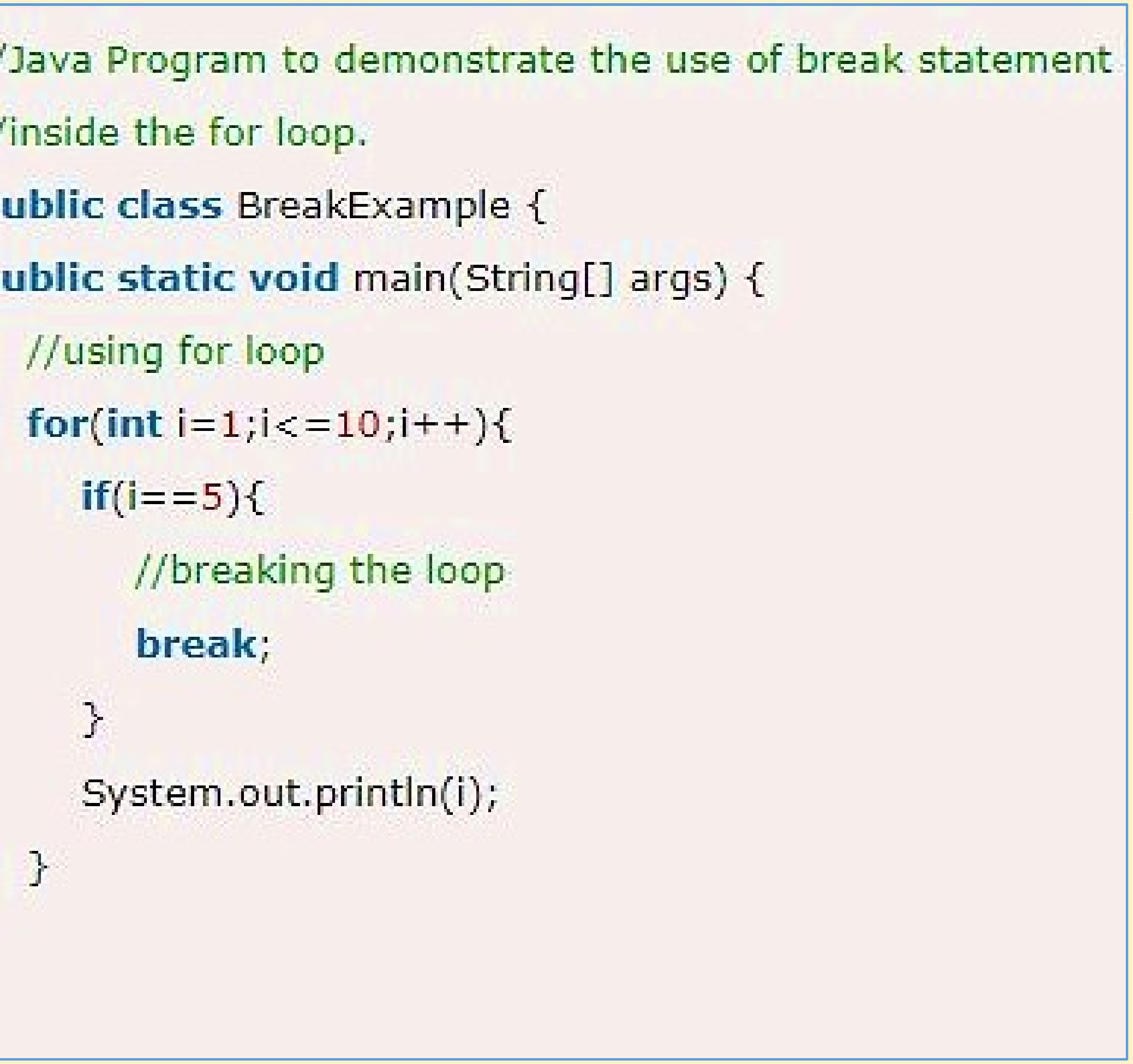
```
lass SampleBreak

  public static void main(String args[])
  {
        int num= 1;
        while (num <= 10) {
          System.out.println(num);
          if(num==5)
          {
              break;
          }
          num++;
        }
  }
```



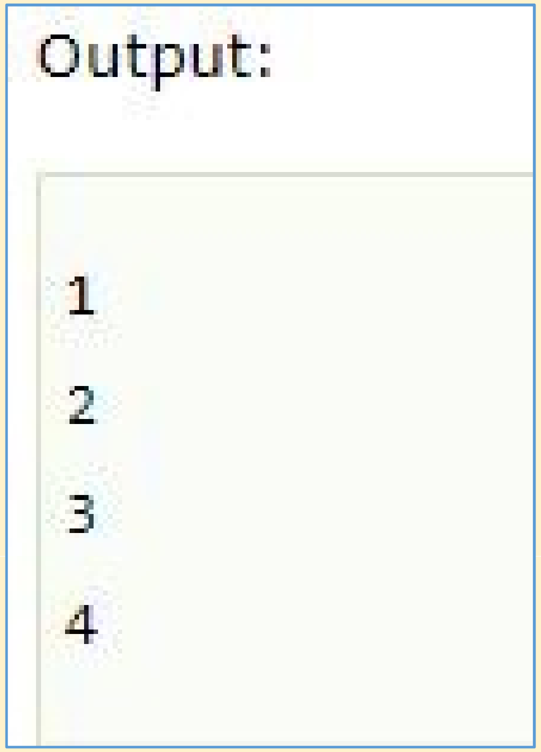
Output

```
1
2
3
4
5
```

xample 2



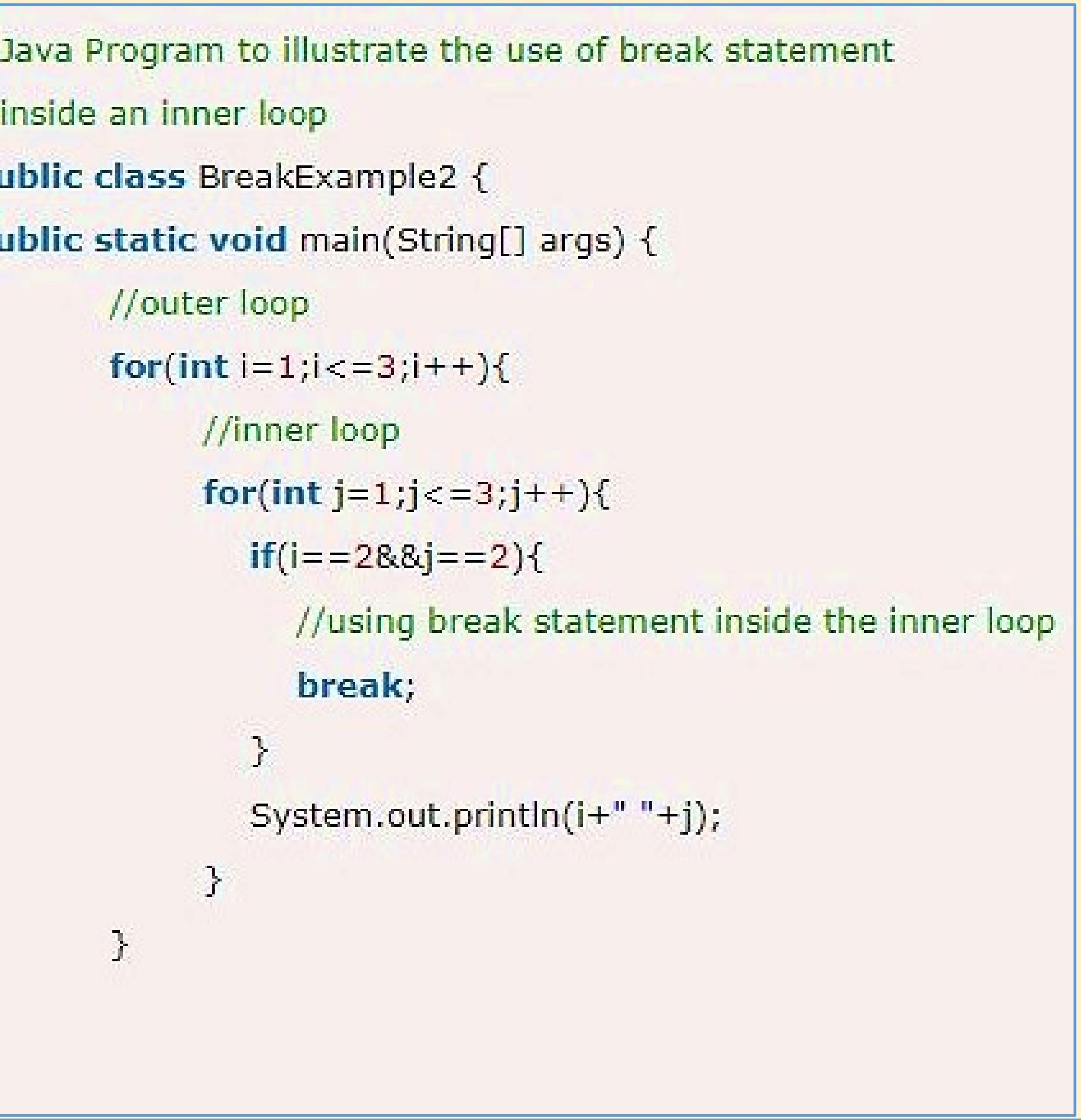
```
'Java Program to demonstrate the use of break statement
'inside the for loop.
ublic class BreakExample {
ublic static void main(String[] args) {
  //using for loop
  for(int i=1;i<=10;i++){
      if(i==5){
          //breaking the loop
          break;
      }
      System.out.println(i);
  }
```



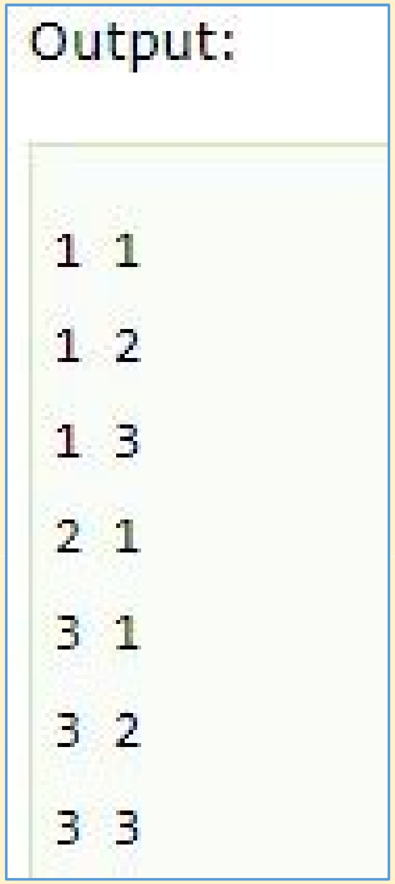
Output:

```
1
2
3
4
```

xample 3



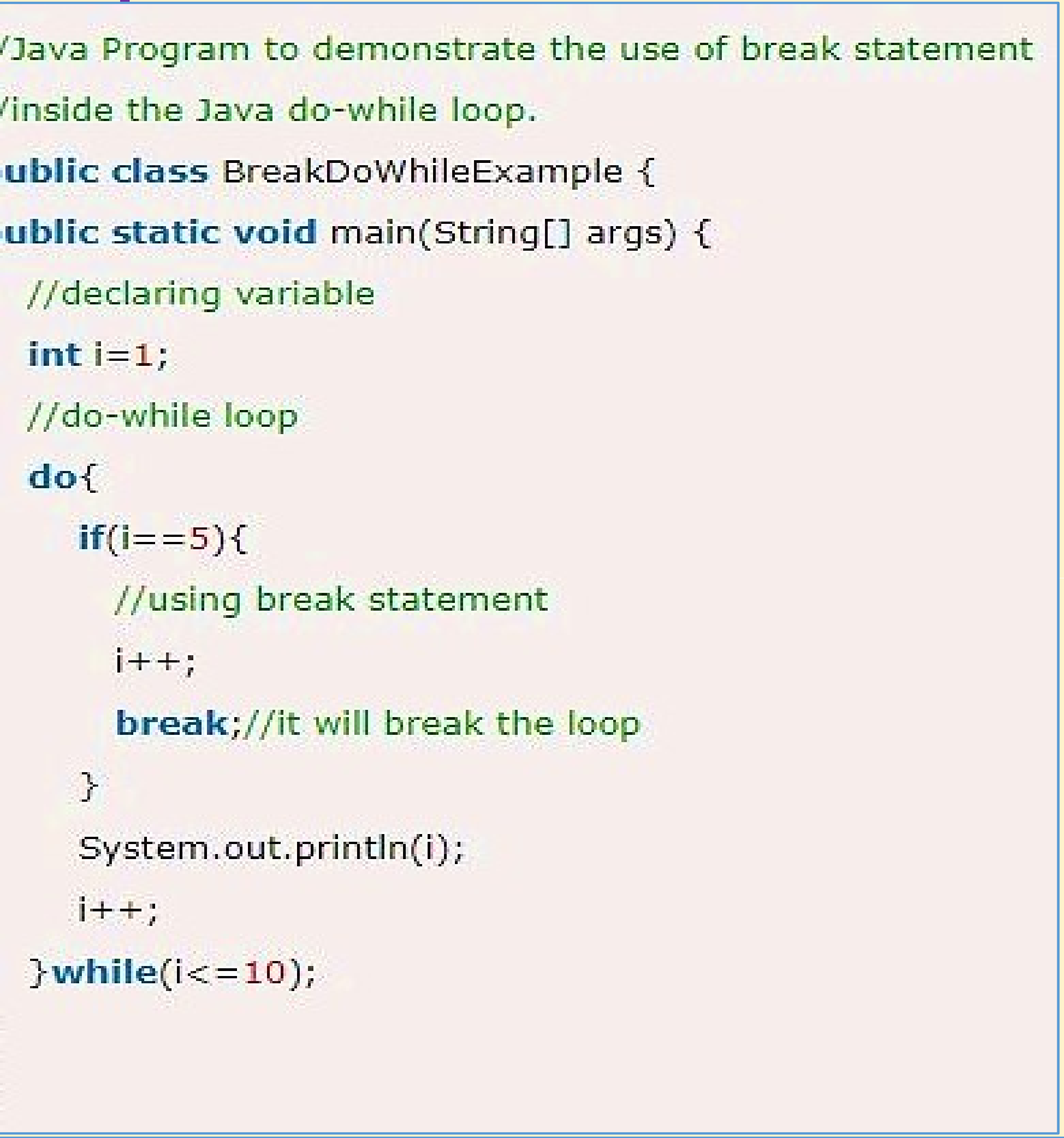
```
Java Program to illustrate the use of break statement
inside an inner loop
ublic class BreakExample2 {
ublic static void main(String[] args) {
    //outer loop
    for(int i=1;i<=3;i++){
        //inner loop
        for(int j=1;j<=3;j++){
            if(i==2&&j==2){
                //using break statement inside the inner loop
                break;
            }
            System.out.println(i+" "+j);
        }
    }
```



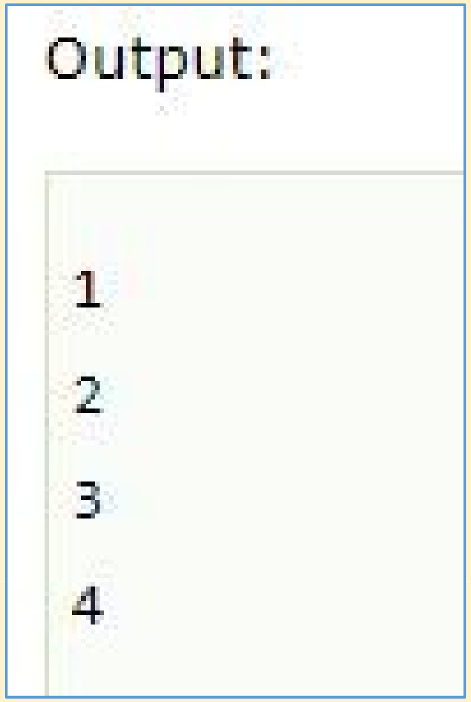
Output:

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

xample 4



```
/Java Program to demonstrate the use of break statement
/inside the Java do-while loop.
ublic class BreakDoWhileExample {
ublic static void main(String[] args) {
  //declaring variable
  int i=1;
  //do-while loop
  do{
     if(i==5){
       //using break statement
       i++;
       break;//it will break the loop
     }
     System.out.println(i);
     i++;
  }while(i<=10);
```



Output:

```
1
2
3
4
```

Java Continue Statement

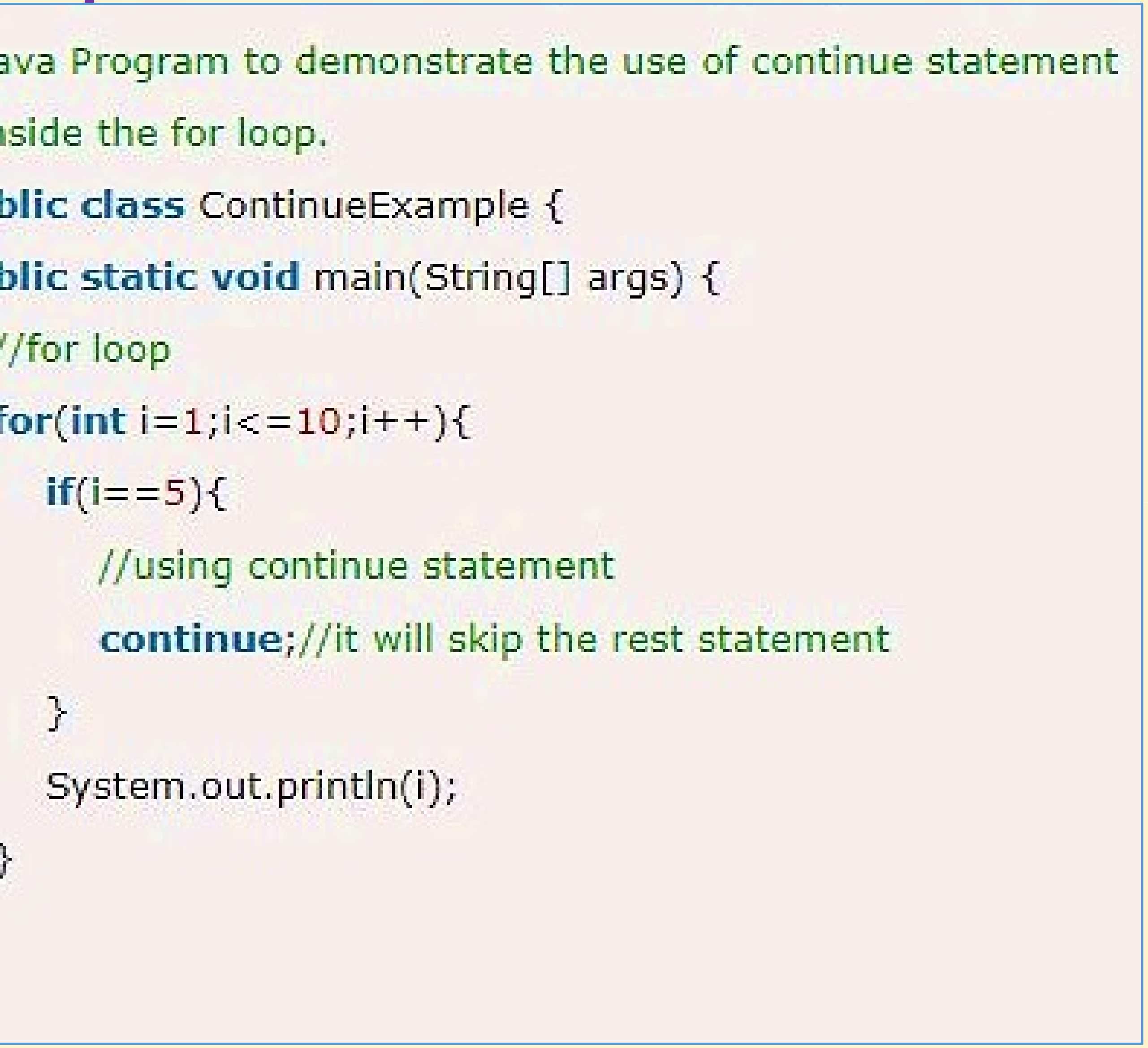
he Java continue statement is used to continue the loop

he continue statement is used in loop control structure when yo
eed to jump to the next iteration of the loop immediately

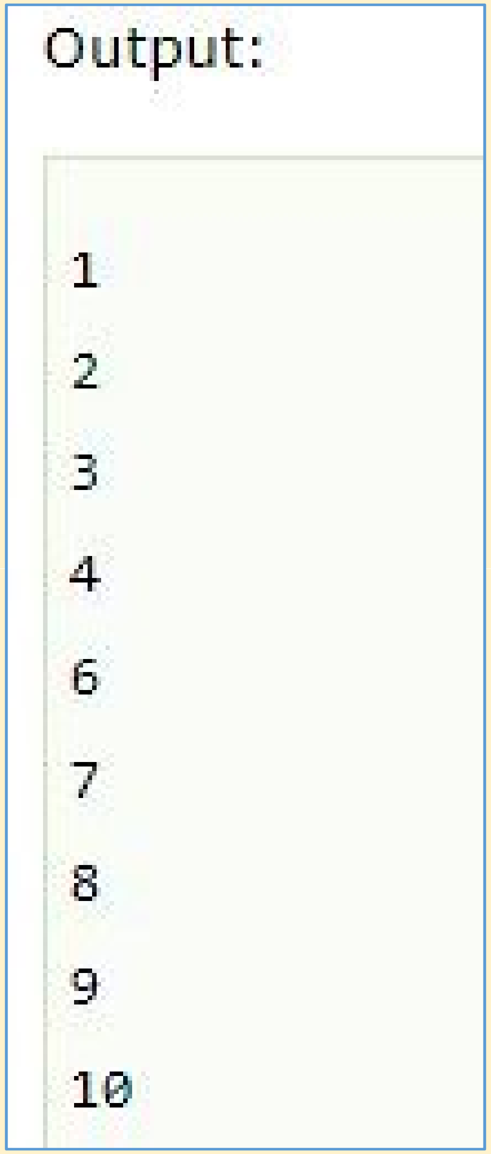
t continues the current flow of the program and skips the remaini
ode at the specified condition.

n case of an inner loop, it continues the inner loop only.

