What is Java?

Simple.java

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

---

6. **C++ vs Java**

C++ Program Example

File: main.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
   cout << "Hello C++ Programming";
   return 0;
}
```

C++ Program Example

```cpp
File: main.cpp
#include <iostream>
using namespace std;
int main() {
   cout << "Hello C++ Programming";
   return 0;
}
```

First Java Program | Hello World Example

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

Output:

Hello Java

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

Output:

```
Hello Java
```

JVM (Java Virtual Machine)

```
//Let's see an example to print the classloader name
public class ClassLoaderExample
{
    public static void main(String[] args)
    {
        // Let's print the classloader name of current class.
        //Application/System classloader will load this class
        Class c=ClassLoaderExample.class;
        System.out.println(c.getClassLoader());
        //If we print the classloader name of String, it will print null becaus
e it is an
        //in-
built class which is found in rt.jar, so it is loaded by Bootstrap classload
er
        System.out.println(String.class.getClassLoader());
    }
}
```

[*Test it Now*]

Output:

```
sun.misc.Launcher$AppClassLoader@4e0e2f2a
null
```

Java Variables

Example to understand the types of variables in java

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

Java Variable Example: Add Two Numbers

```
public class Simple{
public static void main(String[] args){
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}
}
```

Output:

```
20
```

Java Variable Example: Widening

```
public class Simple{
public static void main(String[] args){
int a=10;
float f=a;
System.out.println(a);
System.out.println(f);
}}
```

Output:

```
10
```

```
10.0
```

Java Variable Example: Narrowing (Typecasting)

```java
public class Simple{
public static void main(String[] args){
float f=10.5f;
//int a=f;//Compile time error
int a=(int)f;
System.out.println(f);
System.out.println(a);
}}
```

Output:

```
10.5

10
```

Java Variable Example: Overflow

```java
class Simple{
public static void main(String[] args){
//Overflow
int a=130;
byte b=(byte)a;
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
130

-126
```

Java Variable Example: Adding Lower Type

```java
class Simple{
public static void main(String[] args){
byte a=10;
byte b=10;
//byte c=a+b;//Compile Time Error: because a+b=20 will be int
byte c=(byte)(a+b);
System.out.println(c);
}}
```

Output:

20

Operators in Java

Java Unary Operator Example: ++ and --

```
public class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Output:

```
10

12

12

10
```

Java Unary Operator Example 2: ++ and --

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21

}}
```

Output:

```
22

21
```

Java Unary Operator Example: ~ and !

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

```
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);//-11 (minus of total positive value which starts from 0)
System.out.println(~b);//9 (positive of total minus, positive starts from 0 )
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true
}}
```

Output:

```
-11

9

false

true
```

Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Java Arithmetic Operator Example

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:

```
15

5

50

2
```

```
0
```

Java Arithmetic Operator Example: Expression

```
public class OperatorExample{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}}
```

Output:

```
21
```

Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example

```
public class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

Output:

```
40

80

80

240
```

Java Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

Java Right Shift Operator Example

```
public OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Output:

```
2

5

2
```

Java Shift Operator Example: >> vs >>>

```
public class OperatorExample{
public static void main(String args[]){
    //For positive number, >> and >>> works same
    System.out.println(20>>2);
    System.out.println(20>>>2);
    //For negative number, >>> changes parity bit (MSB) to 0
    System.out.println(-20>>2);
    System.out.println(-20>>>2);
}}
```

Output:

```
5

5

-5

1073741819
```

-_

Java AND Operator Example: Logical && and Bitwise &

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```

Output:

```
false

false
```

Java AND Operator Example: Logical && vs Bitwise &

```java
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a++<c);//false && true = false
System.out.println(a);//10 because second condition is not checked
System.out.println(a<b&a++<c);//false && true = false
System.out.println(a);//11 because second condition is checked
}}
```

Output:

```
false

10

false

11
```

Java OR Operator Example: Logical || and Bitwise |

```java
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```

Output:

```
true
```

```
true

true

10

true

11
```

Java Ternary Operator

```
public class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

```
2
```

Another Example:

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

```
5
```

Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

Java Assignment Operator Example

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
14

16
```

Java Assignment Operator Example

```
public class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
```

Output:

```
13

9

18

9
```

Java Assignment Operator Example: Adding short

```
public class OperatorExample{
public static void main(String args[]){
short a=10;
short b=10;
//a+=b;//a=a+b internally so fine
a=a+b;//Compile time error because 10+10=20 now int
System.out.println(a);
}}
```

