

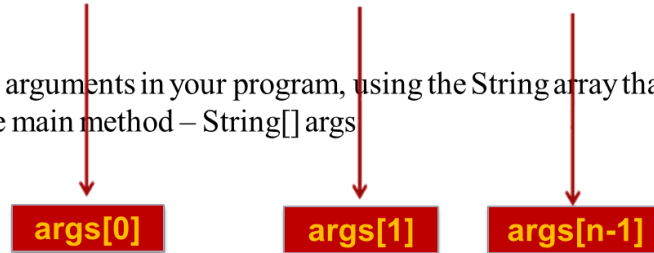
Java Day2

Accessing command line arguments

When you execute a java program, you can pass command line arguments in the following way :

```
java Simple <argument1> <argument2>...<argument-n>
```

You can access these arguments in your program, using the String array that you have passed as an argument to the main method – String[] args



Passing command line arguments

```
class Simple {  
    static public void main(String[] args) {  
        System.out.println(args[0]);  
    }  
}
```

When we compile the above code successfully and execute it as Java Simple Java, the output will be

Java

Accessing numeric command line arguments

```
class Simple {  
    static public void main(String[] args) {  
        int i1 = Integer.parseInt(args[0]);  
        int i2 = Integer.parseInt(args[1]);  
        System.out.println(i1+i2);  
    }  
}
```

When we compile the above code successfully and execute it as Java Simple 10 20, the output will be

30

Finding length of an Array

- How to find the number of command line arguments that a user may pass while executing a java program?
- The answer to the above question lies in args.length, where args is the String array that we pass to the main method and length is the property of the Array Object

Example on Finding length of an Array

```
class FindNumberOfArguments {  
    public static void main(String[] args) {  
        int len = args.length;  
        System.out.println(len);  
    }  
}
```

If you compile the above code successfully and execute it as java FindLength A B C D E F, the result will be

6

And if you execute it as java FindLength Tom John Lee, the result will be

3

Good Programming Practices

Naming Conventions

Class Names

- Class names should be **nouns**, in mixed case with the first letter of each internal word capitalized
- Class names should be simple and descriptive
- Eg: class **Student**, class **TestStudent**

Variable Names

- The variables are in mixed case with a lowercase first letter
- Variable names should not start with underscore _ or dollar sign \$ characters, even though both are allowed
- Variable names should be small yet meaningful
- One-character variable names should be avoided except for temporary “throwaway” variables
- Eg: int y, myWidth;

Good Programming Practices(Contd.)

Naming Conventions

Method Names

- Methods should be **verbs**, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized
- Eg: void run(), void getColor()

Comments

Block Comments

- Block comments are used to provide descriptions of files, methods, data structures and algorithms
- Block comments may be used at the beginning of each file and before each method

/*

Here is a block comment

*/

23

Good Programming Practices(Contd.).

Comments

Single line Comment

- Single line comments can be written using // Single line

Number per Line

- One declaration per line is recommended

int height;

int width;

is preferred over

int height,width;

Do not put different types on the same line

int height,width[]; // Not recommended

The Java Buzzwords

- Simple
- Object-Oriented
 - Supports encapsulation, inheritance, abstraction, and polymorphism
- Distributed
 - Libraries for network programming
 - Remote Method Invocation
- Architecture neutral
 - Java Bytecodes are interpreted by the JVM

The Java Buzzwords (Contd.).

- Secure
 - Difficult to break Java security mechanisms
 - Java Bytecode verification
 - Signed Applets
- Portable
 - Primitive data type sizes and their arithmetic behavior specified by the language
 - Libraries define portable interfaces
- Multithreaded
 - Threads are easy to create and use

Java Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Primitive Data Types

Data Type	Size (in bits)	Minimum Range	Maximum Range	Default Value (for fields)
byte	8	-128	+127	0
short	16	-32768	+32767	0
int	32	-2147483648	+2147483647	0
long	64	-9223372036854775808	+9223372036854775807	0L
float	32	1.40E-45	3.40282346638528860e+38	0.0f
double	64	4.94065645841246544e-324d	1.79769313486231570e+308d	0.0d
char	16		0 to 65,535	'\u0000'
boolean	1	NA	NA	false

Types of Variables

The Java programming language defines the following kinds of Variables:

- Local Variables
 - Tied to a method
 - Scope of a local variable is within the method
- Instance Variables (Non-static)
 - Tied to an object
 - Scope of an instance variable is the whole class
- Static Variables
 - Tied to a class
 - Shared by all instances of a class

Operators

- Java provides a set of operators to manipulate operations.
- Types of operators in java are,
 - Arithmetic Operators
 - Unary Operator
 - Relational Operators
 - Logical Operators
 - Simple Assignment Operator
 - Bitwise Operators

Arithmetic Operators

The following table lists the arithmetic operators

Operator	Description	Example
+	Addition	A + B
-	Subtraction	A - B
*	Multiplication	A * B
/	Division	A/B
%	Modulus	A%B

Arithmetic Operators - Example

/* Example to understand Arithmetic operator */

```
class Sample{
    public static void main(String[ ] args){
        int a = 10;
        int b = 3;
        System.out.println("a + b = " + (a + b) );
        System.out.println("a - b = " + (a - b) );
        System.out.println("a * b = " + (a * b) );
        System.out.println("a / b = " + (a / b) );
        System.out.println("a % b = " + (a % b) );
    }
}
```

Output:

```
a + b = 13
a - b = 7
a * b = 30
a / b = 3
a % b = 1
```

Unary Operators

The following table lists the unary operators

Operator	Description	Example
+	Unary plus operator	+A
-	Unary minus operator	-A
++	Increment operator	++A or A++
--	Decrement operator	--A or A--

Unary Operator - Example

/* Example for Unary Operators */

```
class Sample{
    public static void main(String args[]) {
        int a = 10;
        int b = 20;

        System.out.println("++a    = " + (++a) );
        System.out.println("--b    = " + (--b) );
    }
}
```

Output:

```
++a = 11
--b = 19
```

Relational Operators

The following table lists the relational operators

Operator	Description	Example
==	Two values are checked, and if equal, then the condition becomes true	(A == B)
!=	Two values are checked to determine whether they are equal or not, and if not equal, then the condition becomes true	(A != B)
>	Two values are checked and if the value on the left is greater than the value on the right, then the condition becomes true.	(A > B)
<	Two values are checked and if the value on the left is less than the value on the right, then the condition becomes true	(A < B)
>=	Two values are checked and if the value on the left is greater than equal to the value on the right, then the condition becomes true	(A >= B)
<=	Two values are checked and if the value on the left is less than equal to the value on the right, then the condition becomes true	(A <= B)

Relational Operators - Example

/* Example to understand Relational operator */

```
class Sample{
    public static void main(String[] args){
        int a = 10;
        int b = 20;
        System.out.println("a == b = " + (a == b) );
        System.out.println("a != b = " + (a != b) );
        System.out.println("a > b = " + (a > b) );
        System.out.println("a < b = " + (a < b) );
        System.out.println("b >= a = " + (b >= a) );
        System.out.println("b <= a = " + (b <= a) );
    }
}
```

Output:

```
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

48

Logical Operators

The following table lists the logical operators

Operator	Description	Example
&&	This is known as Logical AND & it combines two variables or expressions and if and only if both the operands are true, then it will return true	(A && B) is false
	This is known as Logical OR & it combines two variables or expressions and if either one is true or both the operands are true, then it will return true	(A B) is true
!	Called Logical NOT Operator. It reverses the value of a Boolean expression	!(A && B) is true

Logical Operators - Example

/* Example to understand logical operator */

```
class Sample{
    public static void main(String[] args){
        boolean a = true;
        boolean b = false;
        System.out.println("a && b = " + (a&&b) );
        System.out.println("a || b = " + (a||b) );
        System.out.println("!(a && b) = " + !(a && b) );
    }
}
```

Output:

```
a && b =
false
a || b = true
!(a && b) =
true
```

Simple Assignment Operator

= Simple assignment operator

Which assigns right hand side value to left hand side variable

Ex:

```
int a;  
a = 10;
```

Shift Operators << and >>

- The shift operators(<< and >>) shift the bits of a number to the left or right, resulting in a new number.
- They are used only on integral numbers(and not on floating point numbers, i.e. decimals).
- The right shift operator(>>) is used to divide a number in the multiples of 2, while the left shift operator(<<) is used to multiply a number in the multiples of 2.

Right Shift Operator >>

- Let us understand the use of right shift operator with the following example :