ample 1



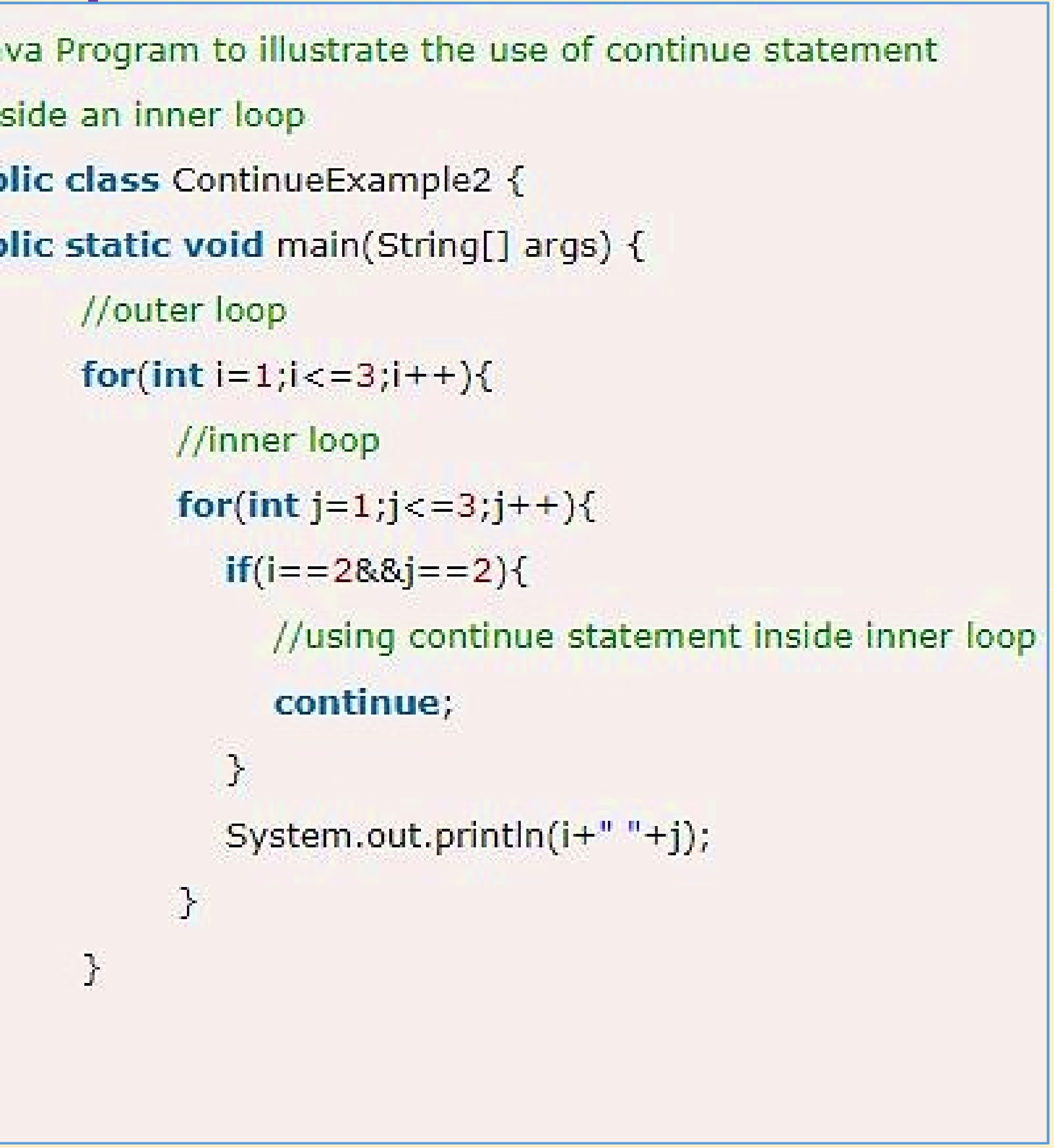
```
ava Program to demonstrate the use of continue statement
nside the for loop.
blic class ContinueExample {
blic static void main(String[] args) {
'/for loop
for(int i=1;i<=10;i++){
    if(i==5){
        //using continue statement
        continue;//it will skip the rest statement
    }
    System.out.println(i);
}
```



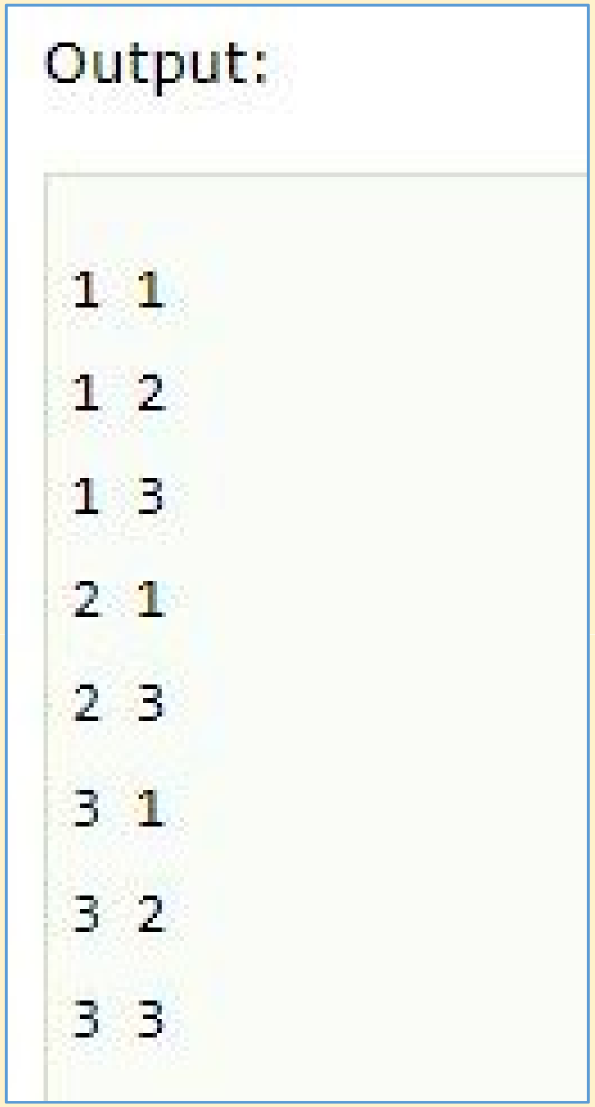
Output:

```
1
2
3
4
6
7
8
9
10
```

ample 2



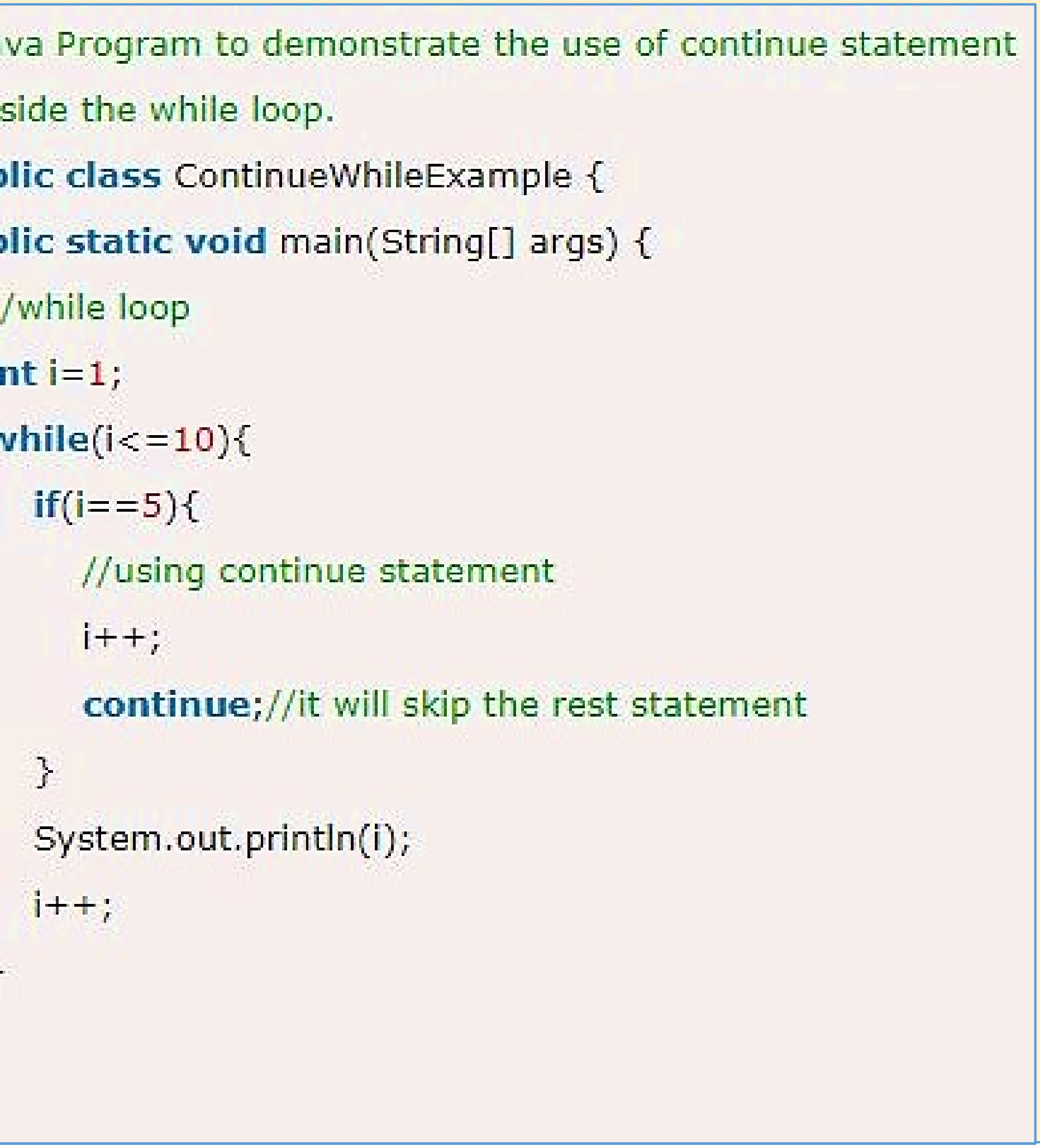
```
va Program to illustrate the use of continue statement
side an inner loop
olic class ContinueExample2 {
olic static void main(String[] args) {
      //outer loop
      for(int i=1;i<=3;i++){
            //inner loop
            for(int j=1;j<=3;j++){
               if(i==2&&j==2){
                  //using continue statement inside inner loop
                  continue;
               }
               System.out.println(i+" "+j);
            }
      }
```



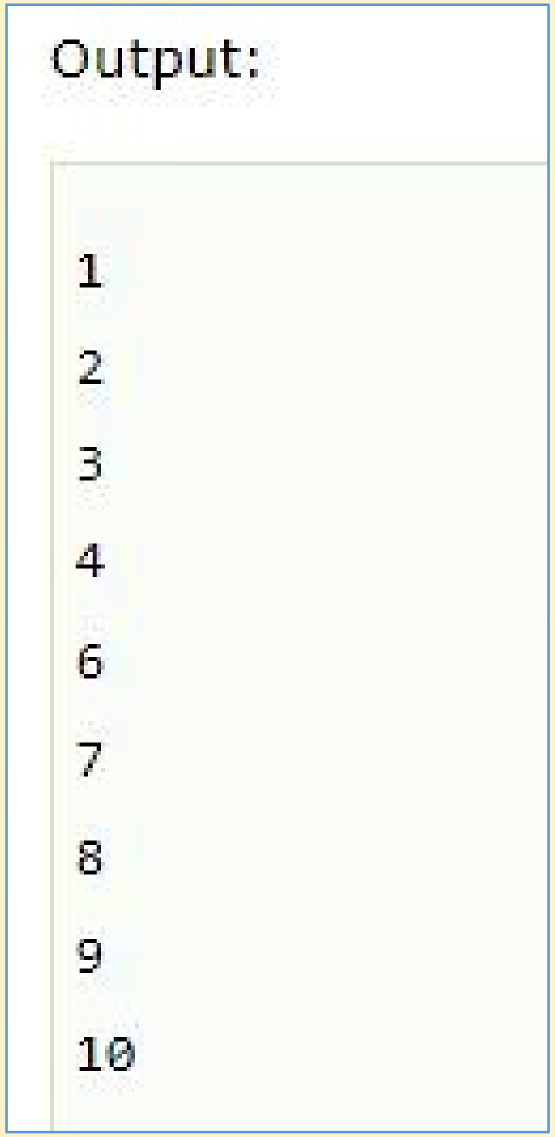
Output:

```
1 1
1 2
1 3
2 1
2 3
3 1
3 2
3 3
```

ample 3



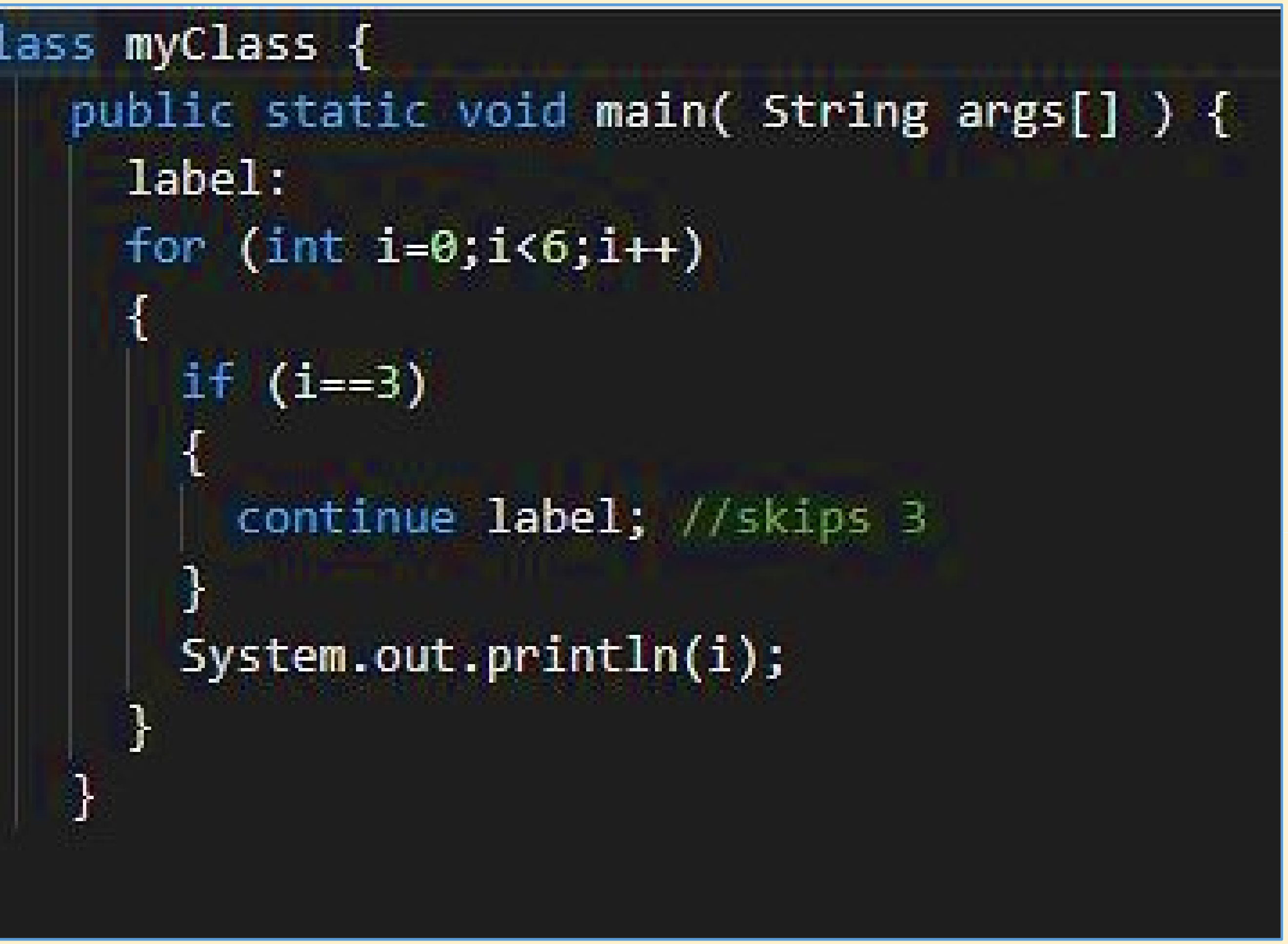
```
va Program to demonstrate the use of continue statement
side the while loop.
lic class ContinueWhileExample {
lic static void main(String[] args) {
/while loop
nt i=1;
vhile(i<=10){
   if(i==5){
      //using continue statement
      i++;
      continue;//it will skip the rest statement
   }
   System.out.println(i);
   i++;
```



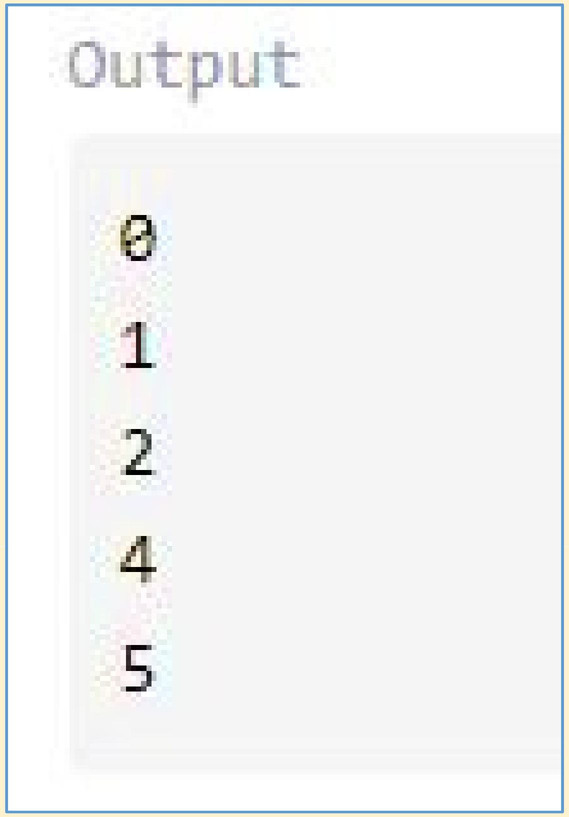
Output:

```
1
2
3
4
6
7
8
9
10
```

ample 4



```
lass myClass {
   public static void main( String args[] ) {
     label:
     for (int i=0;i<6;i++)
     {
       if (i==3)
       {
         continue label; //skips 3
       }
       System.out.println(i);
     }
   }
```



```
Output

0
1
2
4
5
```

ARRAY

An array is a collection of similar data types.

Java array is an object which contains elements of a similar dat type.

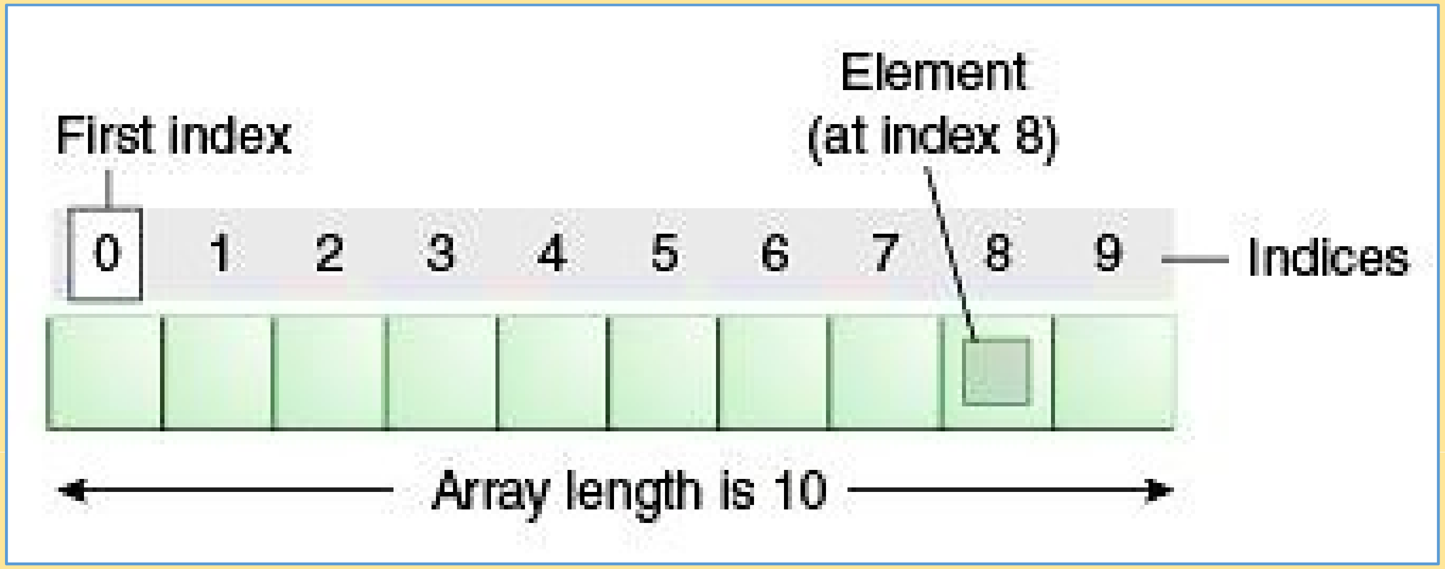
The elements of an array are stored in a contiguous memor location

the size of an array is fixed and cannot increase to accommodat more elements

It is also known as static data structure because size of an arra must be specified at the time of its declaration.

Array in Java is index-based, the first element of the array i stored at the 0th index

ava provides the feature of anonymous arrays which is n
vailable in C/C++.



dvantage of Java Array

Code Optimization: It makes the code optimized, we can retrieve
ort the data easily.

Random access: We can get any data located at any index positior

Disadvantage of Java Array

Size Limit: We can store the only fixed size of elements in th
array. It doesn't grow its size at runtime. To solve this problem
collection framework is used in java.

Features of Array

It is always indexed. The index begins from 0.

It is a collection of similar data types.

It occupies a contiguous memory location.