Output:

```
Compile time error
```

After type cast:

```
public class OperatorExample{
public static void main(String args[]){
short a=10;
short b=10;
a=(short)(a+b);//20 which is int now converted to short
System.out.println(a);
}}
```

Output:

```
20
```

Java Control Statements | Control Flow in Java

Simple if statement

Student.java

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y > 20) {
System.out.println("x + y is greater than 20");
}
}
}
```

Output:

```
x + y is greater than 20
```

---

if-else statement

Student.java

```
public class Student {
```

```
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y < 10) {
System.out.println("x + y is less than      10");
}   else {
System.out.println("x + y is greater than 20");
}
}
}
```

Output:

```
x + y is greater than 20
```

---

3) if-else-if ladder:

Student.java

```
public class Student {
public static void main(String[] args) {
String city = "Delhi";
if(city == "Meerut") {
System.out.println("city is meerut");
}else if (city == "Noida") {
System.out.println("city is noida");
}else if(city == "Agra") {
System.out.println("city is agra");
}else {
System.out.println(city);
}
}
}
```

Output:

```
Delhi
```

---

5. Nested if-statement

```
public class Student {
public static void main(String[] args) {
String city = "Delhi";
if(city == "Meerut") {
System.out.println("city is meerut");
}else if (city == "Noida") {
System.out.println("city is noida");
}else if(city == "Agra") {
System.out.println("city is agra");
}else {
System.out.println(city);
}
}
}
```

Output:

```
Delhi
```

4. Nested if-statement

Student.java

```
public class Student implements Cloneable {
public static void main(String[] args) {
int num = 2;
switch (num){
case 0:
System.out.println("number is 0");
break;
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}
}
}
```

Output:

```
2
```

Loop Statements

Calculation.java

```
public class Calculattion {
public static void main(String[] args) {
// TODO Auto-generated method stub
int sum = 0;
for(int j = 1; j<=10; j++) {
sum = sum + j;
}
System.out.println("The sum of first 10 natural numbers is " + sum);
}
}
```

Output:

```
The sum of first 10 natural numbers is 55
```

Java for-each loop

Calculation.java

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
String[] names = {"Java","C","C++","Python","JavaScript"};
System.out.println("Printing the content of the array names:\n");
for(String name:names) {
System.out.println(name);
}
}
}
```

Output:

```
Printing the content of the array names:


Java
C
```

C++

Python

JavaScript

Java while loop

Calculation .java

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
while(i<=10) {
System.out.println(i);
i = i + 2;
}
}
}
```

Output:

```
Printing the list of first 10 even numbers


0
2
4
6
8
10
```

Java do-while loop

Calculation.java

```
public class Calculation {
```

```
public static void main(String[] args) {
// TODO Auto-generated method stub
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
do {
System.out.println(i);
i = i + 2;
}while(i<=10);
}
}
```

Output:

```
Printing the list of first 10 even numbers

0

2

4

6

8

10
```

Java break statement

BreakExample.java

```
public class BreakExample {

public static void main(String[] args) {
// TODO Auto-generated method stub
for(int i = 0; i<= 10; i++) {
System.out.println(i);
if(i==6) {
break;
}
}
}
}
```

Output:

```
0
1
2
3
4
5
6
```

---

break statement example with labeled for loop

Calculation.java

```
public class Calculation {

public static void main(String[] args) {
// TODO Auto-generated method stub
a:
for(int i = 0; i<= 10; i++) {
b:
for(int j = 0; j<=15;j++) {
c:
for (int k = 0; k<=20; k++) {
System.out.println(k);
if(k==5) {
break a;
}
}
}

}
}
}
```

Output:

```
0
1
2
3
4
5
```

Java continue statement

```
public class ContinueExample {

public static void main(String[] args) {
// TODO Auto-generated method stub

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

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

if(j == 4) {
continue;
}
System.out.println(j);
}
}
}

}
```

Output:

```
0
1
2
3
5
1
2
3
5
2
3
5
```

Example:

File Name: IfExample.java

```
//Java Program to demonstate the use of if statement.
public class IfExample {
public static void main(String[] args) {
    //defining an 'age' variable
    int age=20;
    //checking the age
    if(age>18){
        System.out.print("Age is greater than 18");
    }
}
}
```

Test it Now

Output:

```
Age is greater than 18
```

Java if-else Statement

File Name: IfElseExample.java