```
int x = 16;  
x = x >> 3;
```

- When we apply the right shift operator >>, the value gets divided by 2 to the power of number specified after the operator. In this case, we have 3 as the value after the right shift operator. So, 16 will be divided by the value 2 to the power of 3, which is 8.
- The result is 2.

Right Shift Operator >> (Contd.).

- When we represent 16 in binary form, we will get the following binary value :

[illegible]

- When we apply >> which is the right shift operator, the bit represented by 1 moves by 3 positions to the right(represented by the number after the right shift operator).
- After shifting the binary digit 1, we will get :

[illegible]

- $x=2$

Right Shift Operator >> Demo

```
class ShiftExample1 {
    public static void main(String[] args) {
        int x = 16;
        System.out.println("The original value of x is "+x);
        x = x >> 3;
        System.out.println("After using >> 3, the new value is "+x);
    }
}
```

The original value of x is 16
After using >> 3, the new
value is 2

Left Shift Operator <<

- Let us understand the use of left shift operator with the following example :

```
int x = 8;
x = x << 4;
```

- When we apply the left shift operator \ll , the value gets multiplied by 2 to the power of number specified after the operator. In this case, we have 4 as the value after the left shift operator. So, 8 will be multiplied by the value 2 to the power of 4, which is 16.
- The result is 128.

Left Shift Operator << (Contd.).

- When we represent 8 in binary form, we will get the following binary value :

0 1 0 0 0

- When we apply << which is the left shift operator, the bit represented by 1 moves by 4 positions to the left (represented by the number after the right shift operator).
- After shifting the binary digit 1, we will get :

0 1 0 0 0 0 0 0

↑↑↑↑↑

- X=128

Left Shift Operator << Demo

```
class ShiftExample2 {  
    public static void main(String[] args) {  
        int x =8;  
        System.out.println("The original value of x is "+x);  
        x = x << 4;  
        System.out.println("After using << 4, the new value is "+x);  
    }  
}
```

**The original value of x is 8
After using << 4, the new value
is 128**

Bitwise Operators

The bitwise operators take two bit numbers, use OR/AND to determine the result on a bit by bit basis.

The 3 bitwise operators are :

- & (which is the bitwise AND)
- | (which is the bitwise inclusive OR)
- ^ (which is the bitwise exclusive OR)

Bitwise & (AND) operator

The & operator compares corresponding bits between two numbers and if both the bits are 1, only then the resultant bit is 1. If either one of the bits is 0, then the resultant bit is 0.

Example :

```
int x = 7;
int y = 9;
```

What will be x & y ?

```
7 -> 0 1 1 1
9 -> 1 0 0 1
-----
      0 0 0 1
-----
x & y results in 1.
```

Bitwise & Operator Demo

```
class BitwiseExample1 {
    public static void main(String[] args) {
        int x = 7;
        int y = 9;
        int z = x & y;
        System.out.println("z = "+z);
    }
}
```

Output :
z = 1

Bitwise | (inclusive OR) operator

The | operator will set the resulting bit to 1 if *either* one of them is 1. It will return 0 only if both the bits are 0.

Example :

```
int x = 5;
int y = 9;
```

What will be x | y ?

5 -> 0 1 0 1

9 -> 1 0 0 1

1 1 0 1

x | y results in 13.

Bitwise | Operator Demo

```
class BitwiseExample2 {
    public static void main(String[] args) {
        int x = 5;
        int y = 9;
        int z = x | y;
        System.out.println("z = "+z);
    }
}
```

Output :
z = 13

Bitwise ^ (exclusive OR) operator

The ^ operator compares two bits to check whether these bits are different. If they are different, the result is 1. Otherwise, the result is 0. This operator is also known as XOR operator.

Example :

```
int x = 5;
int y = 9;
```

What will be x ^ y ?

5 -> 0 1 0 1

9 -> 1 0 0 1

1 1 0 0

x ^ y results in 12.

Bitwise ^ Operator Demo