Types of Java Array

Single Dimensional Array

Multidimensional Array

❖ Single Dimensional Array in java

➢Array Declaration

yntax: datatype[ ] arrayname;

g: int[ ] arr;

char[ ] name;

short[ ] arr;

long[ ] arr;

int[ ][ ] arr; //two dimensional array

n C program datatype arrayname[];

➢Initialization of Array

new operator is used to initializing an array.

g 1: int[ ] arr = new int[10];

or

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

g 2: String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

g 3: double[] myList = new double[10];

➢**Accessing array element**

xample: To access 4th element of a given array

```
    int[ ] arr = {10,24,30,50};
    System.out.println("Element at 4th place" + arr[3]);
```

➢To find the length of an array, we can use the following syntax

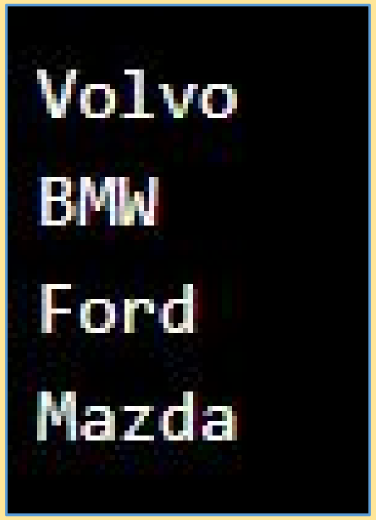
array_name.length

xample:
```
public class MyClass
  {
    public static void main(String[] args)
     {
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
        System.out.println(cars.length);
     }
  }
```

Output 4

➢Loop Through an Array

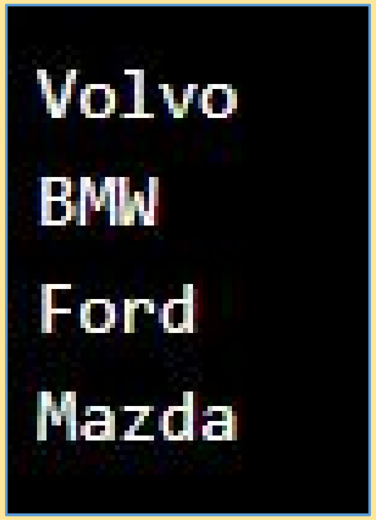
```
ublic class MyClass

 public static void main(String[] args)
    {
      String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
      for (int i = 0; i < cars.length; i++)
        {
          System.out.println(cars[i]);
        }
    }
```



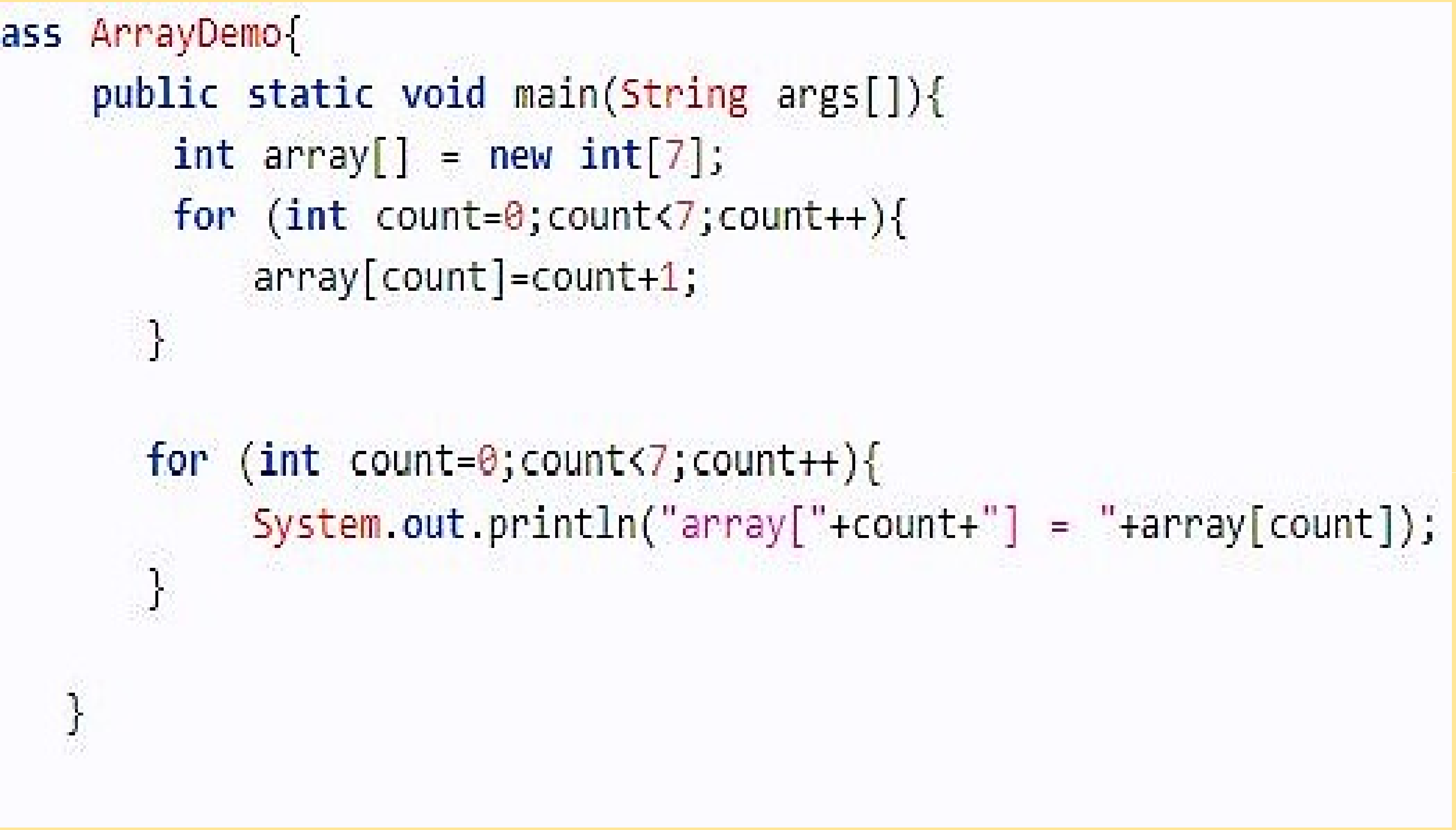
```
Volvo
BMW
Ford
Mazda
```

➢Loop Through an Array with For-Each

```
ublic class MyClass

  public static void main(String[] args)
     {
      String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
      for (String i : cars)
        {
            System.out.println(i);
        }
     }
```

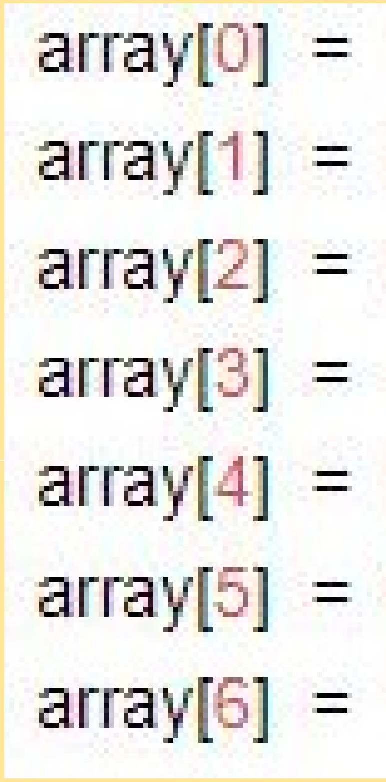


```
Volvo
BMW
Ford
Mazda
```

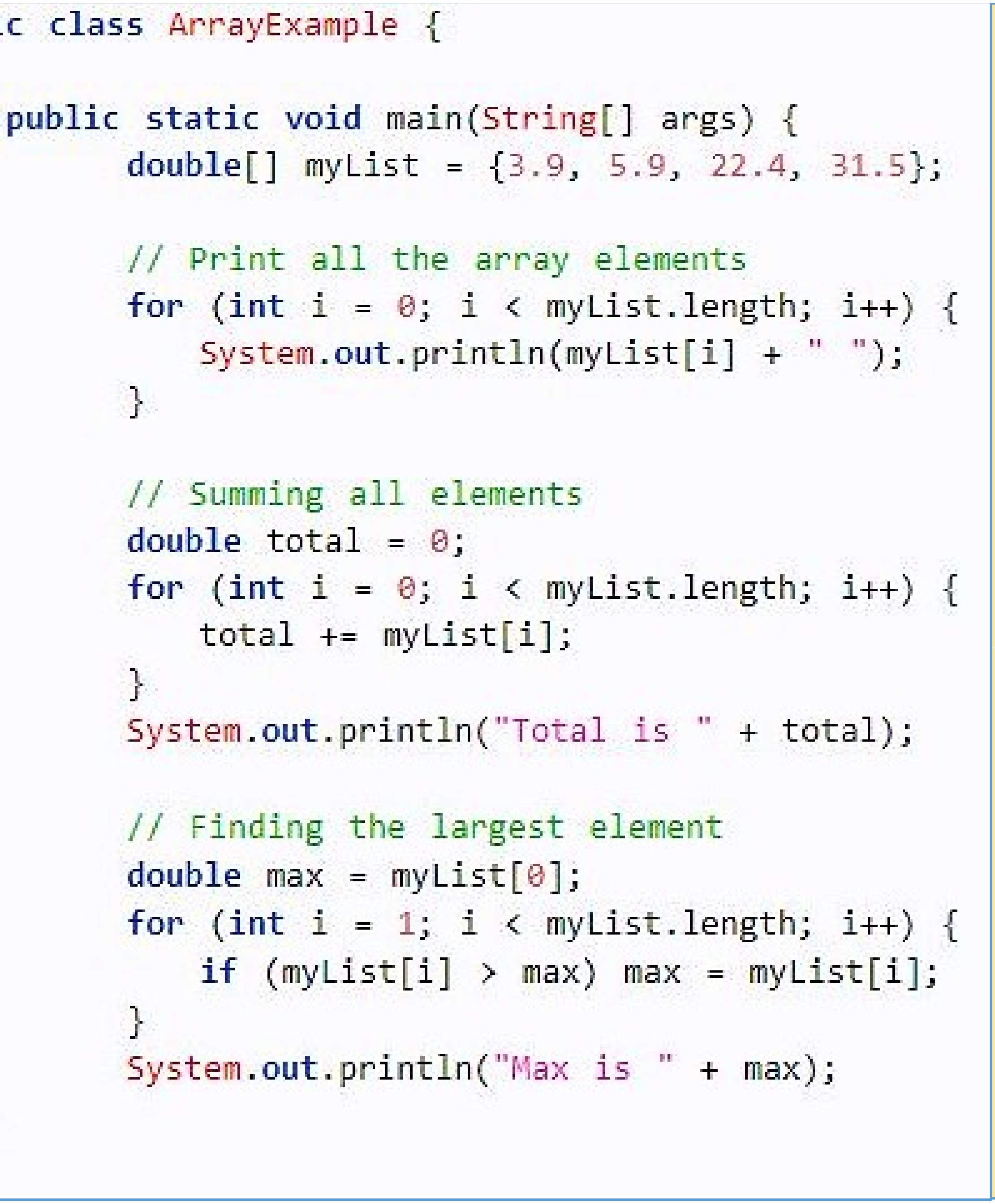


```
ass ArrayDemo{
    public static void main(String args[]){
        int array[] = new int[7];
        for (int count=0;count<7;count++){
            array[count]=count+1;
      }

      for (int count=0;count<7;count++){
            System.out.println("array["+count+"] = "+array[count]);
      }

  }
```

Output

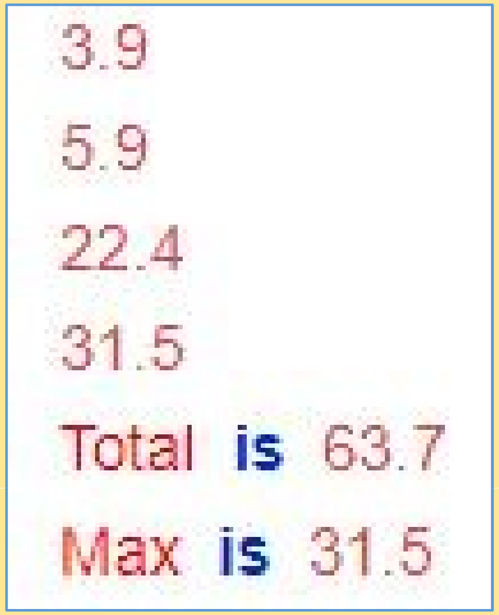


```
array[0] =
array[1] =
array[2] =
array[3] =
array[4] =
array[5] =
array[6] =
```



```
c class ArrayExample {

public static void main(String[] args) {
    double[] myList = {3.9, 5.9, 22.4, 31.5};

    // Print all the array elements
    for (int i = 0; i < myList.length; i++) {
        System.out.println(myList[i] + " ");
    }

    // Summing all elements
    double total = 0;
    for (int i = 0; i < myList.length; i++) {
        total += myList[i];
    }
    System.out.println("Total is " + total);

    // Finding the largest element
    double max = myList[0];
    for (int i = 1; i < myList.length; i++) {
        if (myList[i] > max) max = myList[i];
    }
    System.out.println("Max is " + max);
```

Output



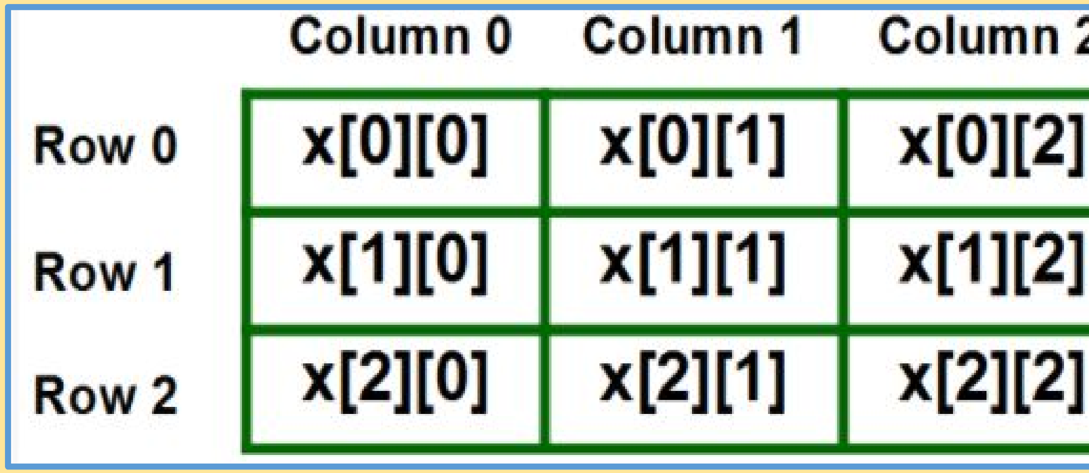
```
3.9
5.9
22.4
31.5
Total is 63.7
Max is 31.5
```

Two Dimensional array

Array Declaration

ntax : datatype[ ][ ] arrayname;

g: int[][] myNumbers ;



	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

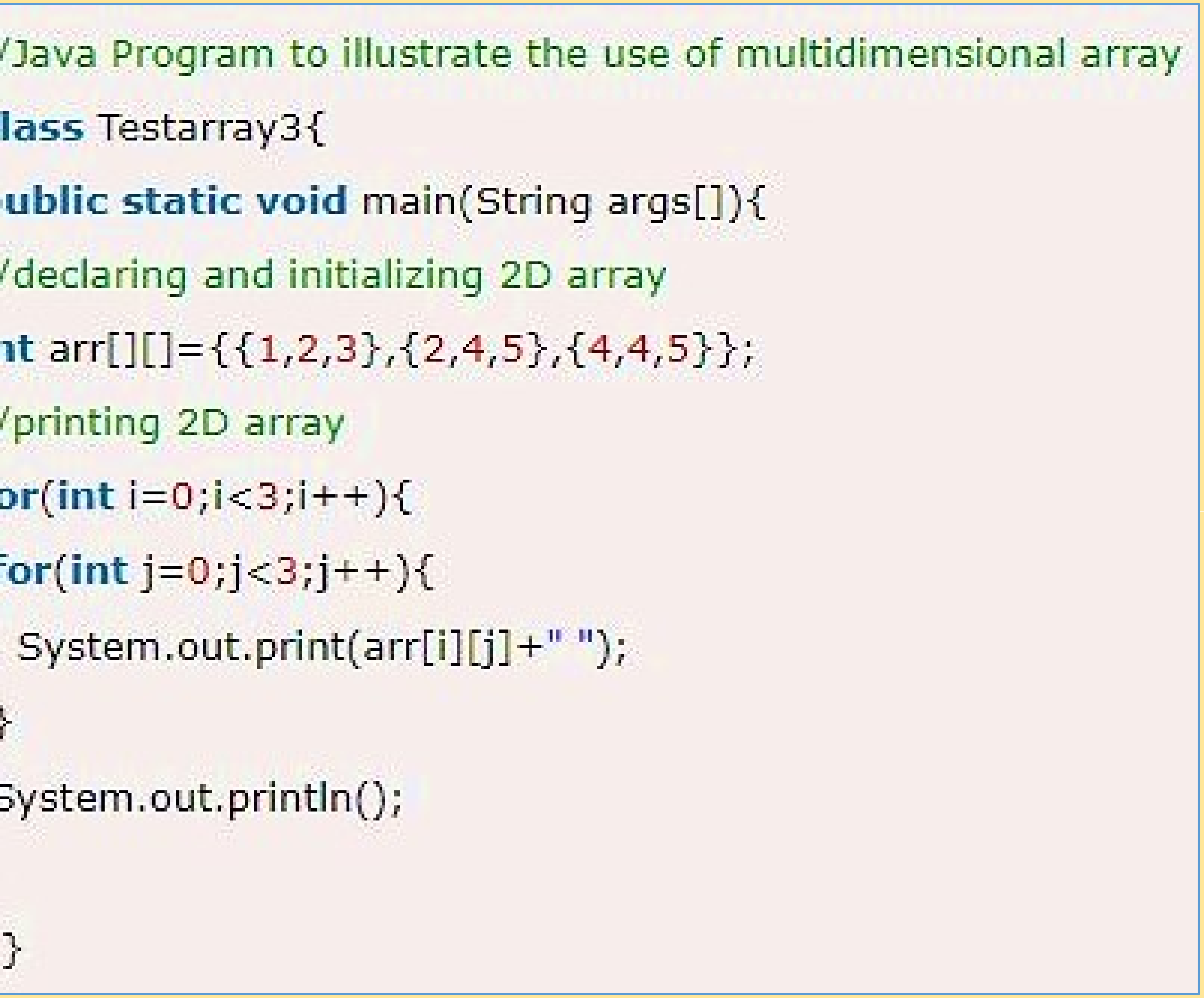
Array Initialization

```
int[ ][ ] arrName = new int[10][10];
```

Or

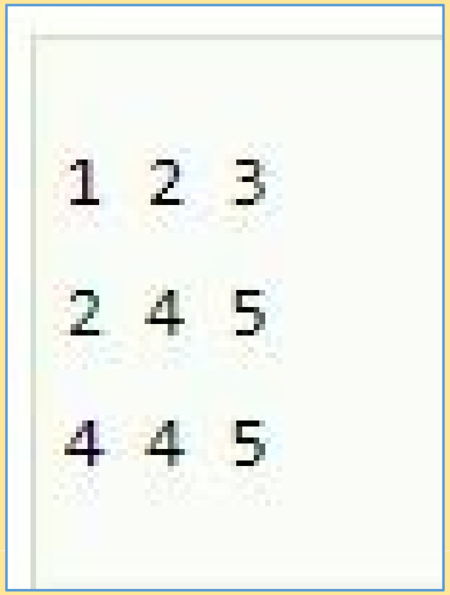
```
nt[ ][ ] arrName = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};  // 3 b
```

is the size of the array.

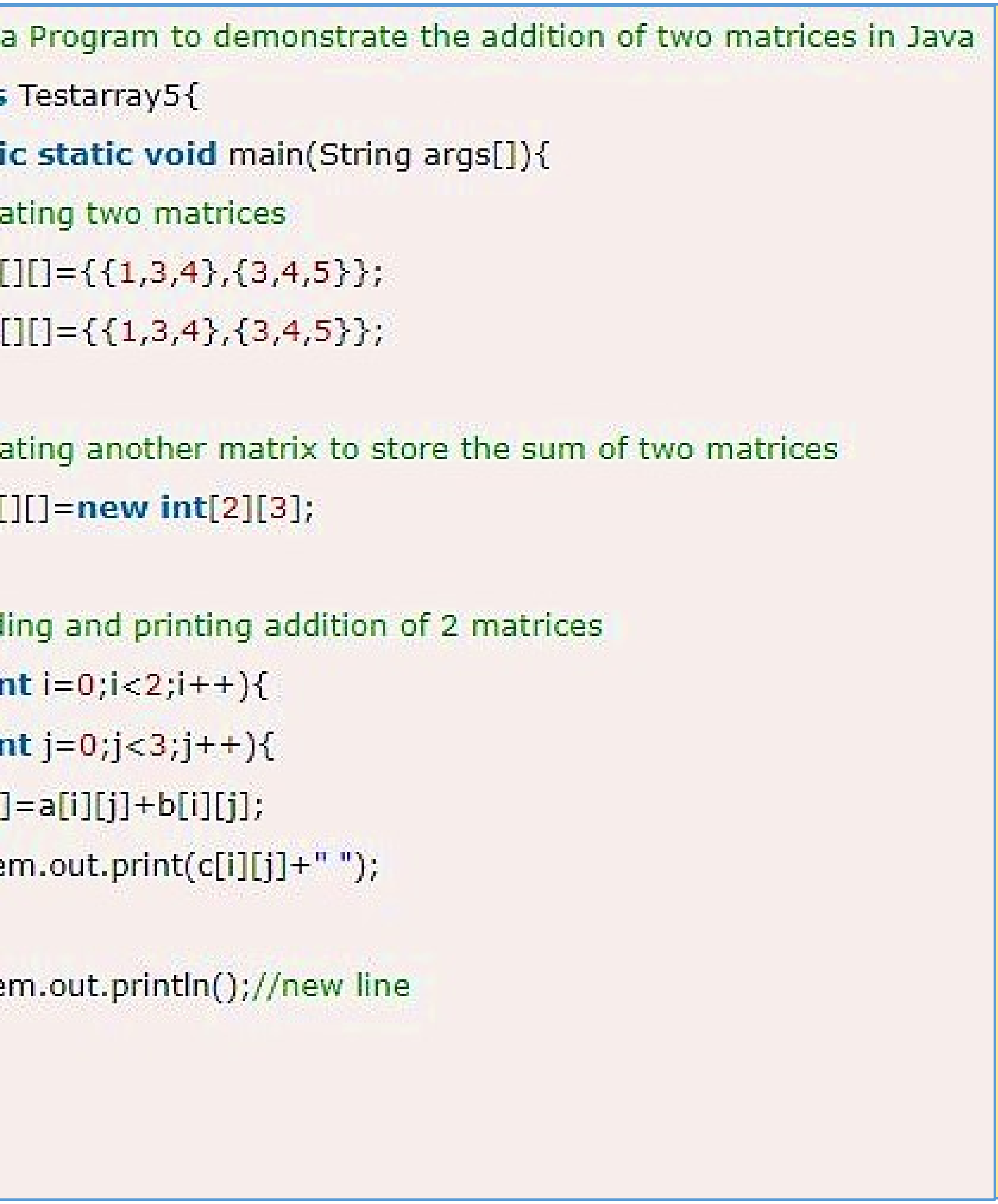


```
/Java Program to illustrate the use of multidimensional array
lass Testarray3{
ublic static void main(String args[]){
/declaring and initializing 2D array
nt arr[][]={{1,2,3},{2,4,5},{4,4,5}};
/printing 2D array
or(int i=0;i<3;i++){
for(int j=0;j<3;j++){
 System.out.print(arr[i][j]+" ");
}
System.out.println();

}
```

Output

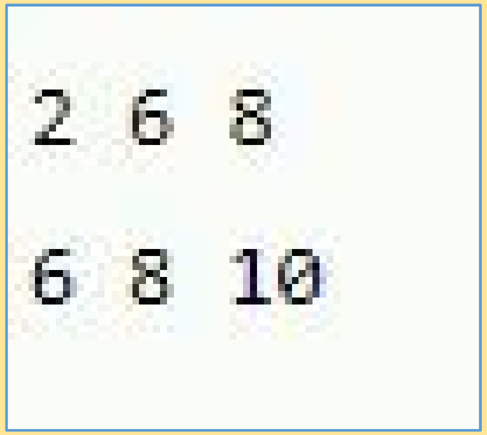


```
1 2 3
2 4 5
4 4 5
```



```
a Program to demonstrate the addition of two matrices in Java
s Testarray5{
ic static void main(String args[]){
ating two matrices
[][]={{1,3,4},{3,4,5}};
[][]={{1,3,4},{3,4,5}};

ating another matrix to store the sum of two matrices
[][]=new int[2][3];

ling and printing addition of 2 matrices
nt i=0;i<2;i++){
nt j=0;j<3;j++){
]=a[i][j]+b[i][j];
em.out.print(c[i][j]+" ");

em.out.println();//new line
```

Output



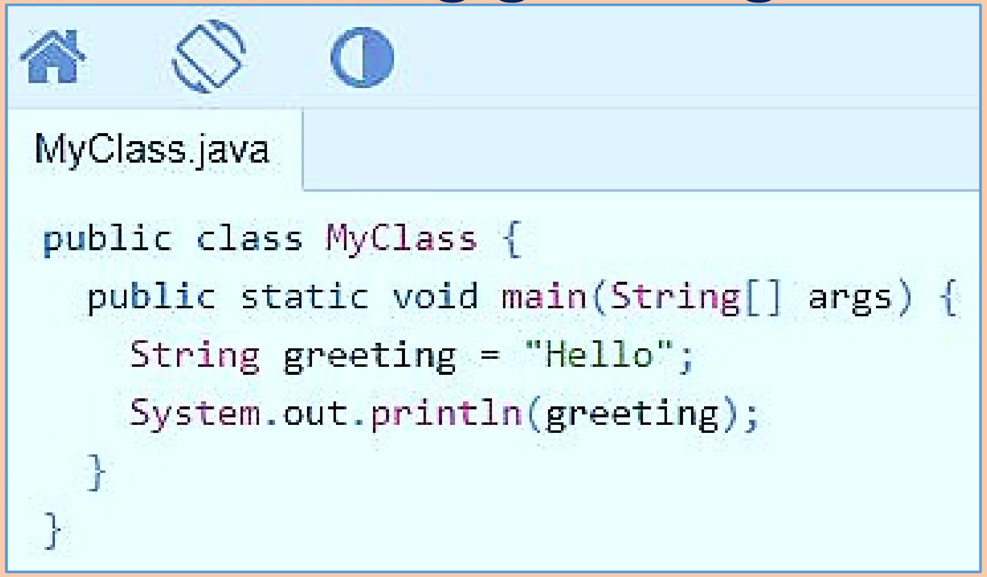
```
2 6 8
6 8 10
```

STRING

- Strings are used for storing text
- A String variable contains a collection of characters surrounded by double quotes