```
//A Java Program to demonstrate the use of if-else statement.
//It is a program of odd and even number.
public class IfElseExample {
public static void main(String[] args) {
    //defining a variable
    int number=13;
    //Check if the number is divisible by 2 or not
    if(number%2==0){
        System.out.println("even number");
    }else{
        System.out.println("odd number");
    }
}
}
```

Test it Now

Output:

```
odd number
```

Leap Year Example Using IfElse:

A year is leap, if it is divisible by 4 and 400. But, not by 100.

File Name: LeapYearExample.java

```
public class LeapYearExample {
public static void main(String[] args) {
    int year=2020;
    if(((year % 4 ==0) && (year % 100 !=0)) || (year % 400==0)){
        System.out.println("LEAP YEAR");
    }
    else{
        System.out.println("COMMON YEAR");
    }
}
}
```

Output:

```
LEAP YEAR
```

Java if-else-if ladder Statement

File Name: IfElseIfExample.java

```
//Java Program to demonstrate the use of If else-if ladder.
//It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.
public class IfElseIfExample {
public static void main(String[] args) {
    int marks=65;

    if(marks<50){
        System.out.println("fail");
    }
    else if(marks>=50 && marks<60){
        System.out.println("D grade");
    }
    else if(marks>=60 && marks<70){
        System.out.println("C grade");
    }
    else if(marks>=70 && marks<80){
        System.out.println("B grade");
    }
    else if(marks>=80 && marks<90){
        System.out.println("A grade");
    }else if(marks>=90 && marks<100){
        System.out.println("A+ grade");
    }else{
        System.out.println("Invalid!");
    }
}
}
```

Output:

```
C grade
```

Program to check POSITIVE, NEGATIVE or ZERO using if-else-if:

File Name: PositiveNegativeExample.java

```
public class PositiveNegativeExample {
public static void main(String[] args) {
    int number=-13;
    if(number>0){
    System.out.println("POSITIVE");
    }else if(number<0){
    System.out.println("NEGATIVE");
    }else{
    System.out.println("ZERO");
   }
}
}
```

Output:

```
NEGATIV
```

Java Nested if statement

File Name: JavaNestedIfExample.java

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample {
public static void main(String[] args) {
    //Creating two variables for age and weight
    int age=20;
```

```
  int weight=80;
  //applying condition on age and weight
  if(age>=18){
    if(weight>50){
      System.out.println("You are eligible to donate blood");
    }
  }
}}
```

Test it Now

Output:

```
You are eligible to donate blood
```

Example 2:

File Name: JavaNestedIfExample2.java

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample2 {
public static void main(String[] args) {
  //Creating two variables for age and weight
  int age=25;
  int weight=48;
  //applying condition on age and weight
  if(age>=18){
    if(weight>50){
      System.out.println("You are eligible to donate blood");
    } else{
      System.out.println("You are not eligible to donate blood");
    }
  } else{
   System.out.println("Age must be greater than 18");
  }
} }
```

Test it Now

Output:

```
You are not eligible to donate blood
```

Ternary Operator

File Name: TernaryExample.java

```java
public class IfElseTernaryExample {
public static void main(String[] args) {
    int number=13;
    //Using ternary operator
    String output=(number%2==0)?"even number":"odd number";
    System.out.println(output);
}
}
```

Output:

```
odd number
```

Java Switch Statement

SwitchExample.java

```java
public class SwitchExample {
public static void main(String[] args) {
    //Declaring a variable for switch expression
    int number=20;
    //Switch expression
    switch(number){
    //Case statements
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
    //Default case statement
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

Test it Now

Output:

```
20
```

Finding Month Example:

SwitchMonthExample.javaHTML

```
//Java Program to demonstrate the example of Switch statement
//where we are printing month name for the given number
public class SwitchMonthExample {
public static void main(String[] args) {
    //Specifying month number
    int month=7;
    String monthString="";
    //Switch statement
    switch(month){
    //case statements within the switch block
    case 1: monthString="1 - January";
    break;
    case 2: monthString="2 - February";
    break;
    case 3: monthString="3 - March";
    break;
    case 4: monthString="4 - April";
    break;
    case 5: monthString="5 - May";
    break;
    case 6: monthString="6 - June";
    break;
    case 7: monthString="7 - July";
    break;
    case 8: monthString="8 - August";
    break;
    case 9: monthString="9 - September";
    break;
    case 10: monthString="10 - October";
    break;
    case 11: monthString="11 - November";
    break;
    case 12: monthString="12 - December";
    break;
    default:System.out.println("Invalid Month!");
    }
    //Printing month of the given number
    System.out.println(monthString);
}
}
```

Test it Now

Output:

```
7 – July
```

Program to check Vowel or Consonant:

If the character is A, E, I, O, or U, it is vowel otherwise consonant. It is not case-sensitive.