```
class BitwiseExample3 {  
    public static void main(String[] args) {  
        int x = 5;  
        int y = 9;  
        int z = x ^ y;  
        System.out.println("z = "+z);  
    }  
}
```

Output :
z = 12

Control Statements

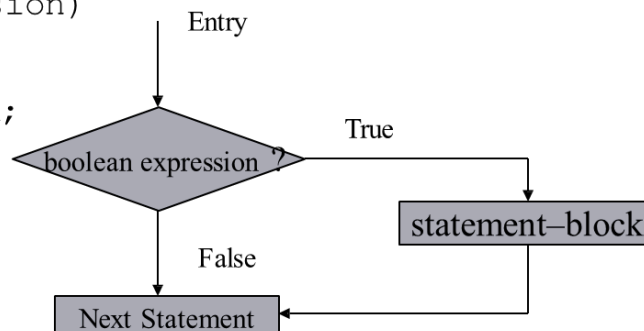
- Control statements are statements which alter the normal execution flow of a program
- There are three types of Control Statements in java

Selection statement	Iteration Statement	Jumping Statement
if	while	break
if – else	for	continue
switch	do – while	return

Simple if statement

syntax :

```
if(boolean expression)  
{  
    statement-block;  
}  
Next statement;
```



If - Example

/* This is an example of a if statement */

```
public class Test {  
    public static void main(String args[]) {  
        int x = 5;  
        if( x < 20 ) {  
            System.out.print("This is if statement");  
        }  
    }  
}
```

Output:

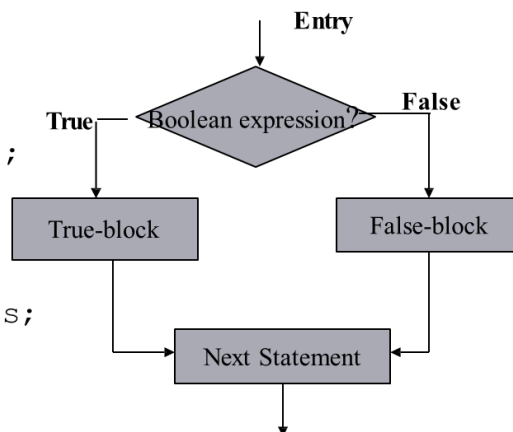
This is if statement

If..else statement

The if...else statement is an extension of simple if statement

Syntax:

```
if(boolean expression)  
{  
    True-block statements;  
}  
else  
{  
    False-block statements;  
}  
Next statement;
```



If – else Example

/ program to check given age input is eligible to vote or not using if- else*/*

```
public class Check {  
    public static void main(String[ ] args) {  
        int age;  
        age = Integer.parseInt(args[0]);  
        if(age>18) {  
            System.out.println("Eligible to vote");  
        }  
        else {  
            System.out.println("Not eligible to vote");  
        }  
    }  
}
```

Cascading if- else

- **Syntax:**

```
if (condition1) {  
    statement-1  
}  
...  
else if(condition-n) {  
    statement-n  
}  
else {  
    default statement  
}  
next statement
```

else - if Example

/* program to print seasons for a month input using if & else if */

```
public class ElseIfDemo {  
    public static void main(String[] args) {  
        int month = Integer.parseInt(args[0]);  
        if(month == 12 || month == 1 || month == 2)  
            System.out.println("Winter");  
        else if(month == 3 || month == 4 || month == 5)  
            System.out.println("Spring");  
        else if(month == 6 || month == 7 || month == 8)  
            System.out.println("Summer");  
        else if(month == 9 || month == 10 || month == 11)  
            System.out.println("Autumn");  
        else  
            System.out.println("invalid month");  
    }  
}
```

If args[0] is 6 then the Output is : Summer

Switch Case

- The switch-case conditional construct is a more structured way of testing for multiple conditions rather than resorting to a multiple if statement

Syntax:

```
switch (expression) {  
    case value-1 : case-1 block  
                    break;  
    case value-2 : case-2 block  
                    break;  
    default : default block  
            break;  
}  
statement-x;
```

Switch Case - Example

/* This is an example of a switch case statement*/

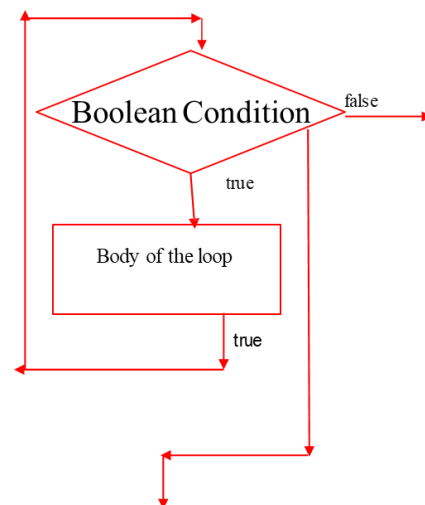
```
public class SwitchDemo {  
    public static void main(String[] args) {  
        int weekday = Integer.parseInt(args[0]);  
        switch(weekday) {  
            case 1:    System.out.println("Sunday");   break;  
            case 2:    System.out.println("Monday");  break;  
            case 3:    System.out.println("Tuesday"); break;  
            case 4:    System.out.println("Wednesday"); break;  
            case 5:    System.out.println("Thursday"); break;  
            case 6:    System.out.println("Friday");  break;  
            case 7:    System.out.println("Saturday"); break;  
            default:   System.out.println("Invalid day");  
        }  
    }  
}
```

If args[0] is 6 then the Output is : Friday

While loop

▪ Syntax

```
while(condition)  
{  
    Body of the loop  
}
```



while loop – Example

/* This is an example for a while loop */

```
public class Sample{  
    public static void main(String[] args) {  
        int i = 0;  
        while (i < 5) {  
            System.out.println("i: "+i);  
            i = i + 1;  
        }  
    }  
}
```

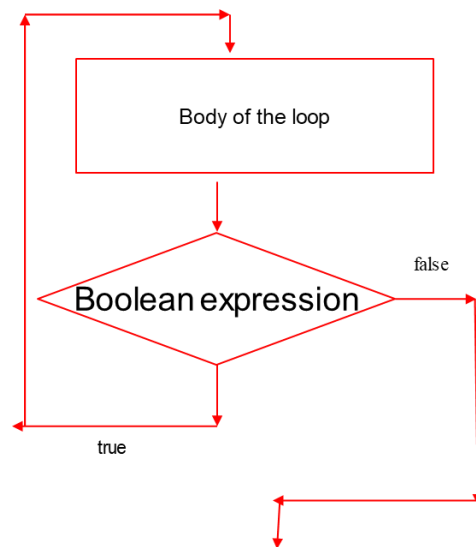
Output:

```
i: 0  
i: 1  
i: 2  
i: 3  
i: 4
```

do-while loop

■ Syntax:

```
do  
{  
    Body of the loop  
} while(boolean expression);
```



do...while loop – Example

/* This is an example of a do-while loop */

```
public class Sample {  
    public static void main(String[] args) {  
        int i =5;  
        do {  
            System.out.println("i: "+i);  
            i = i + 1;  
        } while (i < 5);  
    }  
}
```

Output:
i: 5

For loop

Syntax

```
for(initialization;condition;increment/decrement)  
{  
    Body of the loop  
}
```

For loop - Example

/* This is an example of a for loop */

```
public class Sample {  
    public static void main(String[] args) {  
        for (int i=1; i<=5; i++ ) {  
            System.out.println("i: "+i);  
        }  
    }  
}
```

Output:
i: 1
i: 2
i: 3
i: 4
i: 5

Enhanced for loop

Syntax:

```
for(declaration : expression)
{
    Body of loop
}
```

Enhanced for loop - Example

/* This is an example of a enhanced for loop */

```
public class Sample {
    public static void main(String[] args) {
        int [] numbers = {10, 20, 30, 40, 50};
        for(int i : numbers ) {
            System.out.println( "i: "+i );
        }
    }
}
```

Output:

```
i:10
i:20
i: 30
i:40
i:50
```

break statement

- While the execution of program, the break statement will terminate the iteration or switch case block
- When a break statement is encountered in a loop, the loop is exited and the program continues with the statements immediately following the loop
- When the loops are nested, the break will only terminate the corresponding loop body

break - Example

`/* This is an example of a break statement */`

```
public class Sample{
    public static void main(String[] args) {
        for (int i=1; i<=5; i++ ) {
            if(i==2)
                break;
            System.out.println("i: "+i);
        }
    }
}
```

Output:
i: 1

continue statement

- The continue statement skips the current iteration of a loop
- In while and do loops, continue causes the control to go directly to the test-condition and then continue the iteration process
- In case of for loop, the increment section of the loop is executed before the test-condition is evaluated

Continue - Example