Eg: Create a variable of type String and assign it a value

String greeting = "Hello";



```
public class MyClass {
  public static void main(String[] args) {
    String greeting = "Hello";
    System.out.println(greeting);
  }
}
```

Output

```
Hello
```

n Java, string is basically an object that represents sequence char values

An array of characters works same as Java string. For example:

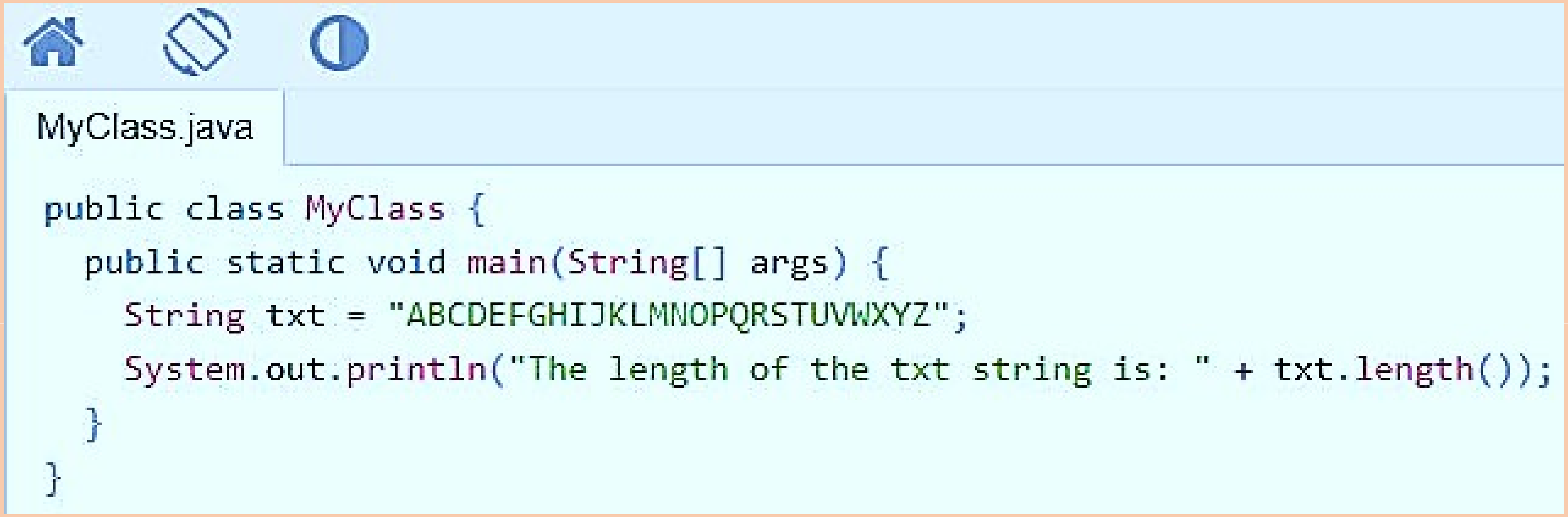
```
char[] ch={'j',‘o',‘s',‘e',‘p',‘h'};
String s=new String(ch);   //converting char array to string
```

is same as

```
String s="joseph";     //creating string by java string literal
```

String Length

The length of a string can be found with the length() method

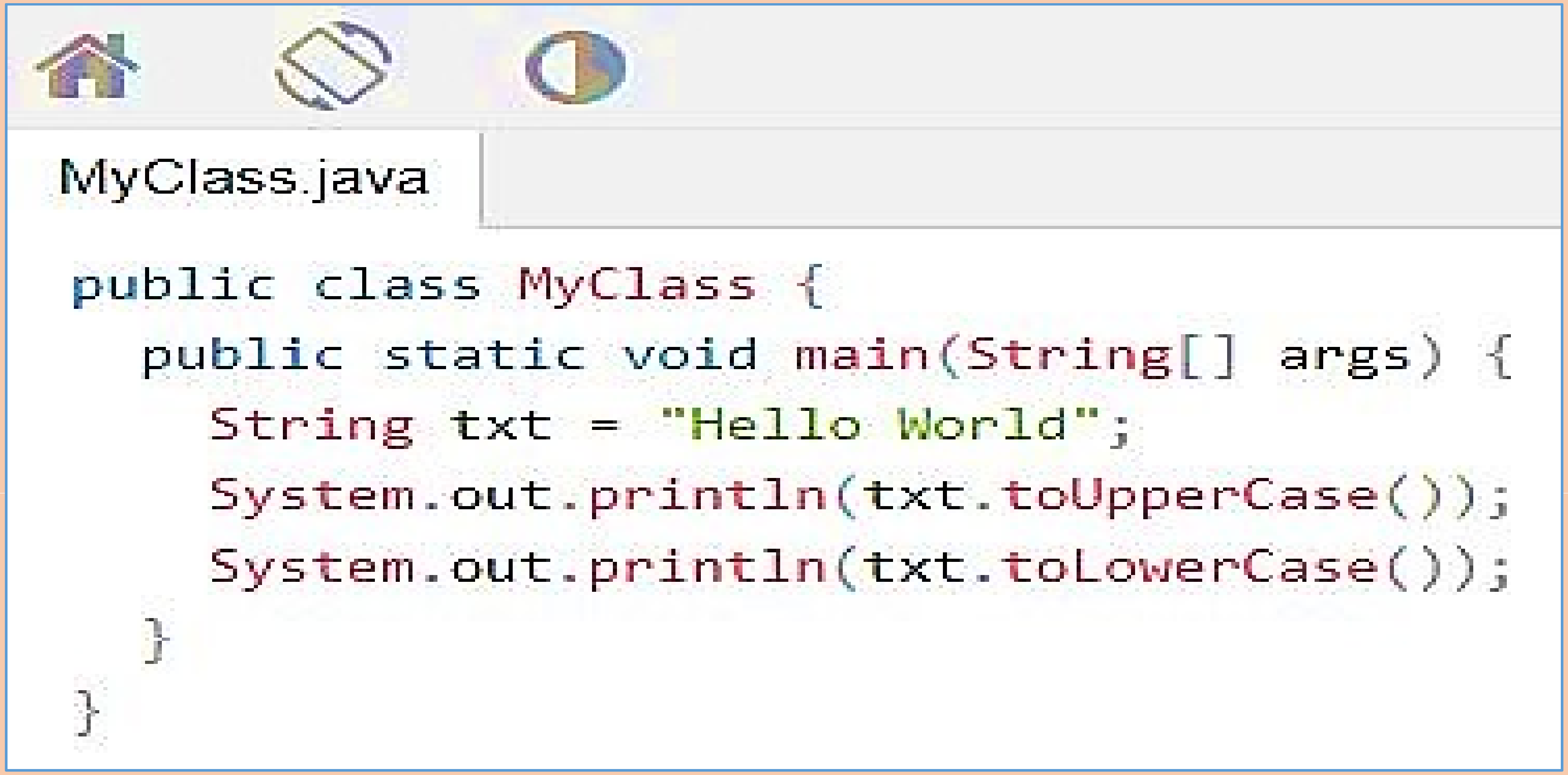


```
public class MyClass {
  public static void main(String[] args) {
    String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    System.out.println("The length of the txt string is: " + txt.length());
  }
}
```

utput

```
he length of the txt string is: 26
```

toUpperCase() and toLowerCase()



MyClass.java

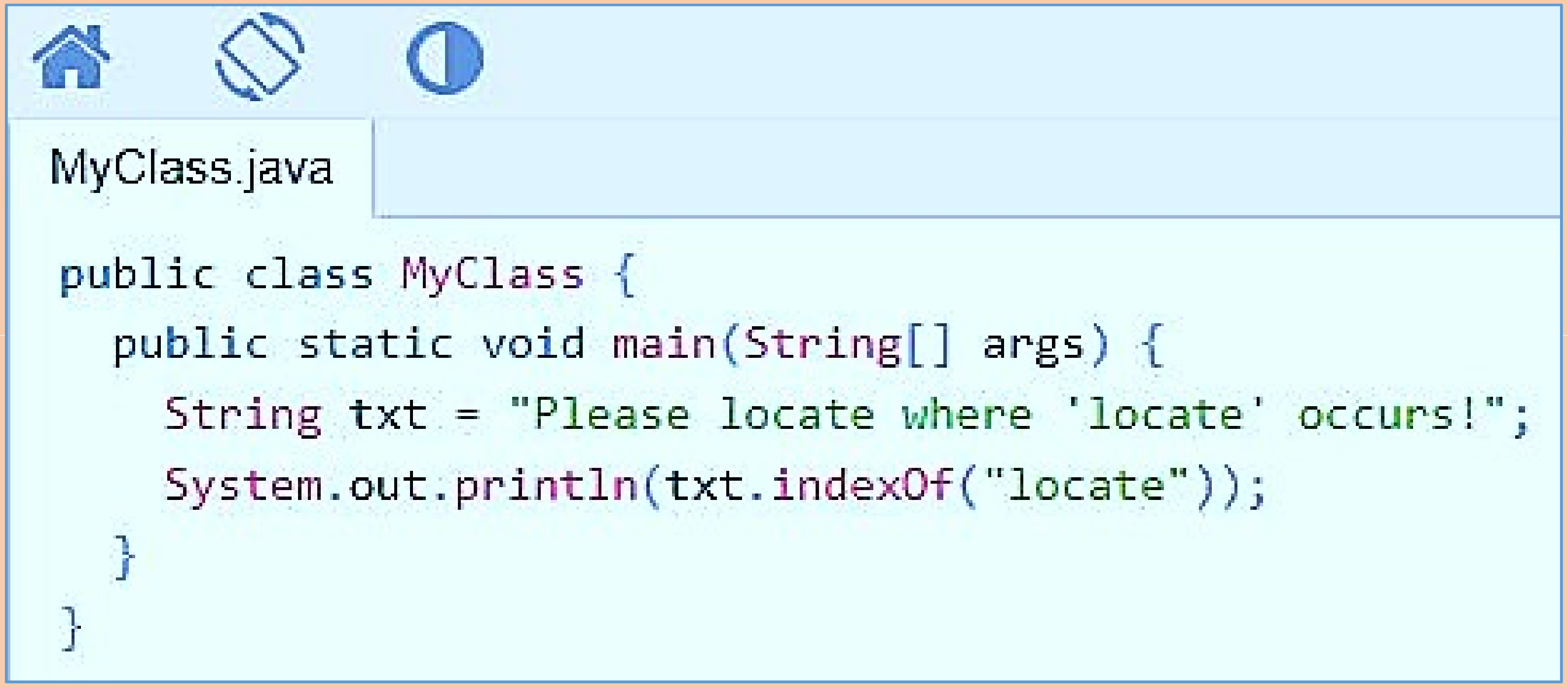
```
public class MyClass {
  public static void main(String[] args) {
    String txt = "Hello World";
    System.out.println(txt.toUpperCase());
    System.out.println(txt.toLowerCase());
  }
}
```

utput

```
ELLO WORLD
ello world
```

Finding a Character in a String

The indexOf() method returns the index (the position) of the fi
ccurrence of a specified text in a string (including whitespace)



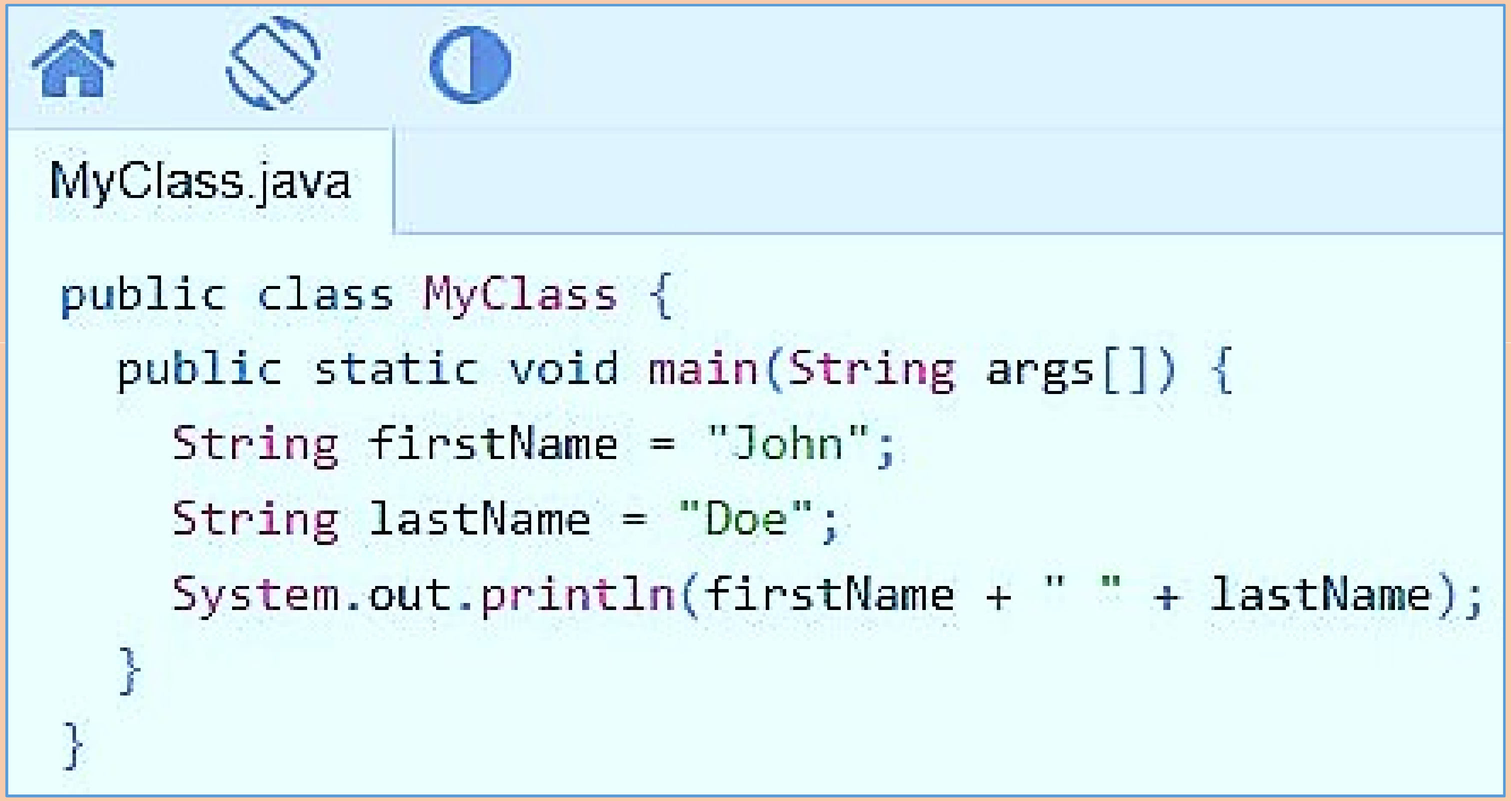
MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    String txt = "Please locate where 'locate' occurs!";
    System.out.println(txt.indexOf("locate"));
  }
}
```

utput 7

String Concatenation

he + operator can be used between strings to combine them. Th
s called concatenation

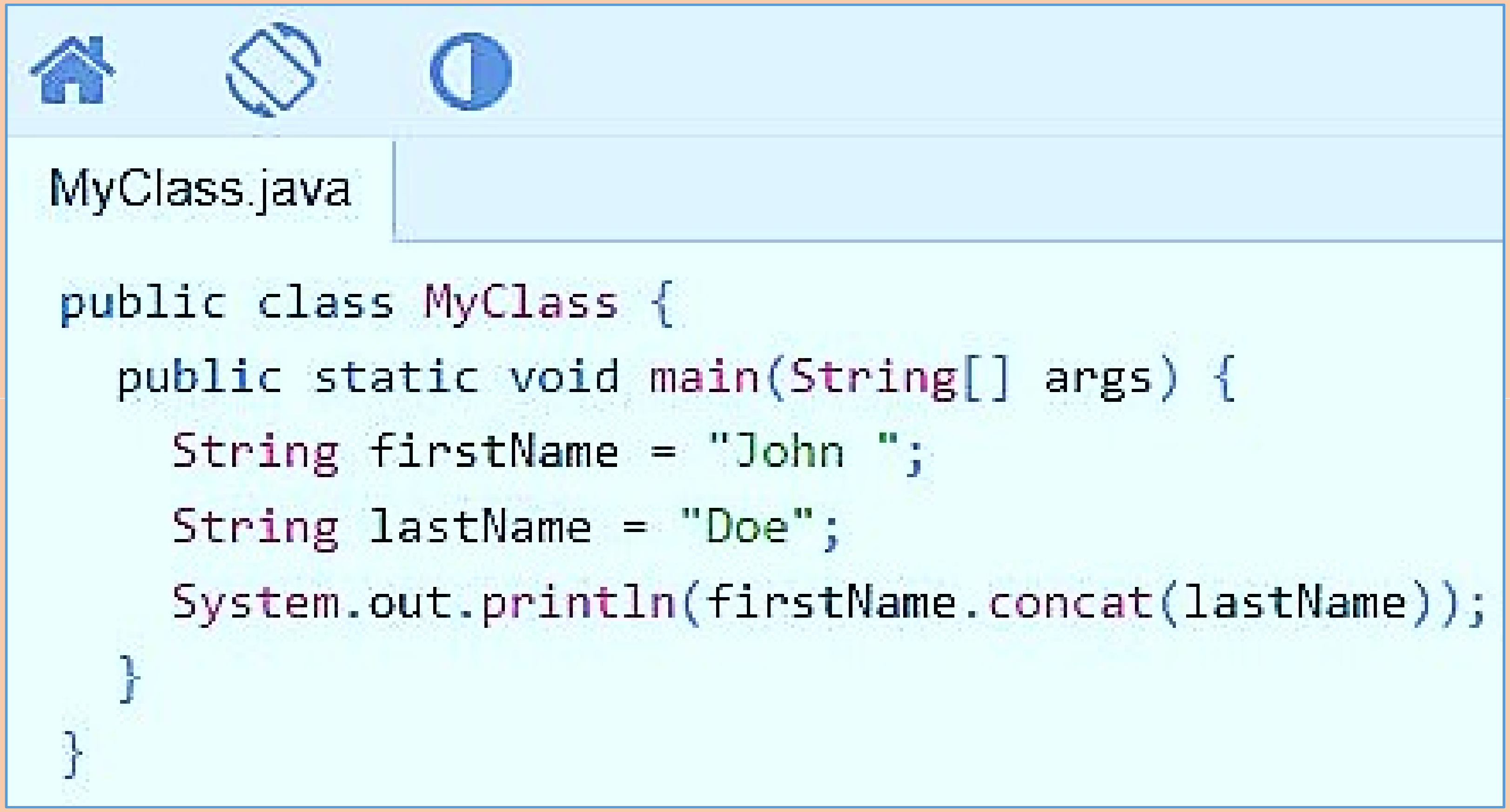


```
public class MyClass {
  public static void main(String args[]) {
    String firstName = "John";
    String lastName = "Doe";
    System.out.println(firstName + " " + lastName);
  }
}
```

utput John Doe

concat() method

We can also use the concat() method to concatenate two strings:



MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    String firstName = "John ";
    String lastName = "Doe";
    System.out.println(firstName.concat(lastName));
  }
}
```

utput John Doe

Special Characters

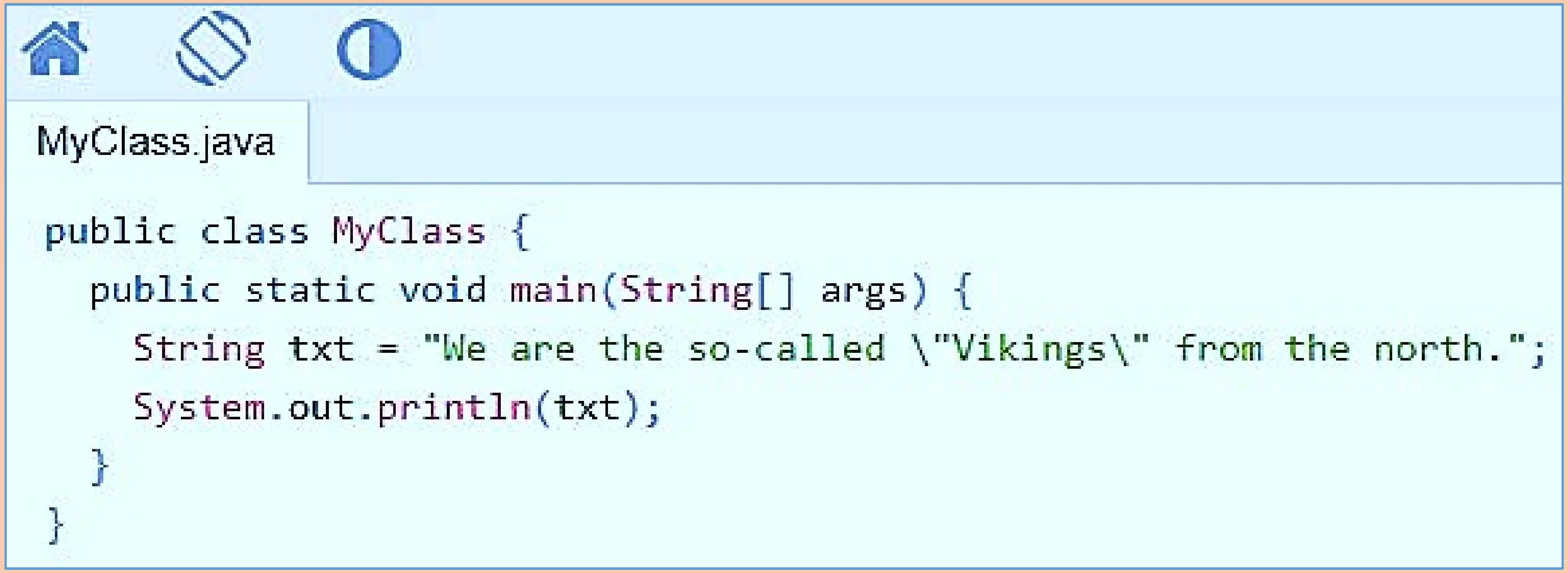
onsider the following example

String txt = "We are the so-called "Vikings" from the north.";

Because strings must be written within quotes, Java w
misunderstand this string

The solution to avoid this problem, is to use the backslash escap
character

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash



```java
public class MyClass {
  public static void main(String[] args) {
    String txt = "We are the so-called \"Vikings\" from the north.";
    System.out.println(txt);
  }
}
```

utput We are the so-called "Vikings" from the north.