SwitchVowelExample.java

```
public class SwitchVowelExample {
public static void main(String[] args) {
    char ch='O';
    switch(ch)
```

```
{
  case 'a':
    System.out.println("Vowel");
    break;
  case 'e':
    System.out.println("Vowel");
    break;
  case 'i':
    System.out.println("Vowel");
    break;
  case 'o':
    System.out.println("Vowel");
    break;
  case 'u':
    System.out.println("Vowel");
    break;
  case 'A':
    System.out.println("Vowel");
    break;
  case 'E':
    System.out.println("Vowel");
    break;
  case 'I':
    System.out.println("Vowel");
    break;
  case 'O':
    System.out.println("Vowel");
    break;
  case 'U':
    System.out.println("Vowel");
    break;
```

```
        default:
            System.out.println("Consonant");
    }
}
}
```

Output:

```
Vowel
```

Java Switch Statement is fall-through

Example:

SwitchExample2.java

```
//Java Switch Example where we are omitting the
//break statement
public class SwitchExample2 {
public static void main(String[] args) {
    int number=20;
    //switch expression with int value
    switch(number){
    //switch cases without break statements
    case 10: System.out.println("10");
    case 20: System.out.println("20");
    case 30: System.out.println("30");
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

Test it Now

Output:

```
20
30
Not in 10, 20 or 30
```

Java Switch Statement with String

Example:

SwitchStringExample.java

```
//Java Program to demonstrate the use of Java Switch
//statement with String
public class SwitchStringExample {
public static void main(String[] args) {
    //Declaring String variable
    String levelString="Expert";
    int level=0;
    //Using String in Switch expression
    switch(levelString){
    //Using String Literal in Switch case
    case "Beginner": level=1;
    break;
    case "Intermediate": level=2;
    break;
    case "Expert": level=3;
    break;
    default: level=0;
    break;
    }
    System.out.println("Your Level is: "+level);
}
}
```

Test it Now

Output:

```
Your Level is: 3
```

Java Nested Switch Statement

We can use switch statement inside other switch statement in Java. It is known as nested switch statement.

Example:

NestedSwitchExample.java

```
//Java Program to demonstrate the use of Java Nested Switch
public class NestedSwitchExample {
    public static void main(String args[])
      {
      //C - CSE, E - ECE, M - Mechanical
        char branch = 'C';
        int collegeYear = 4;
        switch( collegeYear )
        {
            case 1:
                System.out.println("English, Maths, Science");
                break;
            case 2:
                switch( branch )
                {
                    case 'C':
                        System.out.println("Operating System, Java, Data Structure");
                        break;
                    case 'E':
                        System.out.println("Micro processors, Logic switching theory");
                        break;
                    case 'M':
                        System.out.println("Drawing, Manufacturing Machines");
                        break;
```

```java
      }
      break;
   case 3:
      switch( branch )
      {
         case 'C':
            System.out.println("Computer Organization, MultiMedia");
            break;
         case 'E':
            System.out.println("Fundamentals of Logic Design, Microelectronics");
            break;
         case 'M':
            System.out.println("Internal Combustion Engines, Mechanical Vibration");
            break;
      }
      break;
   case 4:
      switch( branch )
      {
         case 'C':
            System.out.println("Data Communication and Networks, MultiMedia");
            break;
         case 'E':
            System.out.println("Embedded System, Image Processing");
            break;
         case 'M':
            System.out.println("Production Technology, Thermal Engineering");
            break;
      }
      break;
}
```

```
    }
}
```

[Test it Now](#)

Output:

```
Data Communication and Networks, MultiMedia
```

Java Enum in Switch Statement

Java allows us to use enum in switch statement. Java enum is a class that represent the group of constants. (immutable such as final variables). We use the keyword enum and put the constants in curly braces separated by comma.

Example:

JavaSwitchEnumExample.java

```
//Java Program to demonstrate the use of Enum
//in switch statement
public class JavaSwitchEnumExample {
      public enum Day {  Sun, Mon, Tue, Wed, Thu, Fri, Sat  }
      public static void main(String args[])
      {
        Day[] DayNow = Day.values();
          for (Day Now : DayNow)
          {
              switch (Now)
              {
                  case Sun:
                      System.out.println("Sunday");
                      break;
                  case Mon:
                      System.out.println("Monday");
                      break;
                  case Tue:
                      System.out.println("Tuesday");
                      break;
```