```
/* This is an example of a continue loop */

public class Sample {
    public static void main(String[] args) {
        int [] numbers = {1, 2, 3, 4, 5};
        for(int i : numbers ) {
            if( i == 3 ) {
                continue;
            }
            System.out.println( "i: "+i );
        }
    }
}
```

Output:

```
i: 1
i:2
i:4
i:5
```

Good Programming Practices

if statement

- Always use {} for if statements
- Avoid the following error prone

```
if(condition)      //ERROR
    statement;
```

switch-case statement

Number per Line

- One declaration per line is recommended

```
int height;
int width;
```

is preferred over

```
int height,width;
```

Do not put different types on the same line

```
int height,width[];    //WRONG
```

Lab: Simple Interest Calculator

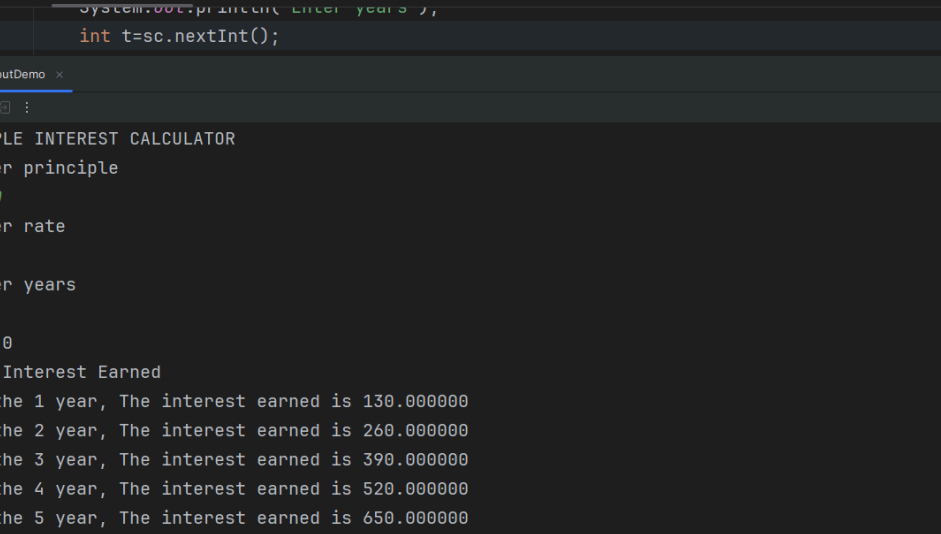
```
package com.Day2;
import java.util.Scanner;
```



```

public class InputDemo {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("SIMPLE INTEREST CALCULATOR");
        System.out.println("Enter principle");
        double p=sc.nextDouble();
        System.out.println("Enter rate");
        float r=sc.nextFloat();
        System.out.println("Enter years");
        int t=sc.nextInt();
        System.out.println("Enter your name");
        String name=sc.next();
        System.out.println("Enter your gender(M/F)");
        char gender=sc.next().charAt(0);
        double interest=(p*r*t)/100;
        System.out.println(interest);
        if(interest>5000){
            System.out.println("High Interest Earned");
        }
        else if(interest>2000 && interest<=5000){
            System.out.println("Moderate Interest Earned");
        }
        else{
            System.out.println("Low Interest Earned");
        }
        for(int year=1;year<=t;year++){
            double ipy=(p*r*year)/100;
            System.out.printf("At the %d year, The interest earned is %f%n",year,ipy);
        }
    }
}

```



```
1  System.out.println("Enter years ");
2
3  int t=sc.nextInt();
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Run InputDemo

SIMPLE INTEREST CALCULATOR

Enter principle

1000

Enter rate

13

Enter years

5

650.0

Low Interest Earned

At the 1 year, The interest earned is 130.000000

At the 2 year, The interest earned is 260.000000

At the 3 year, The interest earned is 390.000000

At the 4 year, The interest earned is 520.000000

At the 5 year, The interest earned is 650.000000

JavaLearning > src > com > Day2 > InputDemo > main 13:28 CRLF UTF-8 4 spaces