The sequence \" inserts a double quote in a string

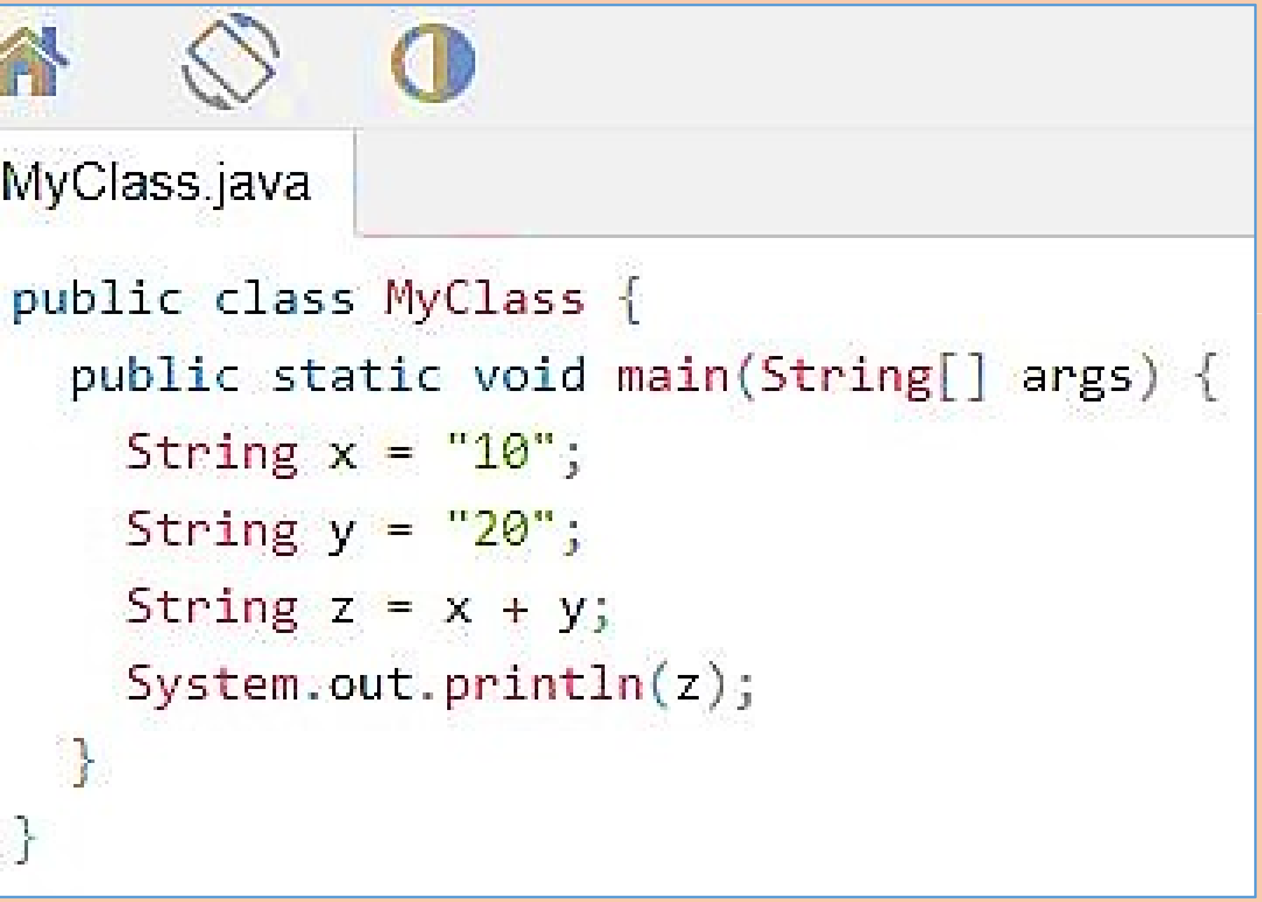
The sequence \' inserts a single quote in a string

The sequence \\ inserts a single backslash in a string

Adding Numbers and Strings

ava uses the + operator for both addition and concatenation.

f we add two strings, the result will be a string concatenation

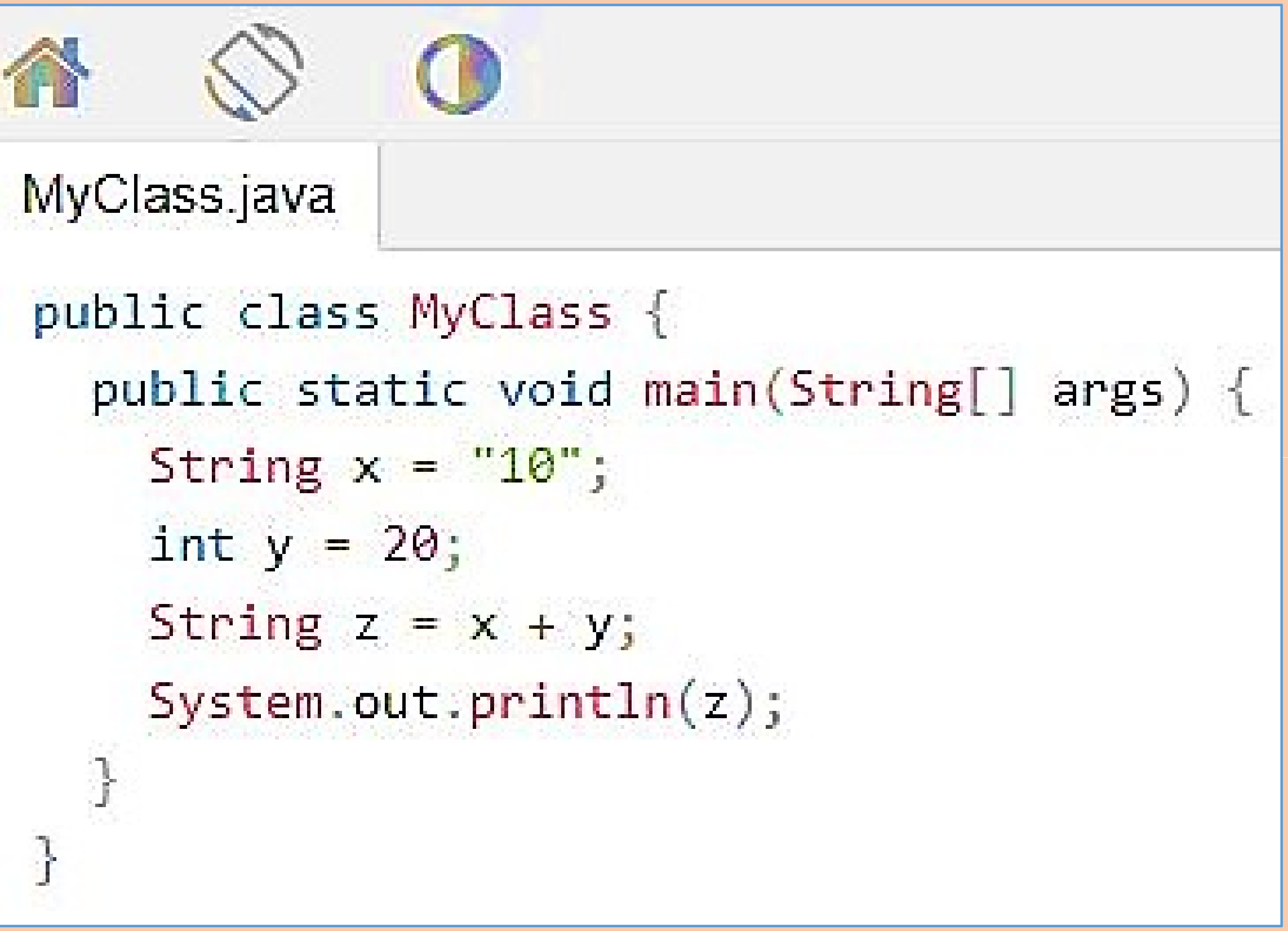


```
public class MyClass {
  public static void main(String[] args) {
    String x = "10";
    String y = "20";
    String z = x + y;
    System.out.println(z);
  }
}
```

Output

```
1020
```

f we add a number and a string, the result will be a stri
concatenation



```
public class MyClass {
  public static void main(String[] args) {
    String x = "10";
    int y = 20;
    String z = x + y;
    System.out.println(z);
  }
}
```

Output

```
1020
```

MODULE - 2

CHAPTER – 3
INHERITANCE

INHERITANCE IN JAVA

Inheritance in Java is a mechanism in which one object acquire all the properties and behaviors of a parent object.

The idea behind inheritance in Java is that you can create nev classes that are built upon existing classes.

When you inherit from an existing class, you can reuse method and attributes of the parent class. Moreover, you can add nev methods and attributes in your current class also

Inheritance represents the **IS-A relationship** which is also know as a *parent-child* relationship.

Terms used in Inheritance

lass: A class is a template or blueprint from which objects ar reated.

ub Class/Child Class: Subclass is a class which inherits the othe ass. It is also called a derived class, extended class, or child class.

uper Class/Parent Class: Superclass is the class from where ubclass inherits the features. It is also called a base class or arent class.

eusability: As the name specifies, reusability is a mechanism whic cilitates you to reuse the attributes and methods of the existin ass when you create a new class. We can use the same attribute nd methods already defined in the previous class.

EDULINE

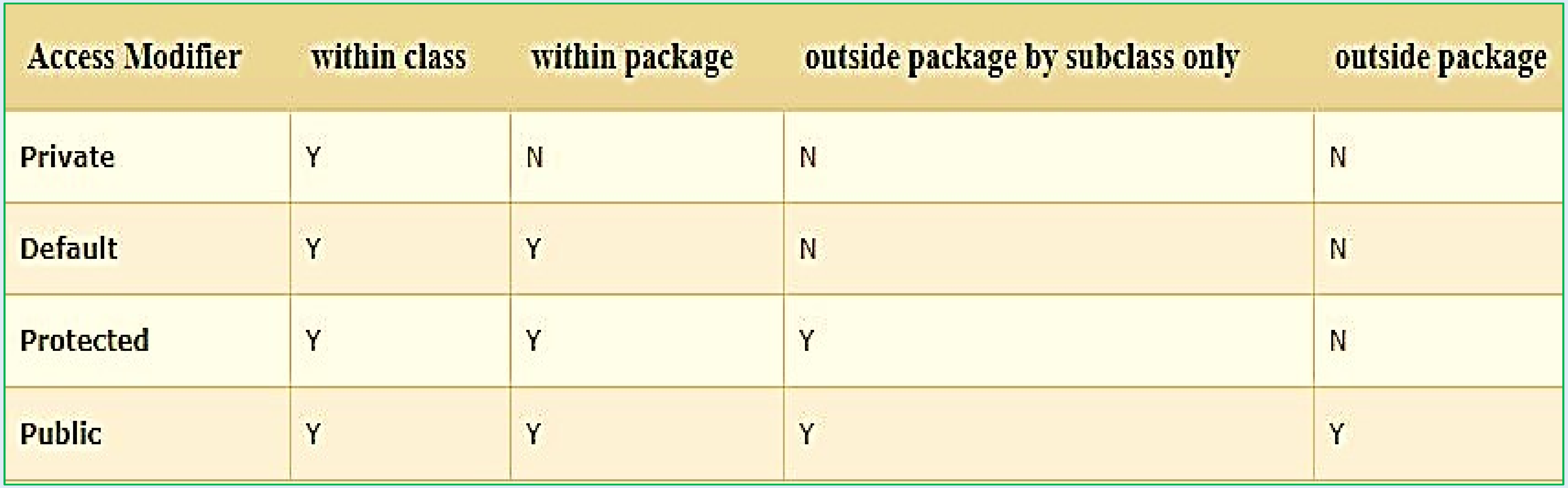
➢Access Modifiers - There are four types of Java access modifiers:

rivate: The access level of a private modifier is only within th
ass. It cannot be accessed from outside the class.

efault: The access level of a default modifier is only within th
ackage. It cannot be accessed from outside the package. If you d
ot specify any access level, it will be the default.

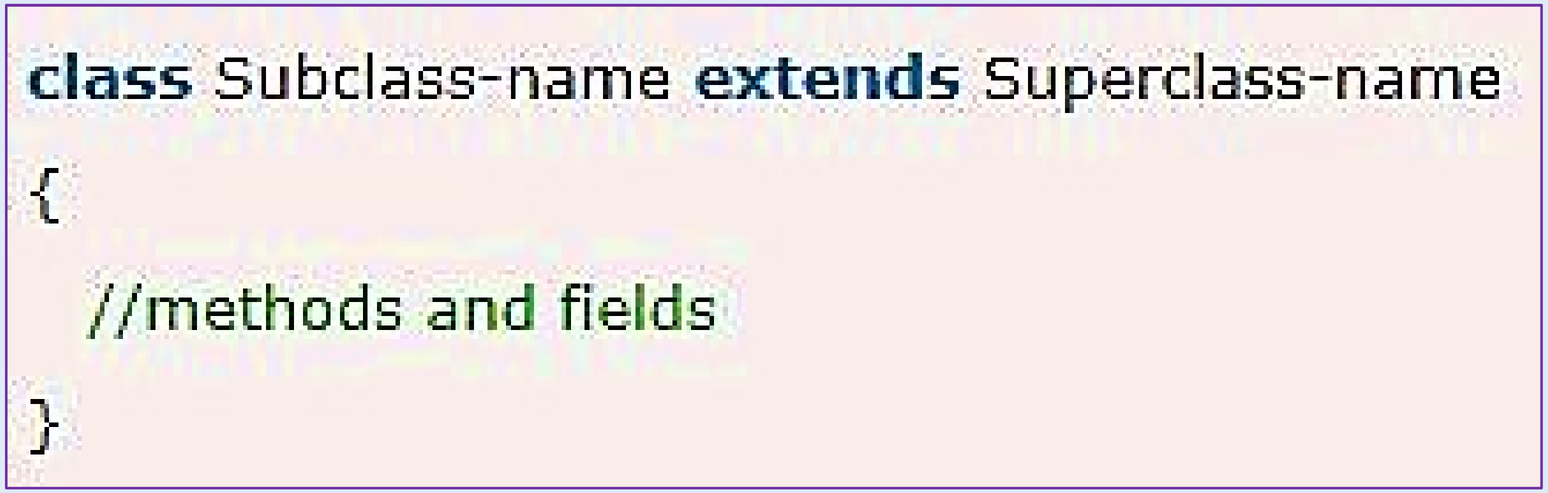
rotected: The access level of a protected modifier is within th
ackage and outside the package through child class. If you do no
nake the child class, it cannot be accessed from outside th
ackage.

ublic: The access level of a public modifier is everywhere. It can b
ccessed from within the class, outside the class, within th
ackage and outside the package.



Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

The syntax of Java Inheritance



```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

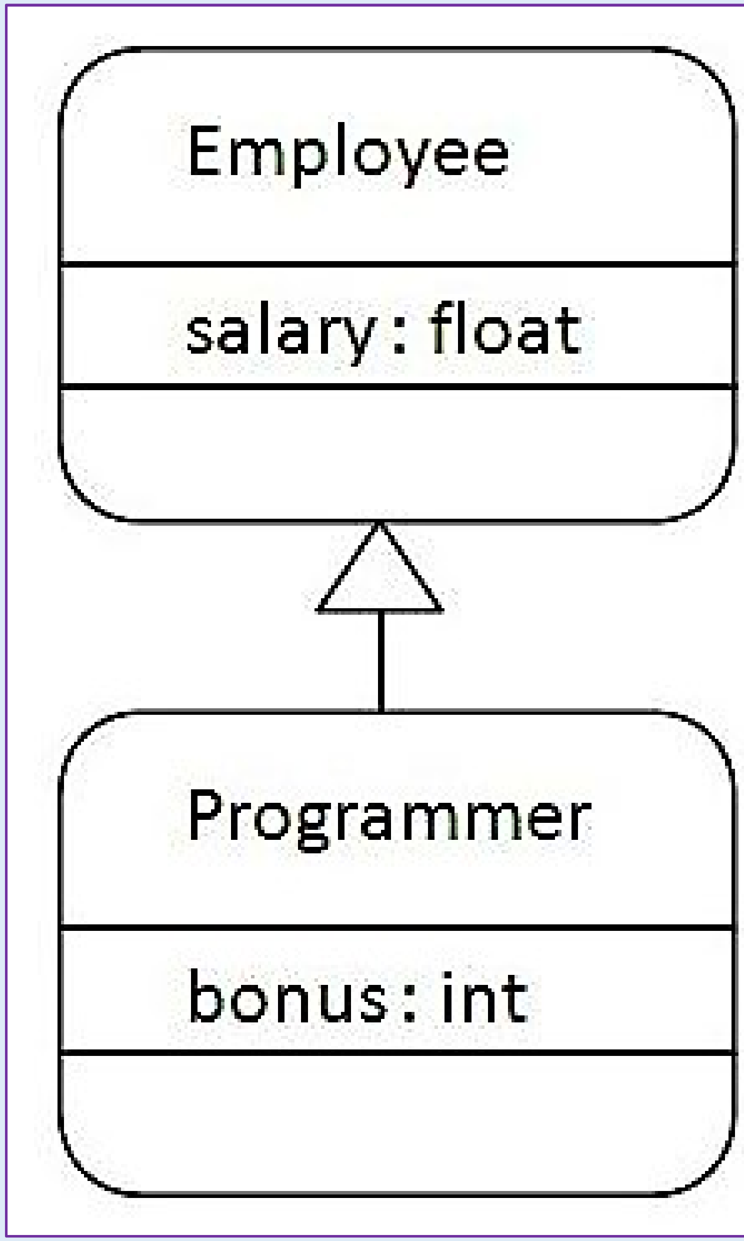
- The extends keyword indicates that you are making a new clas that derives from an existing class.
- The meaning of "extends" is to increase the functionality.
- In the terminology of Java, a class which is inherited is called parent or superclass, and the new class is called child or subclass.

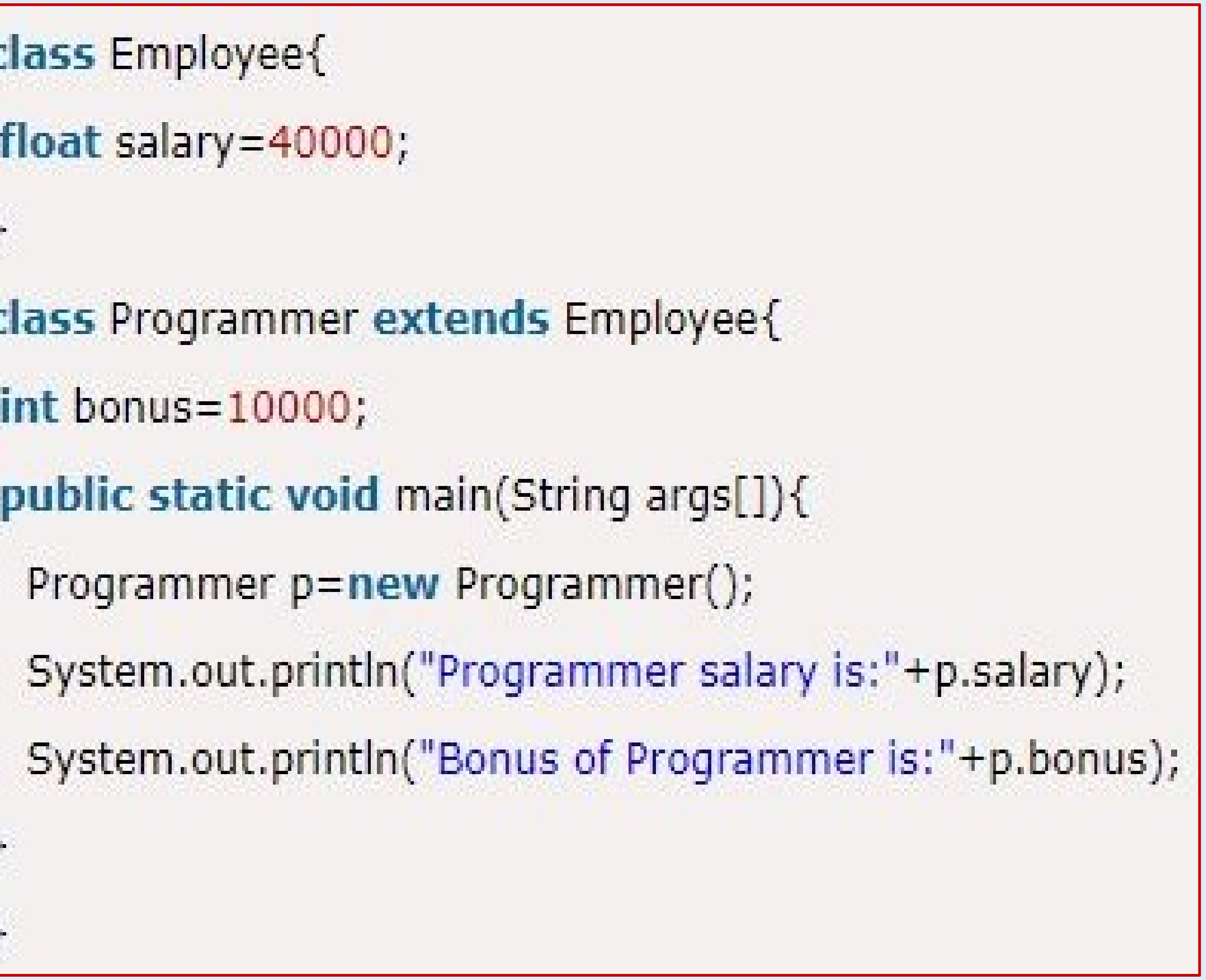
Programmer is the subclass (child class)

Employee is the superclass (Parent class)

The relationship between the two classes is Programmer IS-A Employee

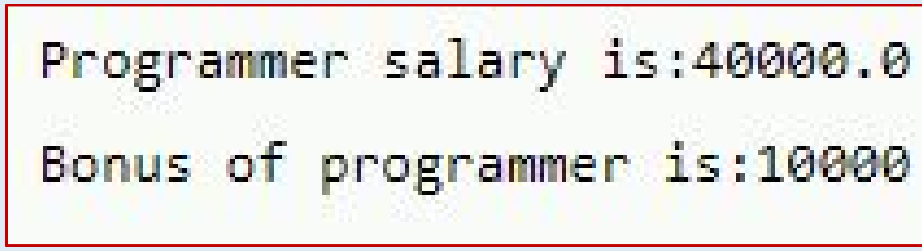
It means that Programmer is a type of Employee.