```
                case Wed:
                  System.out.println("Wednesday");
                  break;
                case Thu:
                  System.out.println("Thursday");
                  break;
                case Fri:
                  System.out.println("Friday");
                  break;
                case Sat:
                  System.out.println("Saturday");
                  break;
              }
          }
      }
}
```

Test it Now

Output:

```
Sunday
Monday
Twesday
Wednesday
Thursday
Friday
Saturday
```

Java Wrapper in Switch Statement

Java allows us to use four wrapper classes: Byte, Short, Integer and Long in switch statement.

Example:

WrapperInSwitchCaseExample.java

```
//Java Program to demonstrate the use of Wrapper class
```

```
//in switch statement
public class WrapperInSwitchCaseExample {
      public static void main(String args[])
      {
          Integer age = 18;
          switch (age)
          {
              case (16):
                  System.out.println("You are under 18.");
                  break;
              case (18):
                  System.out.println("You are eligible for vote.");
                  break;
              case (65):
                  System.out.println("You are senior citizen.");
                  break;
              default:
                  System.out.println("Please give the valid age.");
                  break;
          }
      }
}
```

Test it Now

Output:

Loops in Java

```
You are eligible for vote
```

Loops in Java

Example:

ForExample.java

```
//Java Program to demonstrate the example of for loop
//which prints table of 1
```

```
public class ForExample {
public static void main(String[] args) {
    //Code of Java for loop
    for(int i=1;i<=10;i++){
        System.out.println(i);
    }
}
}
```

Test it Now

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Java Nested for Loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Example:

NestedForExample.java

```
public class NestedForExample {
public static void main(String[] args) {
//loop of i
for(int i=1;i<=3;i++){
//loop of j
for(int j=1;j<=3;j++){
        System.out.println(i+" "+j);
}//end of i
}//end of j
}
}
```

Output:

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

Pyramid Example 1:

PyramidExample.java

```
public class PyramidExample {
public static void main(String[] args) {
for(int i=1;i<=5;i++){
for(int j=1;j<=i;j++){
        System.out.print("* ");
}
System.out.println();//new line
}
}
}
```

Output:

```
*
```

```
* *
* * *
* * * *
* * * * *
```

Pyramid Example 2:

PyramidExample2.java

```
public class PyramidExample2 {
public static void main(String[] args) {
int term=6;
for(int i=1;i<=term;i++){
for(int j=term;j>=i;j--){
        System.out.print("* ");
}
System.out.println();//new line
}
}
}
```

Output:

```
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

Java for-each Loop

Example:

ForEachExample.java

```
//Java For-each loop example which prints the
//elements of the array
```

```
public class ForEachExample {
public static void main(String[] args) {
    //Declaring an array
    int arr[]={12,23,44,56,78};
    //Printing array using for-each loop
    for(int i:arr){
        System.out.println(i);
    }
}
}
```

Test it Now

Output:

```
12
23
44
56
78
```

Java Labeled For Loop

Example:

LabeledForExample.java

```
//A Java program to demonstrate the use of labeled for loop
public class LabeledForExample {
public static void main(String[] args) {
    //Using Label for outer and for loop
    aa:
        for(int i=1;i<=3;i++){
            bb:
                for(int j=1;j<=3;j++){
                    if(i==2&&j==2){
                        break aa;
                    }
```

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

Output:

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

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

LabeledForExample2.java

```
public class LabeledForExample2 {
public static void main(String[] args) {
  aa:
    for(int i=1;i<=3;i++){
      bb:
        for(int j=1;j<=3;j++){
          if(i==2&&j==2){
            break bb;
          }
          System.out.println(i+" "+j);
        }
    }
}
}
```

Output:

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

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

Java Infinitive for Loop

ForExample.java

```
//Java program to demonstrate the use of infinite for loop
//which prints an statement
public class ForExample {
public static void main(String[] args) {
    //Using no condition in for loop
    for(;;){
        System.out.println("infinitive loop");
    }
}
}
```

Output:

```
infinitive loop
infinitive loop
infinitive loop
infinitive loop
infinitive loop
ctrl+c
```

Now, you need to press ctrl+c to exit from the program.