```
class Employee{
float salary=40000;
}
class Programmer extends Employee{
int bonus=10000;
public static void main(String args[]){
  Programmer p=new Programmer();
  System.out.println("Programmer salary is:"+p.salary);
  System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Output

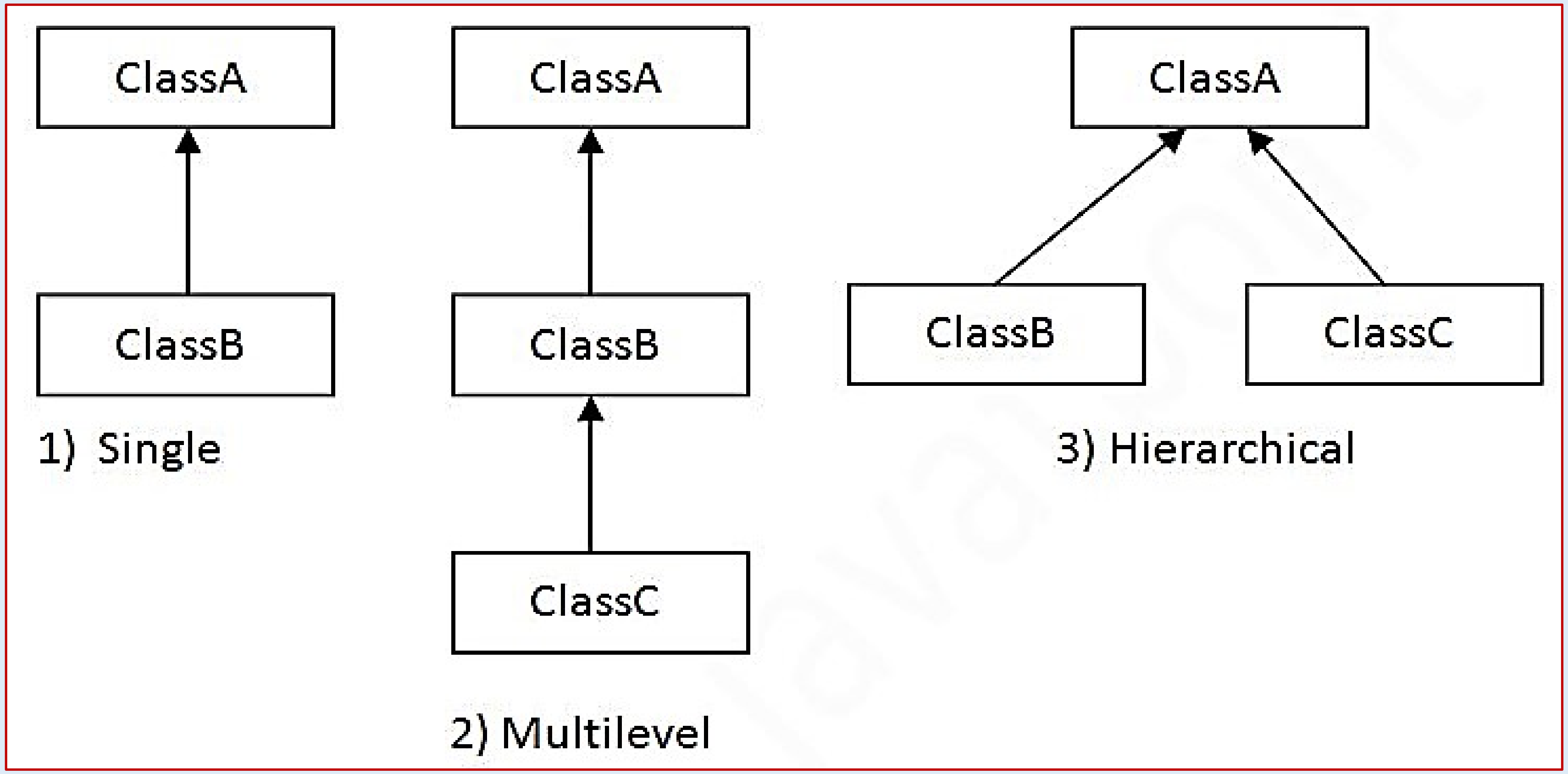


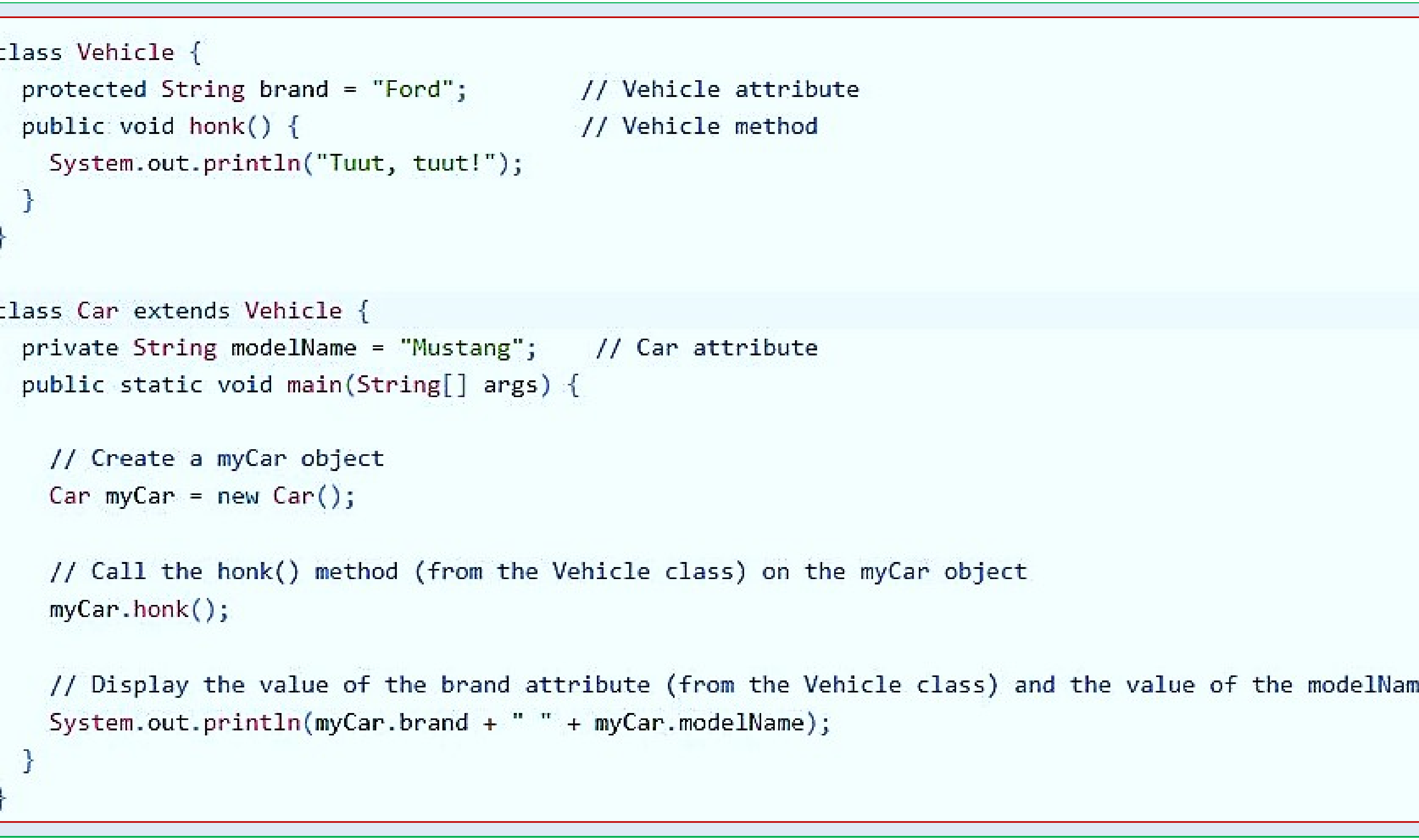
```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

Programmer object can access the attribute of its own class a
well as of Employee class i.e. code reusability.

Types of inheritance in java

- On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.





```
class Vehicle {
  protected String brand = "Ford";          // Vehicle attribute
  public void honk() {                      // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";     // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle class) and the value of the modelNam
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

SUPER KEYWORD

The super keyword in Java is a reference variable which is used t
refer immediate parent class object.

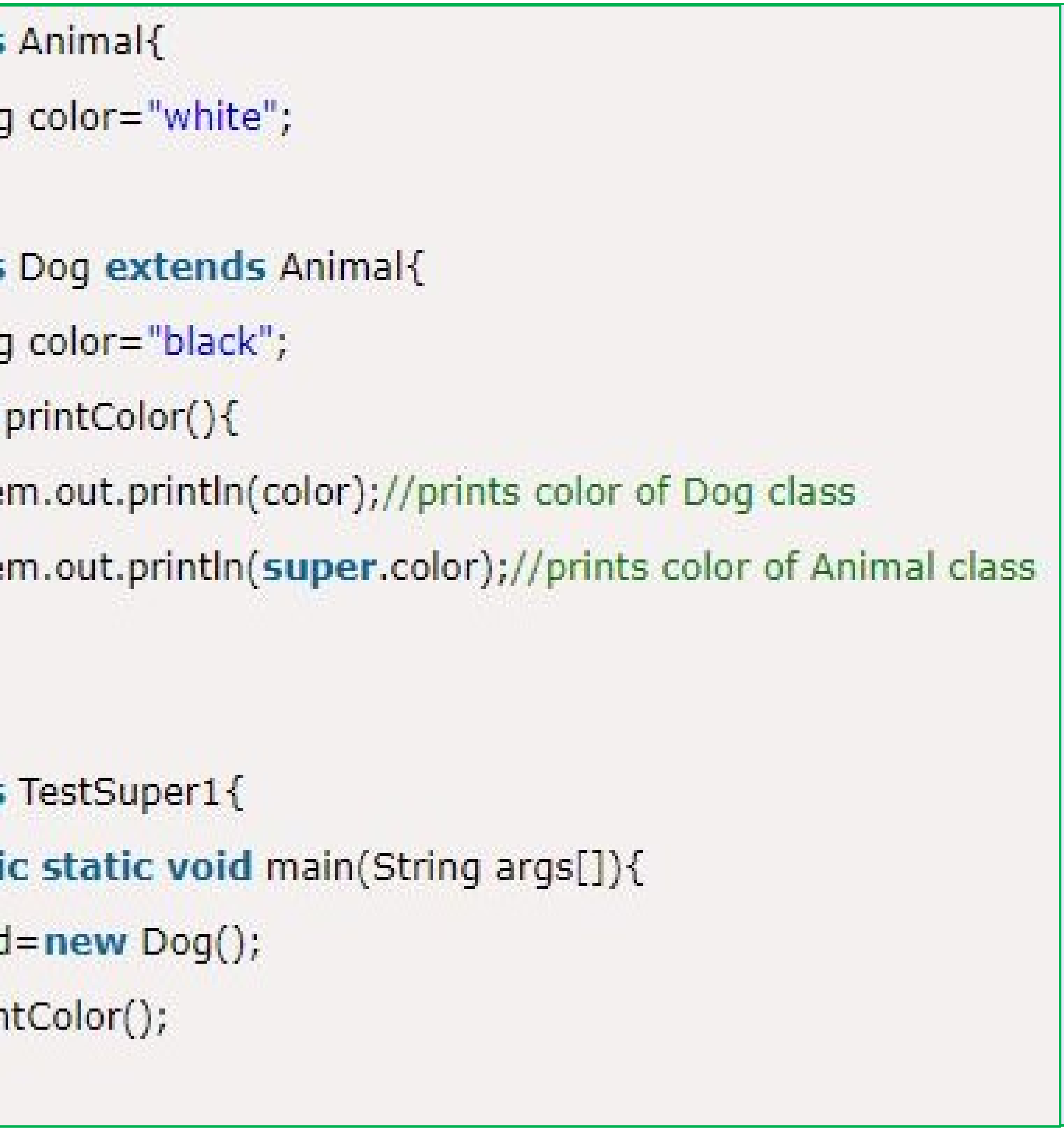
Usage of Java super Keyword

super can be used to refer immediate parent class instan
variable.

super can be used to invoke immediate parent class method.

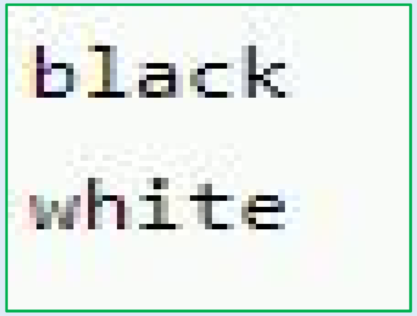
super() can be used to invoke immediate parent class constructor

We can use super keyword to access the data member (attribut
of parent class. It is used if parent class and child class have sam
attribute.



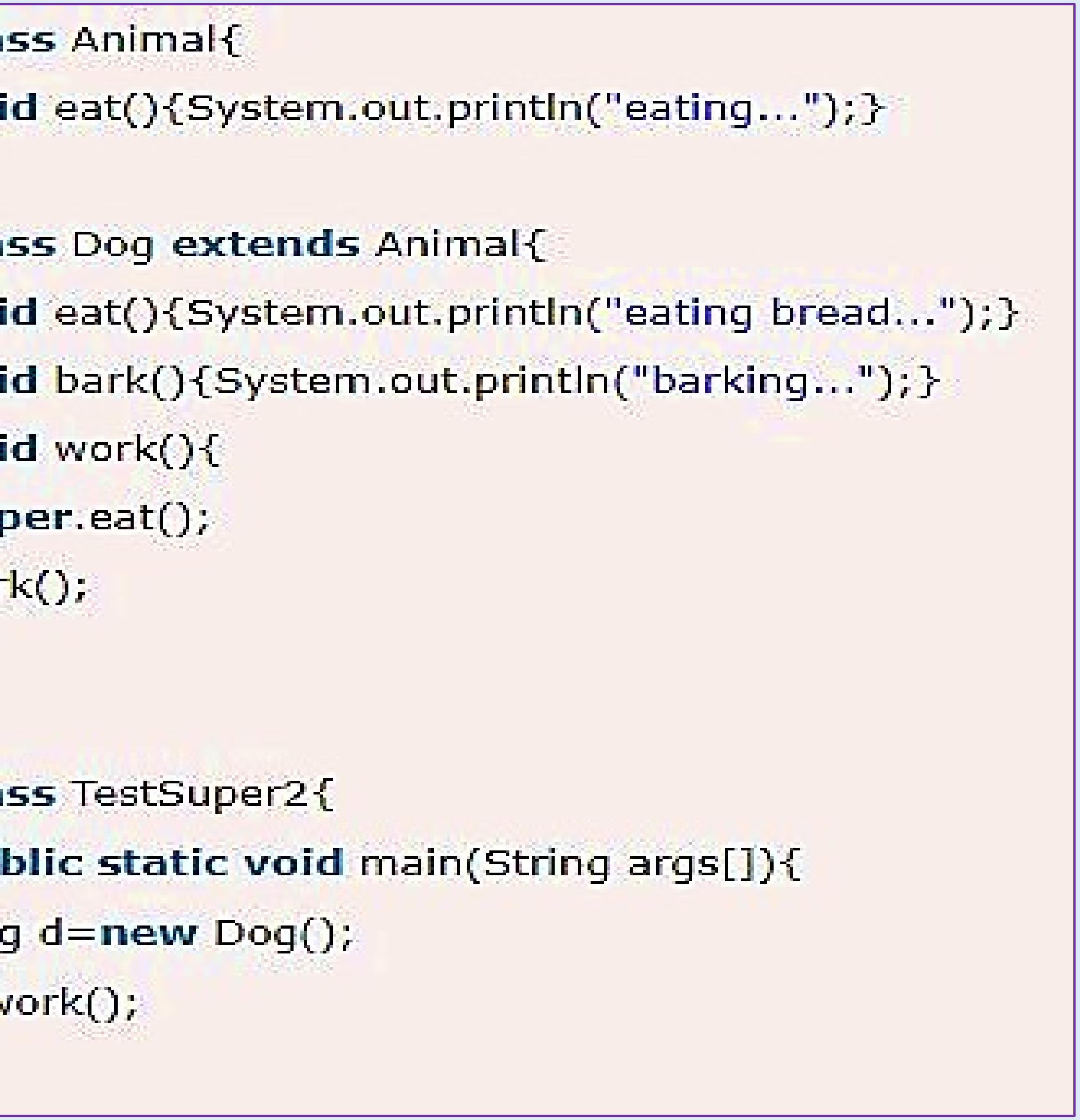
```
s Animal{
g color="white";

s Dog extends Animal{
g color="black";
printColor(){
em.out.println(color);//prints color of Dog class
em.out.println(super.color);//prints color of Animal class



s TestSuper1{
ic static void main(String args[]){
d=new Dog();
ntColor();
```

output



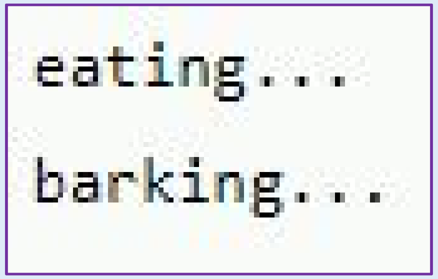
```
black
white
```

Animal and Dog both classes hav common property color. If we pri color property, it will print the co of current class by default. To acc the parent property, we need to super keyword.



```
ss Animal{
id eat(){System.out.println("eating...");}

ss Dog extends Animal{
id eat(){System.out.println("eating bread...");}
id bark(){System.out.println("barking...");}
id work(){
per.eat();
k();



ss TestSuper2{
blic static void main(String args[]){
g d=new Dog();
vork();
```

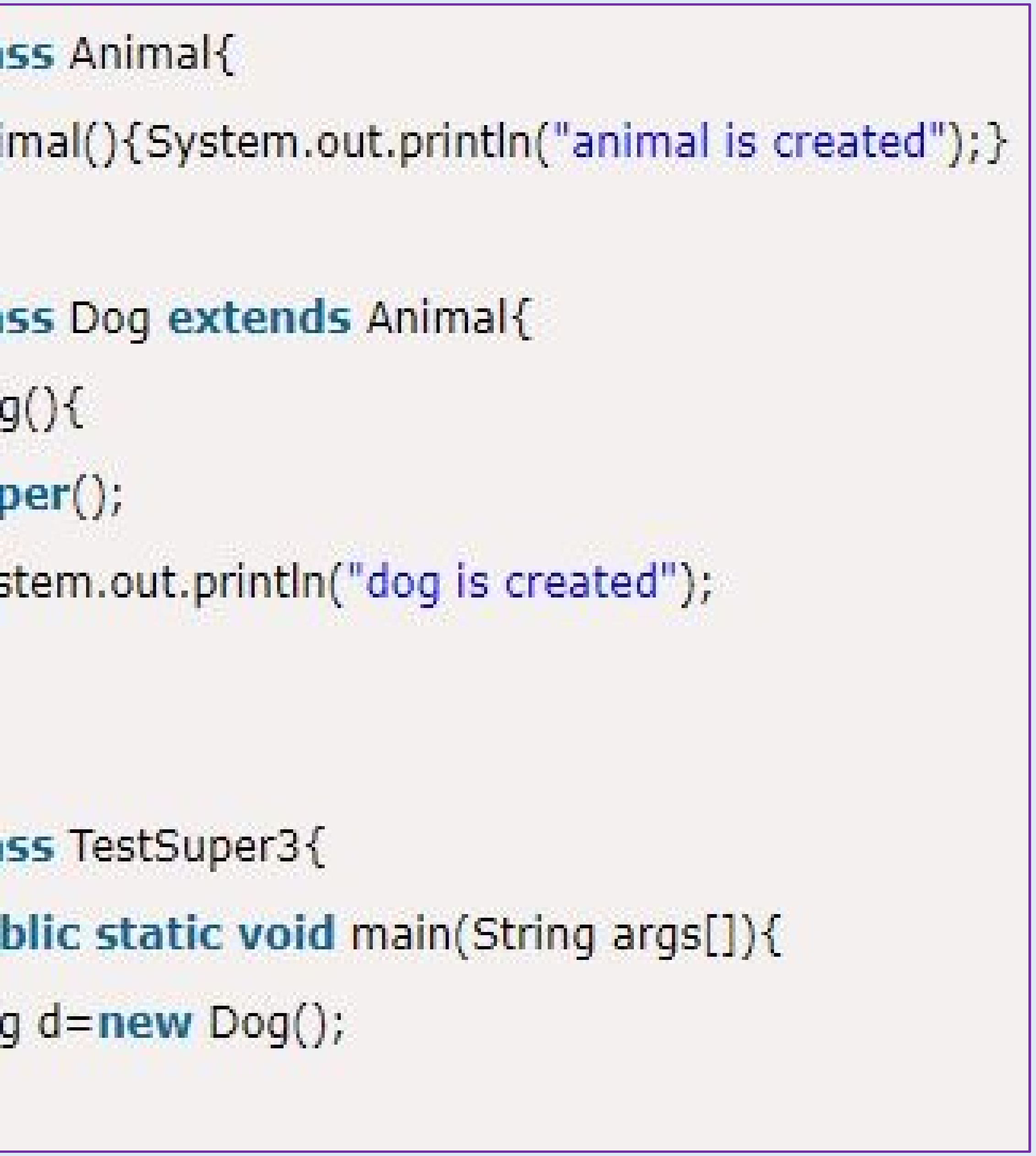
Output



```
eating...
barking...
```

The super keyword can also b
used to invoke(call) parent cla
method.

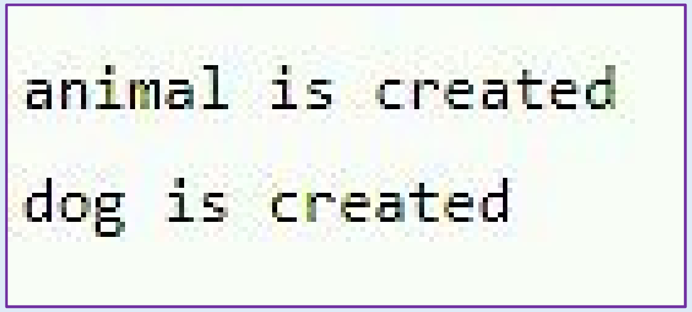
- In the above example Animal and Dog both classes have eat method
- If we call eat() method from Dog class, it will call the eat() metho of Dog class by default because priority is given to local.
- To call the parent class method, we need to use super keyword.



```
ss Animal{
imal(){System.out.println("animal is created");}

ss Dog extends Animal{
g(){
per();
stem.out.println("dog is created");



ss TestSuper3{
blic static void main(String args[]){
g d=new Dog();
```

The super keyword can also be used to invoke the parer class constructor.

Output



```
animal is created
dog is created
```

Calling order of constructors in inheritance

Order of execution of constructors in inheritance relationship
rom base (parent) class to derived (child)class.

We know that when we create an object of a class then t
onstructors get called automatically.

In inheritance relationship, when we create an object of a ch
lass, then first base class constructor and then derived cla
onstructor get called implicitly.

In simple word, we can say that the parent class constructor g
alled first, then of the child class constructor.

```
class A {
  A() {
          System.out.println("Inside A's constructor.");
        }
 }
// Create a subclass by extending class A.
class B extends A {
  B() {
          System.out.println("Inside B's constructor.");
       }
}
// Create another subclass by extending B.
class C extends B {
  C() {
          System.out.println("Inside C's constructor.");
       }
}
public class Main{
    public static void main(String args[])
       {
         C c = new C();
       }
}
```

Output

```
Inside A's constructor.
Inside B's constructor.
Inside C's constructor.
```

METHOD OVERRIDING

If subclass (child class) has the same method as declared in tl
parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementatic
of the method that has been declared by one of its parent class,
s known as method overriding

Jsage of Java Method Overriding

Method overriding is used to provide the specific implementatic
of a method which is already provided by its superclass.

Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

he method must have the same name as in the parent class

he method must have the same parameter as in the parent class

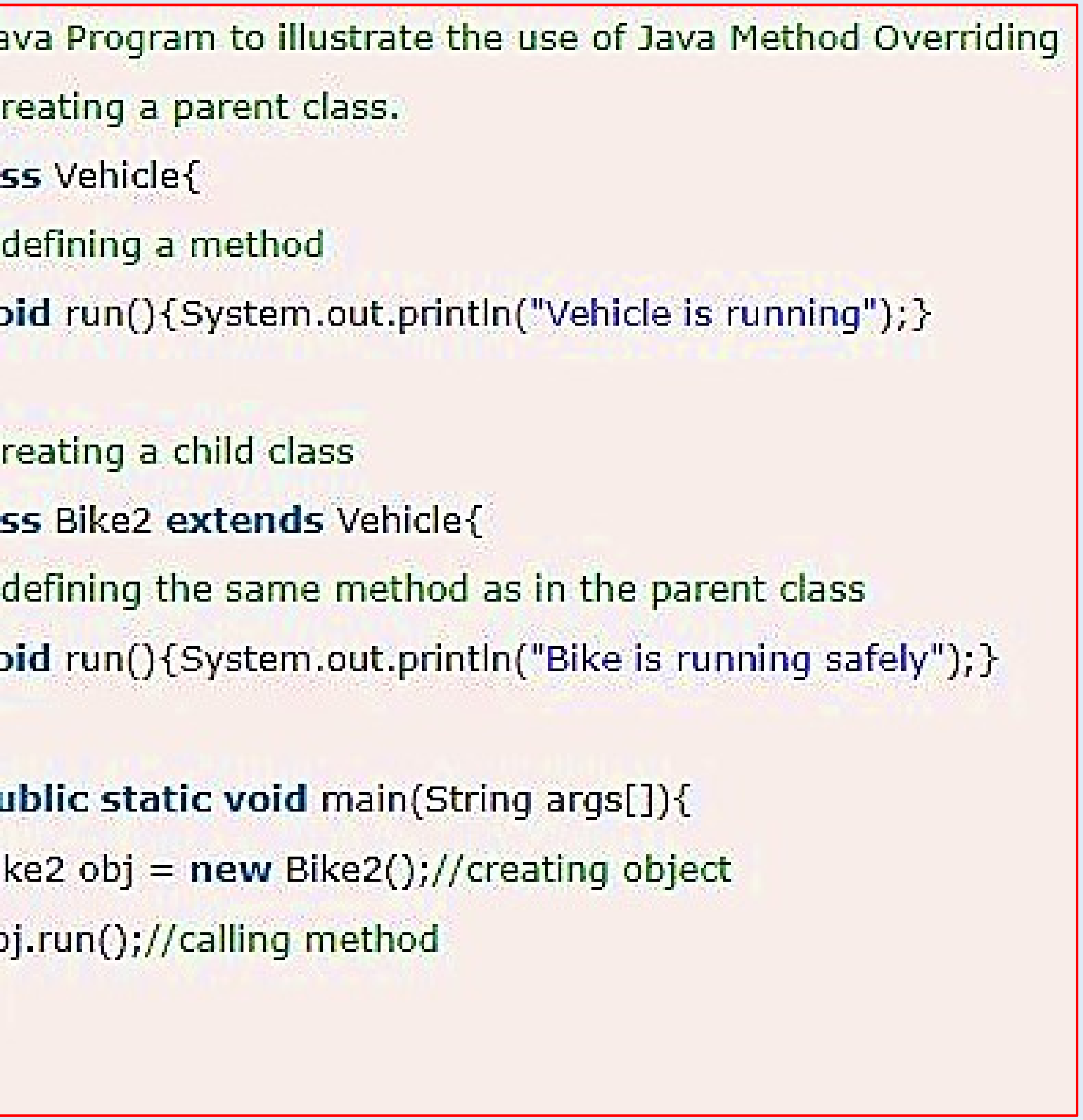
here must be an IS-A relationship (inheritance).

emember.......

 static method cannot be overridden. It is because the stat method is bound with class whereas instance method is bour with an object. Static belongs to the class area, and an instanc elongs to the heap area.

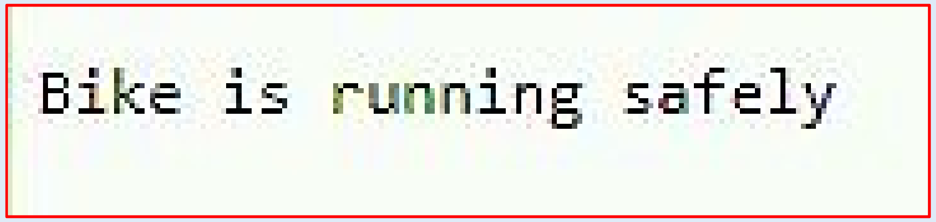
an we override java main method? - No, because the main is tatic method.

xample - method overriding



```
ava Program to illustrate the use of Java Method Overriding
reating a parent class.
ss Vehicle{
defining a method
oid run(){System.out.println("Vehicle is running");}

reating a child class
ss Bike2 extends Vehicle{
defining the same method as in the parent class
oid run(){System.out.println("Bike is running safely");}

ublic static void main(String args[]){
ke2 obj = new Bike2();//creating object
oj.run();//calling method
```

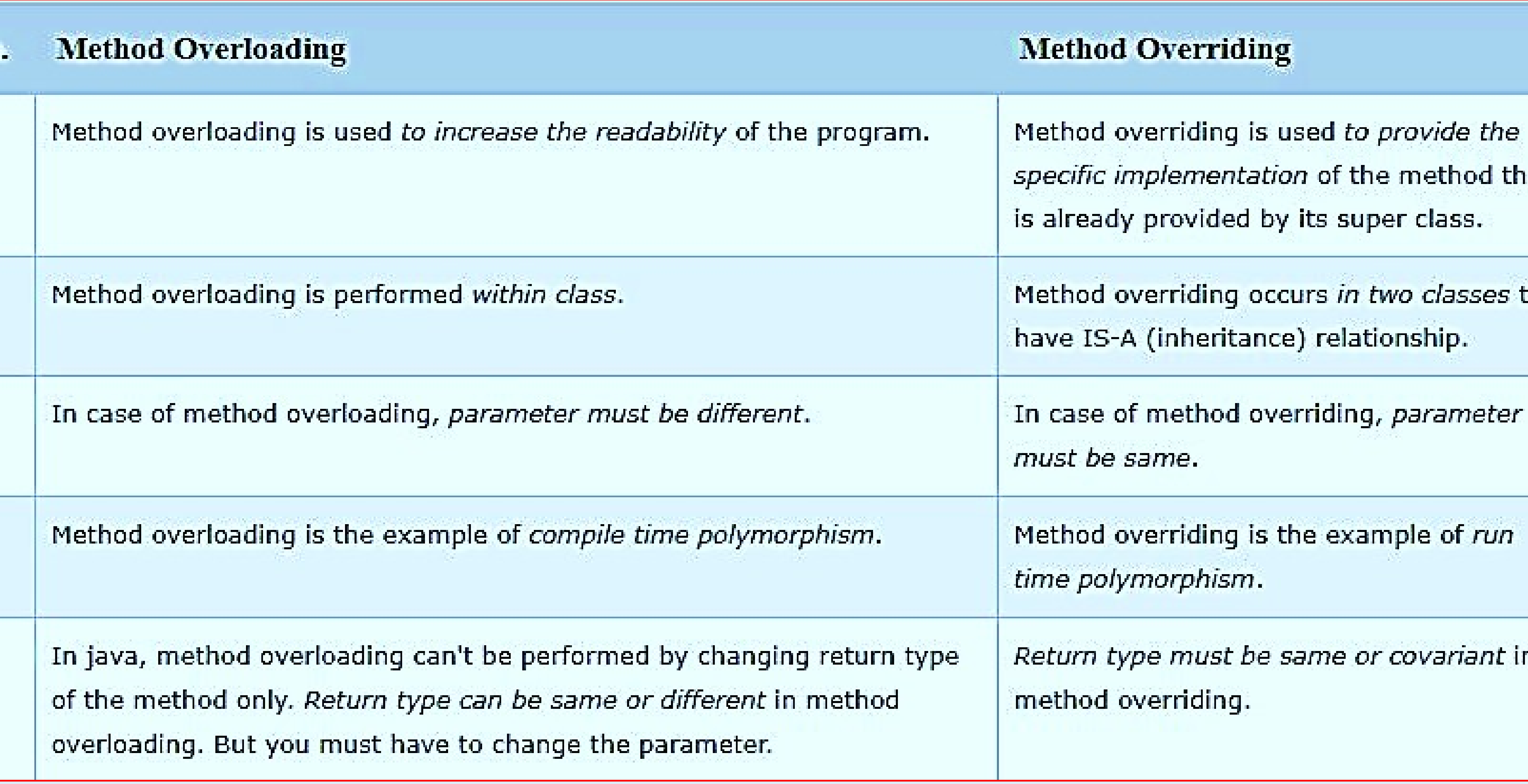
Output



```
Bike is running safely
```

we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the methoc are the same, and there is IS-/ relationship between the classes, so there is method overriding.

method overloading Vs. method overriding



.	Method Overloading	Method Overriding
	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method th is already provided by its super class.
	Method overloading is performed *within class*.	Method overriding occurs *in two classes* t have IS-A (inheritance) relationship.
	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* ir method overriding.

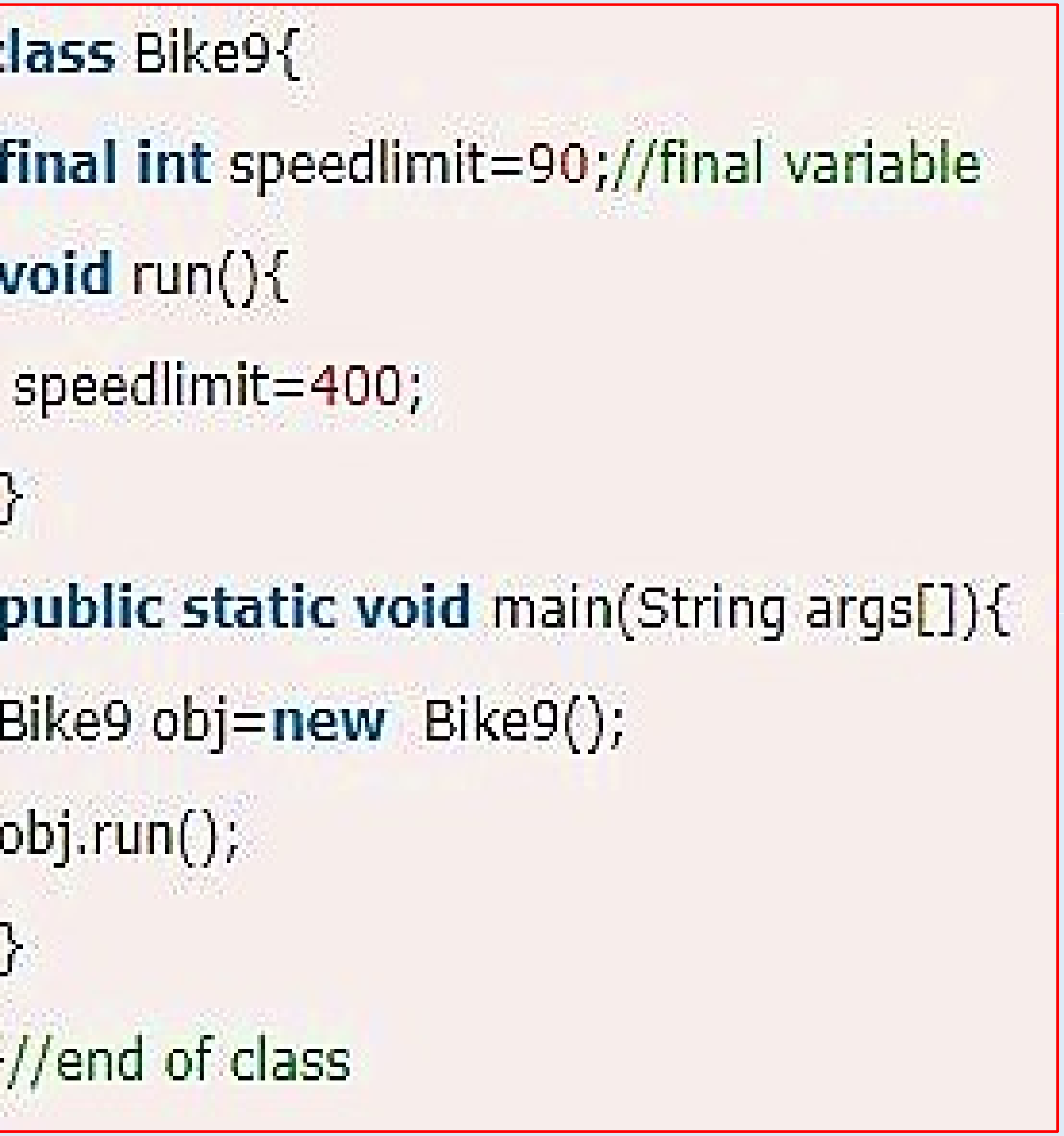
FINAL KEYWORD

The final keyword in java is used to restrict the user. The java final
eyword can be used in many context. Final can be:

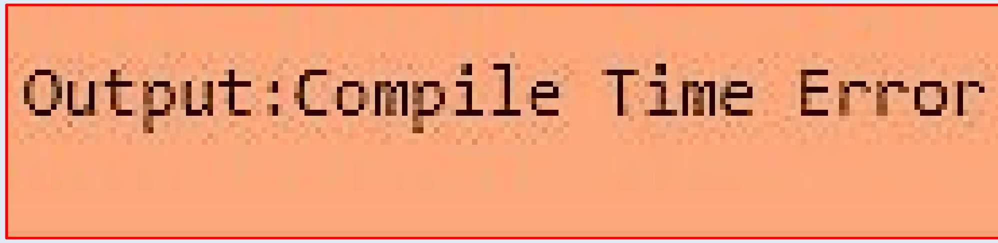
1. **variable**
2. **method**
3. **class**

Java final variable

f you make any variable as final, you cannot change the value
inal variable(It will be constant)



```
class Bike9{
final int speedlimit=90;//final variable
void run(){
 speedlimit=400;
}
public static void main(String args[]){
Bike9 obj=new  Bike9();
obj.run();
}
}//end of class
```

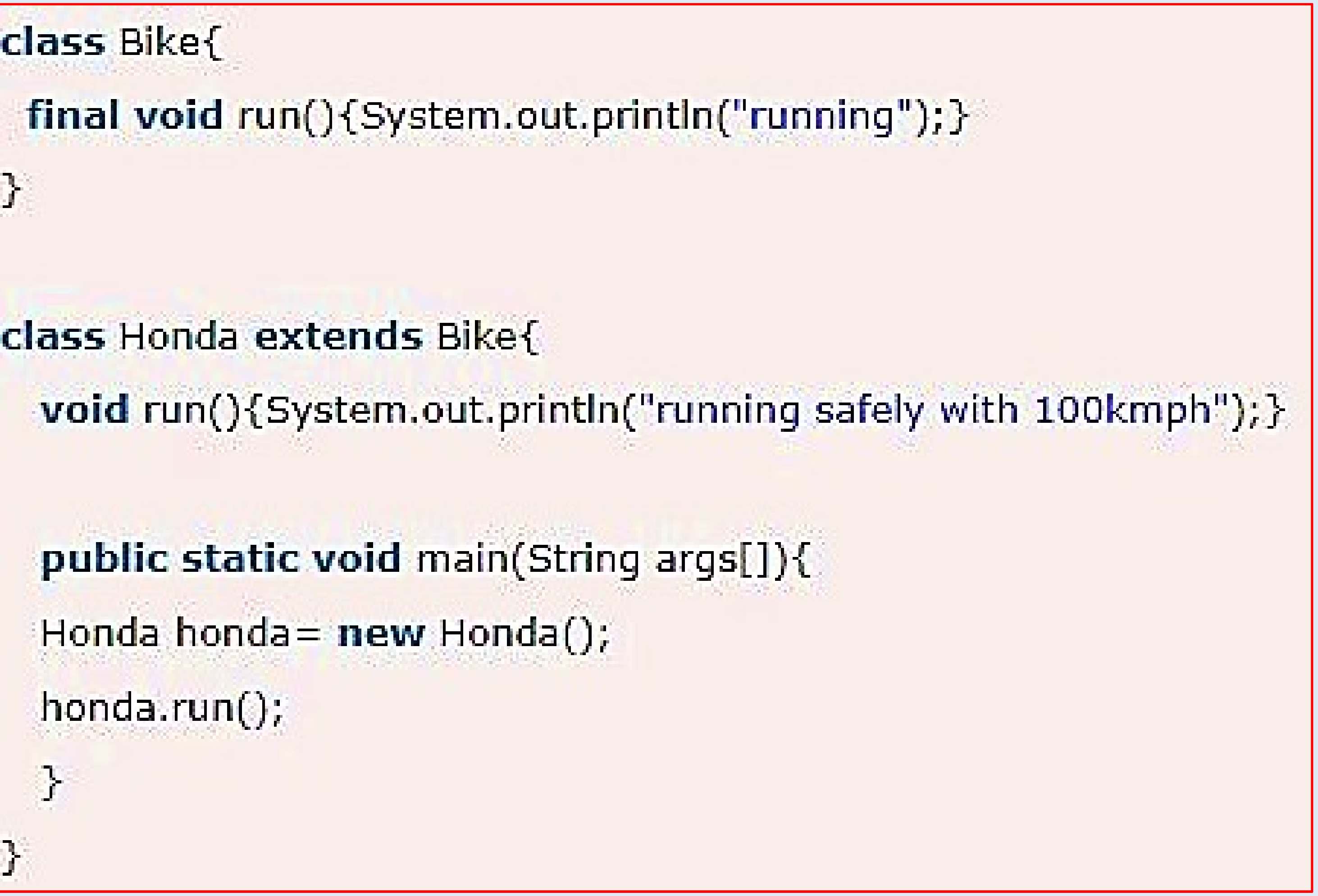


```
Output:Compile Time Error
```

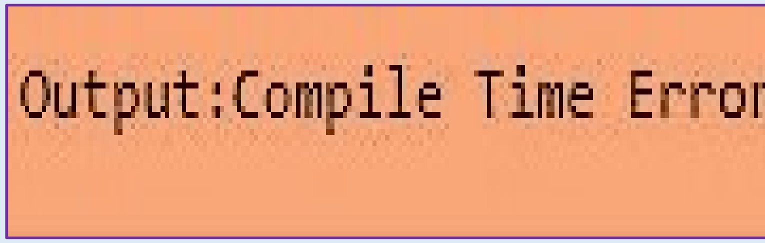
There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

Java final method

If we make any method as final, **we cannot override** it



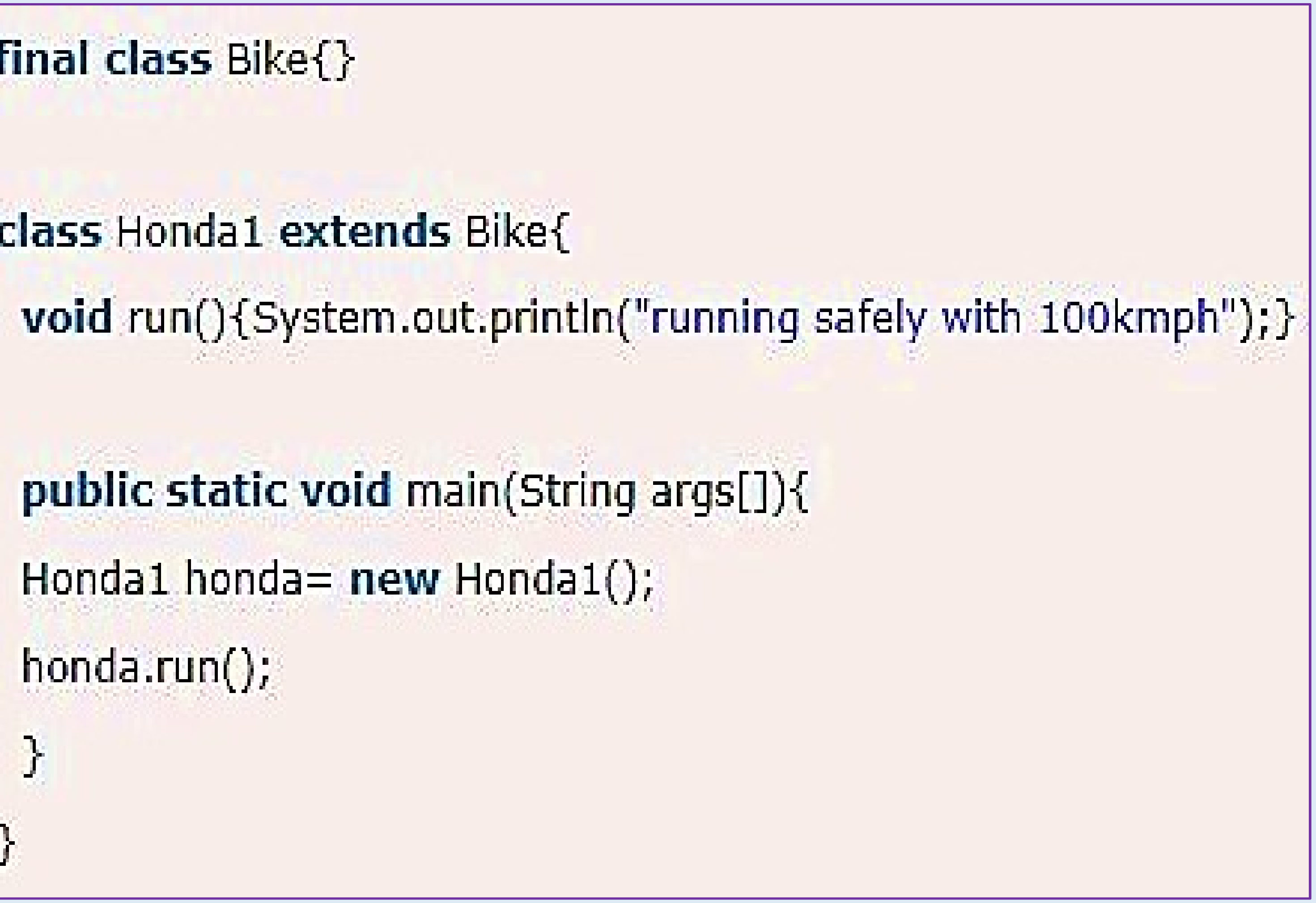
```
class Bike{
 final void run(){System.out.println("running");}
}

class Honda extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```



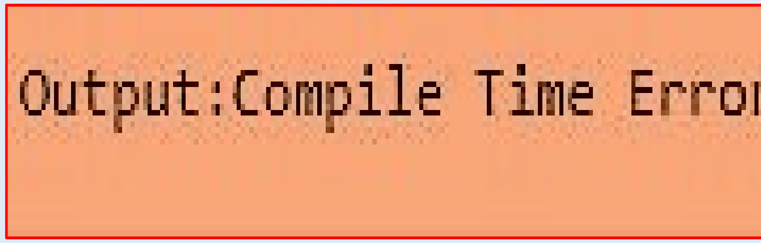
```
Output:Compile Time Erro
```

Java final class

If we make any class as final, we cannot extend it.