Java for Loop vs while Loop vs do-while Loop

Java While Loop

WhileExample.java

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

[Test it Now]

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Java Infinitive While Loop

Example:

WhileExample2.java

```
public class WhileExample2 {
public static void main(String[] args) {
 // setting the infinite while loop by passing true to the condition
    while(true){
        System.out.println("infinitive while loop");
    }
}
}
```

Output:

```
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
ctrl+c
```

Java do-while Loop

DoWhileExample.java

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

Test it Now

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Java Infinitive do-while Loop

Example:

DoWhileExample2.java

```
public class DoWhileExample2 {
public static void main(String[] args) {
    do{
        System.out.println("infinitive do while loop");
    }while(true);
}
}
```

Output:

```
infinitive do while loop
infinitive do while loop
infinitive do while loop
ctrl+c
```

In the above code, we need to enter Ctrl + C command to terminate the infinite loop.

Java Break Statement with Loop

Example:

BreakExample.java

```
//Java Program to demonstrate the use of break statement
//inside the for loop.
public class BreakExample {
public 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
```

Java Break Statement with Inner Loop

BreakExample2.java

```
//Java Program to illustrate the use of break statement
//inside an inner loop
public class BreakExample2 {
public 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
```

Java Break Statement with Labeled For Loop

We can use break statement with a label. The feature is introduced since JDK 1.5. So, we can break any loop in Java now whether it is outer or inner loop.

Example:

BreakExample3.java

```
//Java Program to illustrate the use of continue statement
//with label inside an inner loop to break outer loop
public class BreakExample3 {
public static void main(String[] args) {
            aa:
            for(int i=1;i<=3;i++){
                  bb:
                  for(int j=1;j<=3;j++){
                     if(i==2&&j==2){
                        //using break statement with label
                        break aa;
                     }
                     System.out.println(i+" "+j);
                  }
            }
}
}
```

Output:

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

Java Break Statement in while loop

BreakWhileExample.java

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

Output:

```
1
2
3
4
```

Java Break Statement in do-while loop

BreakDoWhileExample.java

```
//Java Program to demonstrate the use of break statement
//inside the Java do-while loop.
```

```
public class BreakDoWhileExample {
public 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 Example

ContinueExample.java

```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample {
public 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);
    }
}
}
```

Test it Now

Output:

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

As you can see in the above output, 5 is not printed on the console. It is because the loop is continued when it reaches to 5.

Java Continue Statement with Inner Loop

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

ContinueExample2.java

```
//Java Program to illustrate the use of continue statement
//inside an inner loop
public class ContinueExample2 {
public 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
```

Java Continue Statement with Labelled For Loop

We can use continue statement with a label. This feature is introduced since JDK 1.5. So, we can continue any loop in Java now whether it is outer loop or inner.

Example:

ContinueExample3.java

```
//Java Program to illustrate the use of continue statement
//with label inside an inner loop to continue outer loop
public class ContinueExample3 {
public static void main(String[] args) {
        aa:
        for(int i=1;i<=3;i++){
          bb:
          for(int j=1;j<=3;j++){
            if(i==2&&j==2){
              //using continue statement with label
              continue aa;
            }
            System.out.println(i+" "+j);
          }
      }
}
}
```

Output:

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

Java Continue Statement in while loop

ContinueWhileExample.java

```
//Java Program to demonstrate the use of continue statement
//inside the while loop.
public class ContinueWhileExample {
public static void main(String[] args) {
  //while loop
  int i=1;
  while(i<=10){
    if(i==5){
      //using continue statement
```

```
            i++;
            continue;//it will skip the rest statement
        }
        System.out.println(i);
        i++;
    }
}
}
```

Test it Now

Output:

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

Java Continue Statement in do-while Loop

ContinueDoWhileExample.java

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

Test it Now

Output:

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

Java Comments

CommentExample1.java

```
public class CommentExample1 {
public static void main(String[] args) {
int i=10; // i is a variable with value 10
System.out.println(i);  //printing the variable i
}
}
```

Output:

```
10
```

CommentExample2.java

```
public class CommentExample2 {
public static void main(String[] args) {
/* Let's declare and
 print variable in java. */
  int i=10;
    System.out.println(i);
/* float j = 5.9;
    float k = 4.4;
    System.out.println( j + k ); */
}
}
```

Output:

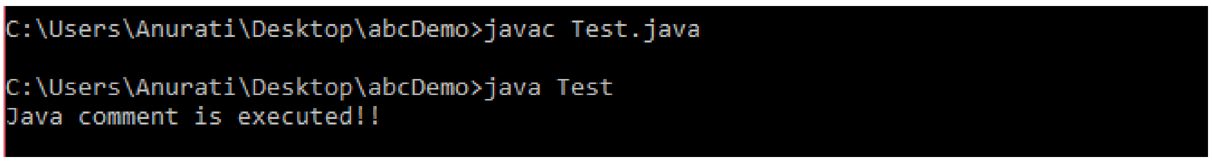
```
10
```

Are Java comments executable?

Test.java