```
final class Bike{}

class Honda1 extends Bike{
 void run(){System.out.println("running safely with 100kmph");}

 public static void main(String args[]){
 Honda1 honda= new Honda1();
 honda.run();
 }
}
```

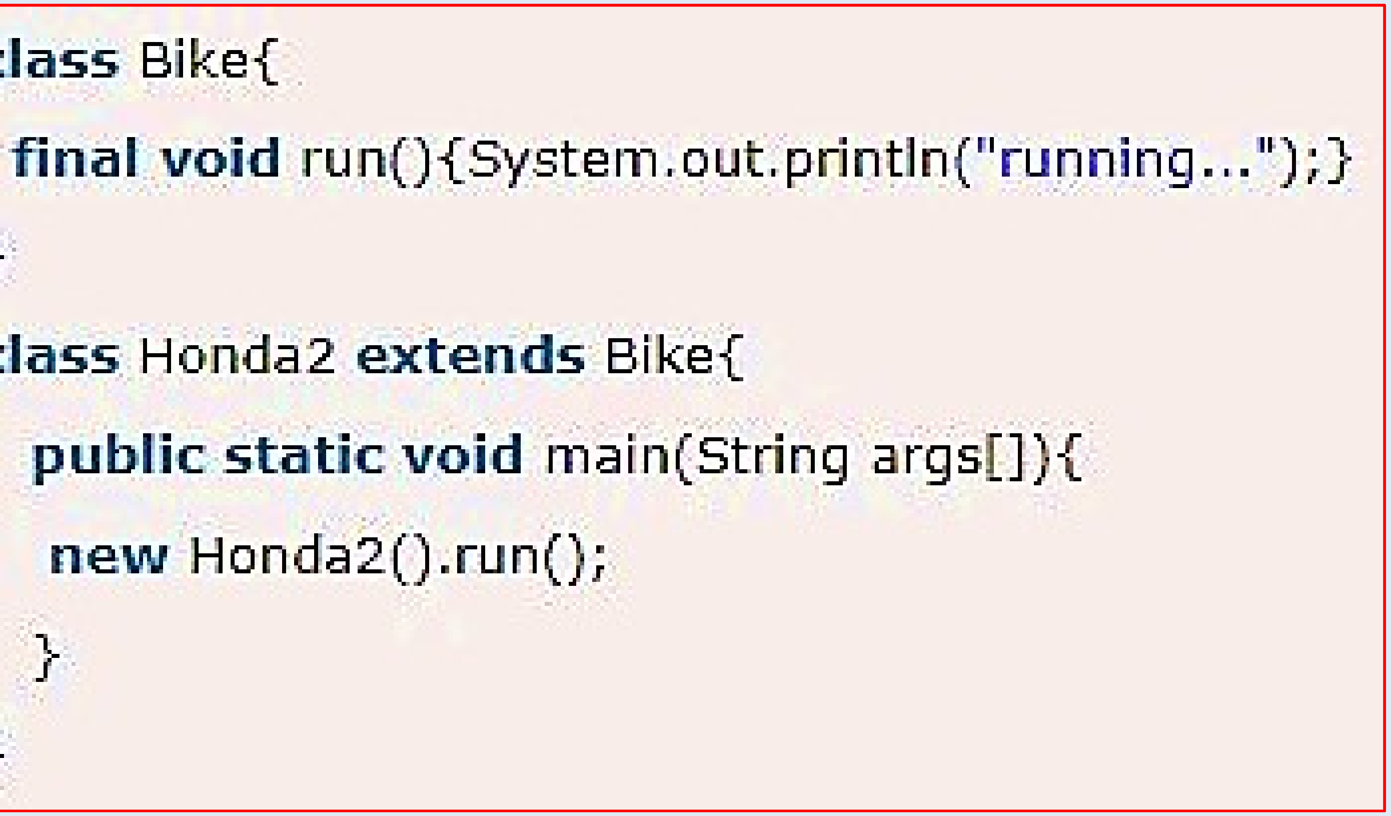


```
Output:Compile Time Erro
```

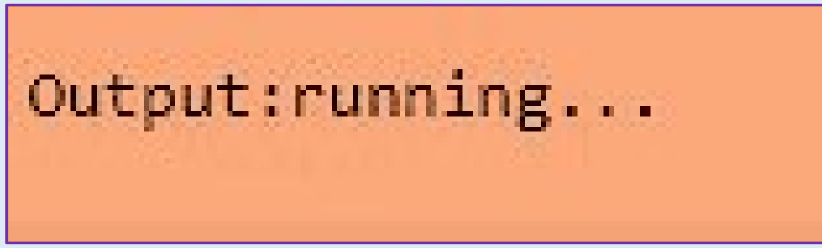
EDULINE

Is final method inherited?

Yes, final method is inherited but you cannot override it. For Example:



```
class Bike{
 final void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
  public static void main(String args[]){
   new Honda2().run();
  }
}
```



```
Output:running...
```

EDULINE

Points to Remember

A constructor cannot be declared as final.

Local final variable must be initializing during declaration.

We cannot change the value of a final variable.

A final method cannot be overridden.

A final class not be inherited.

If method parameters are declared final then the value of thes
arameters cannot be changed.

final, finally and finalize are three different terms. finally is used
xception handling and finalize is a method that is called by JV
uring garbage collection.

ABSTRACT CLASSES AND METHODS

Data abstraction is the process of hiding certain details an showing only essential information to the user.

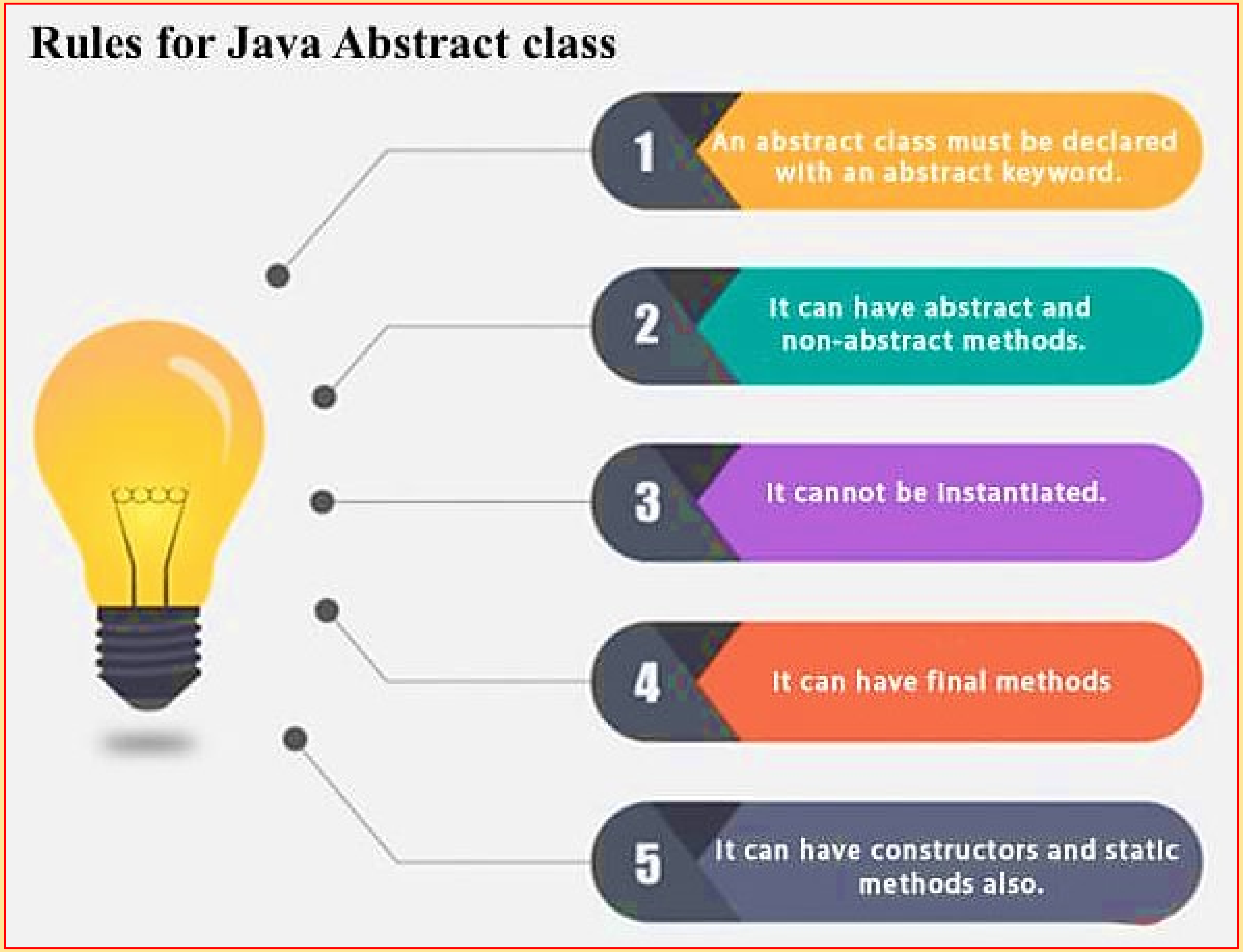
Abstract class: is a restricted class that cannot be used to creat objects (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and does not have a body. The body is provided by the subclas (inherited from).

An abstract class can have both abstract and regular methods:

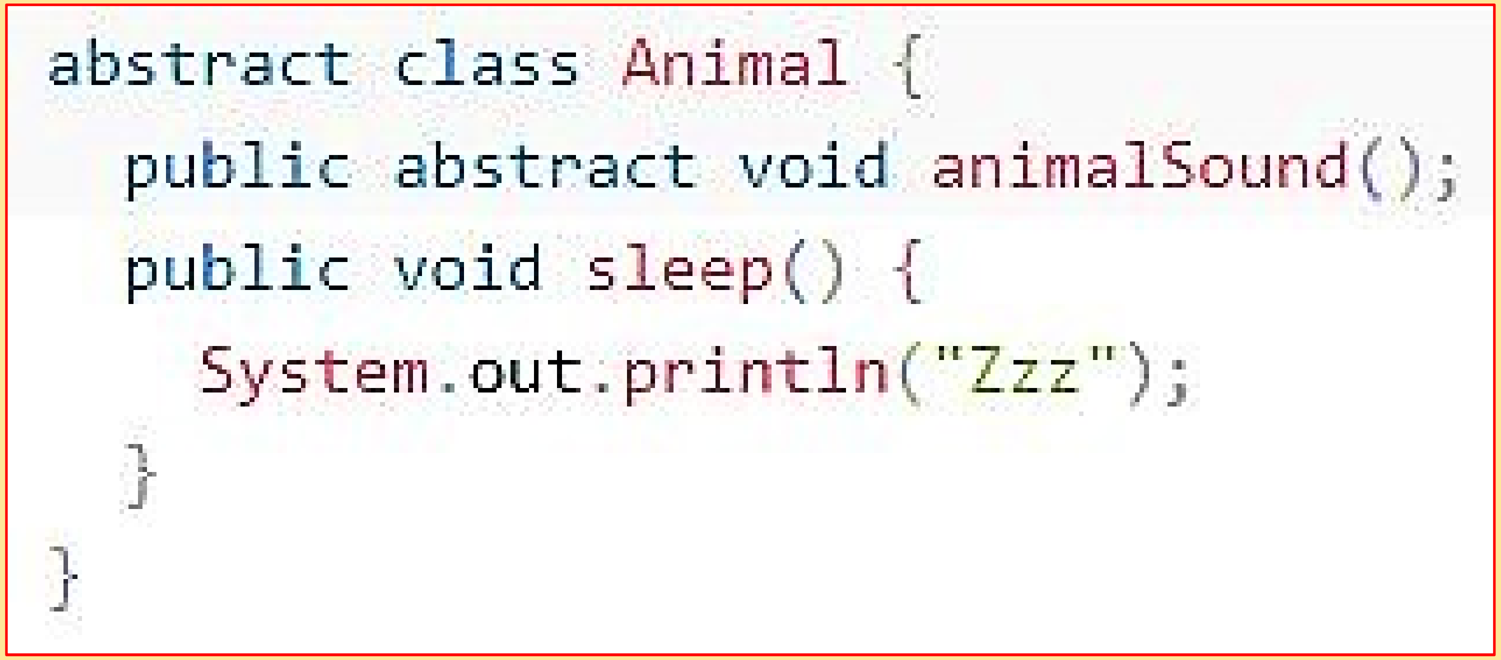
EDULINE

Abstract class



Rules for Java Abstract class

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.



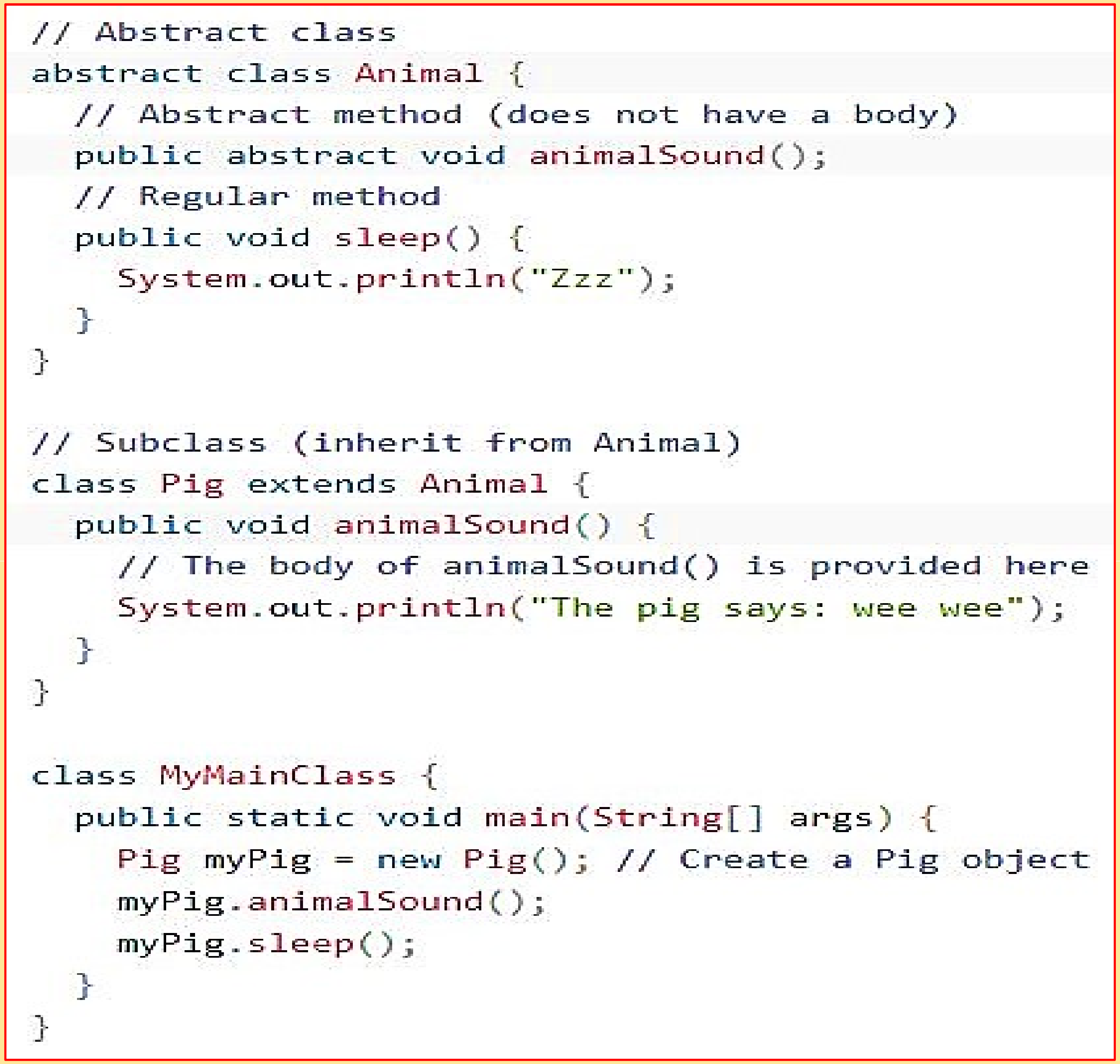
```java
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

From the example above, it is not possible to create an object
the Animal class

Animal myObj = new Animal();   // will generate an error

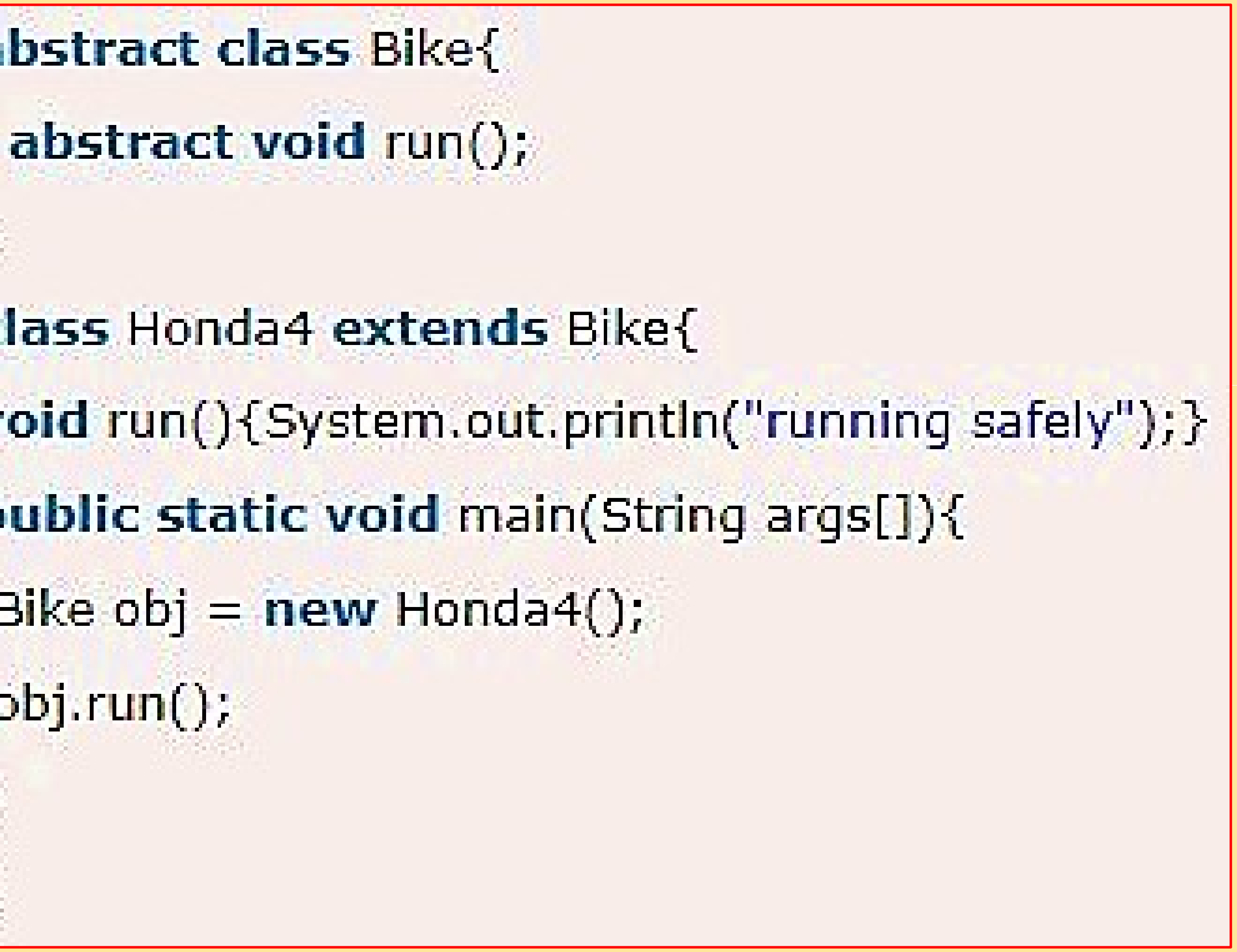
To access the abstract class, it must be inherited from anothe
class

xample



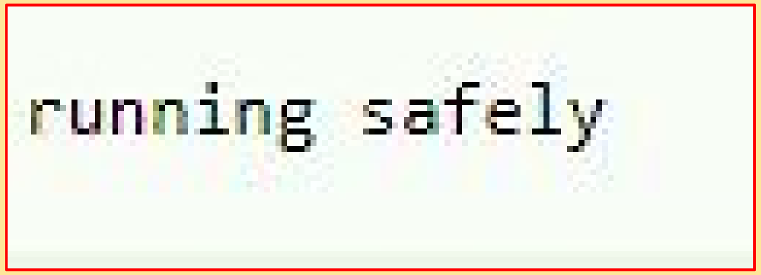
```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

xample - Here Bike is an abstract class that contains only on
bstract method run. Its implementation is provided by the Hond
ass.



```
bstract class Bike{
 abstract void run();

lass Honda4 extends Bike{
oid run(){System.out.println("running safely");}
ublic static void main(String args[]){
Bike obj = new Honda4();
obj.run();
```

Output



```
running safely
```

THE OBJECT CLASS

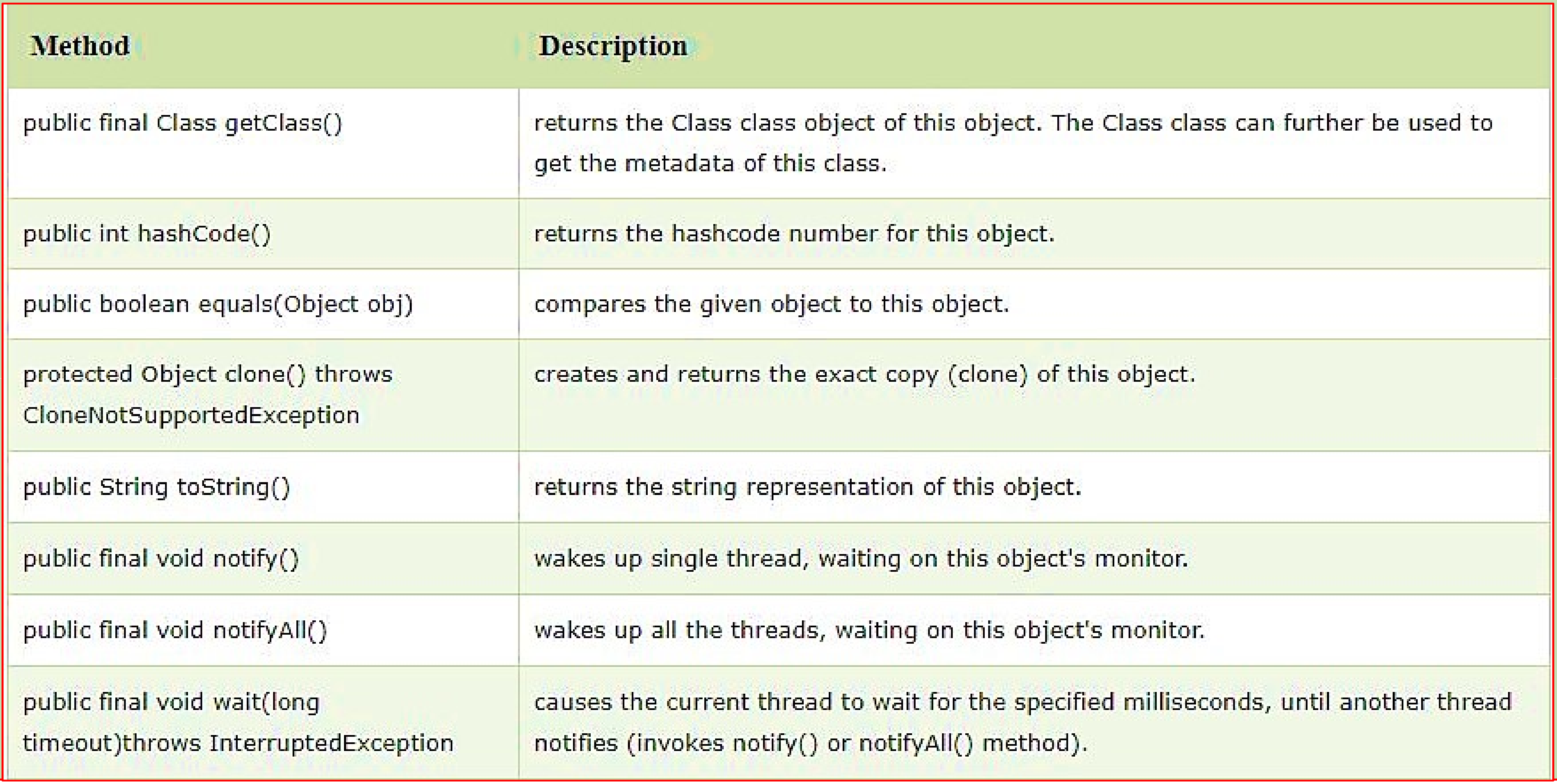
he Object class is the parent class of all the classes in java b
default. In other words, it is the topmost class of java.

he Object class provides some common behaviors to all th
bjects such as object can be compared, object can be clone
bject can be notified etc.

bject class is present in java.lang package

very class in Java is directly or indirectly derived from the Obje
lass

Methods of Object class



Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).