```
public class Test{
    public static void main(String[] args) {
        //the below comment will be executed
// \u000d System.out.println("Java comment is executed!!");
    }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac Test.java

C:\Users\Anurati\Desktop\abcDemo>java Test
Java comment is executed!!
```

The above code generate the output because the compiler parses the Unicode character \u000d as a new line before the lexical transformation, and thus the code is transformed as shown below:

Test.java

```
public class Test{
    public static void main(String[] args) {
        //the below comment will be executed
//
System.out.println("Java comment is executed!!");
    }
}
```

Thus, the Unicode character shifts the print statement to next line and it is executed as a normal Java code.

Objects and Classes in Java

```
//Java Program to demonstrate having the main method in
//another class
//Creating Student class.
class Student{
 int id;
 String name;
}
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Output:

```
0
Null
```

---

3 Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable

2. By method
3. By constructor

1) Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

File: TestStudent2.java

```
class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

[Test it Now]()

Output:

```
101 Sonoo
```

We can also create multiple objects and store information in it through reference variable.

File: TestStudent3.java

```
class Student{
 int id;
 String name;
}
class TestStudent3{
 public static void main(String args[]){
  //Creating objects
  Student s1=new Student();
  Student s2=new Student();
  //Initializing objects
  s1.id=101;
  s1.name="Sonoo";
  s2.id=102;
  s2.name="Amit";
  //Printing data
  System.out.println(s1.id+" "+s1.name);
  System.out.println(s2.id+" "+s2.name);
 }
}
```

Test it Now

Output:

```
101 Sonoo
102 Amit
```

---

2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

File: TestStudent4.java

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
```

```
public static void main(String args[]){
 Student s1=new Student();
 Student s2=new Student();
 s1.insertRecord(111,"Karan");
 s2.insertRecord(222,"Aryan");
 s1.displayInformation();
 s2.displayInformation();
}
}
```

Test it Now

Output:

```
111 Karan
222 Aryan
```

Object and Class Example: Initialization through a constructor

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

File: TestEmployee.java

```
class Employee{
    int id;
    String name;
    float salary;
    void insert(int i, String n, float s) {
        id=i;
        name=n;
        salary=s;
    }
    void display(){System.out.println(id+" "+name+" "+salary);}
}
public class TestEmployee {
public static void main(String[] args) {
    Employee e1=new Employee();
    Employee e2=new Employee();
    Employee e3=new Employee();
    e1.insert(101,"ajeet",45000);
    e2.insert(102,"irfan",25000);
    e3.insert(103,"nakul",55000);
    e1.display();
    e2.display();
    e3.display();
}
}
```

Test it Now

Output:

```
101 ajeet 45000.0
102 irfan 25000.0
103 nakul 55000.0
```

Object and Class Example: Rectangle

There is given another example that maintains the records of Rectangle class.

File: TestRectangle1.java

```
class Rectangle{
 int length;
 int width;
 void insert(int l, int w){
  length=l;
  width=w;
 }
 void calculateArea(){System.out.println(length*width);}
}
class TestRectangle1{
 public static void main(String args[]){
  Rectangle r1=new Rectangle();
  Rectangle r2=new Rectangle();
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
}
}
```

Test it Now

Output:

```
55
45
```

---

anonymous object in Java.

```
class Calculation{
 void fact(int  n){
  int fact=1;
  for(int i=1;i<=n;i++){
   fact=fact*i;
  }
 System.out.println("factorial is "+fact);
}
public static void main(String args[]){
 new Calculation().fact(5);//calling method with anonymous object
}
}
```

Output:

```
Factorial is 120
```

Creating multiple objects by one type only

```
//Java Program to illustrate the use of Rectangle class which
//has length and width data members
class Rectangle{
 int length;
 int width;
 void insert(int l,int w){
  length=l;
  width=w;
 }
 void calculateArea(){System.out.println(length*width);}
}
class TestRectangle2{
 public static void main(String args[]){
  Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
}
}
```

Test it Now

Output:

```
55
45
```

Real World Example: Account

File: TestAccount.java

```
//Java Program to demonstrate the working of a banking-system
//where we deposit and withdraw amount from our account.
//Creating an Account class which has deposit() and withdraw() methods
class Account{
int acc_no;
String name;
float amount;
//Method to initialize object
```

```
void insert(int a,String n,float amt){
acc_no=a;
name=n;
amount=amt;
}
//deposit method
void deposit(float amt){
amount=amount+amt;
System.out.println(amt+" deposited");
}
//withdraw method
void withdraw(float amt){
if(amount<amt){
System.out.println("Insufficient Balance");
}else{
amount=amount-amt;
System.out.println(amt+" withdrawn");
}
}
//method to check the balance of the account
void checkBalance(){System.out.println("Balance is: "+amount);}
//method to display the values of an object
void display(){System.out.println(acc_no+" "+name+" "+amount);}
}
//Creating a test class to deposit and withdraw amount
class TestAccount{
public static void main(String[] args){
Account a1=new Account();
a1.insert(832345,"Ankit",1000);
a1.display();
a1.checkBalance();
a1.deposit(40000);
a1.checkBalance();
a1.withdraw(15000);
a1.checkBalance();
}}
```

Test it Now

Output:

```
832345 Ankit 1000.0
Balance is: 1000.0
40000.0 deposited
Balance is: 41000.0
15000.0 withdrawn
Balance is: 26000.0
```

Constructors in Java

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Test it Now

Output:

```
Bike is created
```

Example of default constructor that displays the default values

```
//Let us see another example of default constructor
```

```
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

[Test it Now]

Output:

```
0 null
0 null
```

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }
```

```java
    //method to display the values
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

[Test it Now]()

Output:

```
111 Karan
222 Aryan
```

Example of Constructor Overloading

```java
//Java program to overload constructors
class Student5{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i,String n){
    id = i;
    name = n;
    }
    //creating three arg constructor
    Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Test it Now

Output:

```
111 Karan 0
222 Aryan 25
```

Java Copy Constructor

```
//Java program to initialize the values from one object to another object.
class Student6{
    int id;
    String name;
    //constructor to initialize integer and string
    Student6(int i,String n){
    id = i;
    name = n;
    }
    //constructor to initialize another object
    Student6(Student6 s){
    id = s.id;
    name =s.name;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }
 }
```

[Test it Now]

Output:

```
111 Karan
111 Karan
```

Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
class Student7{
    int id;
    String name;
    Student7(int i,String n){
    id = i;
    name = n;
    }
    Student7(){}
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student7 s1 = new Student7(111,"Karan");
    Student7 s2 = new Student7();
    s2.id=s1.id;
    s2.name=s1.name;
```

```
 s1.display();
 s2.display();
}
}
```

[Test it Now]

Output:

```
111 Karan
111 Karan
```

Example of static variable

```
//Java Program to demonstrate the use of static variable
class Student{
  int rollno;//instance variable
  String name;
  static String college ="ITS";//static variable
  //constructor
  Student(int r, String n){
  rollno = r;
  name = n;
  }
  //method to display the values
  void display (){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to show the values of objects
public class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by the single line of code
 //Student.college="BBDIT";
 s1.display();
 s2.display();
 }
}
```

Test it Now

Output:

```
111 Karan ITS
222 Aryan ITS
```

Program of the counter without static variable

```
//Java Program to demonstrate the use of an instance variable
//which get memory each time when we create an object of the class.
class Counter{
int count=0;//will get memory each time when the instance is created

Counter(){
count++;//incrementing value
System.out.println(count);
}

public static void main(String args[]){
//Creating objects
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Test it Now

Output:

```
1
1
1
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
//Java Program to illustrate the use of static variable which
//is shared with all objects.
class Counter2{
static int count=0;//will get memory only once and retain its value

Counter2(){
count++;//incrementing the value of static variable
System.out.println(count);
}

public static void main(String args[]){
//creating objects
Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();
}
}
```

Test it Now

Output:

```
1

2

3
```

Example of static method

```
//Java Program to demonstrate the use of a static method.
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    //static method to change the value of static variable
    static void change(){
    college = "BBDIT";
    }
    //constructor to initialize the variable
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object
public class TestStaticMethod{
   public static void main(String args[]){
   Student.change();//calling change method
   //creating objects
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   Student s3 = new Student(333,"Sonoo");
   //calling display method
   s1.display();
   s2.display();
   s3.display();
   }
}
```

Test it Now

```
Output:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT
```

Example of static method

```
//Java Program to demonstrate the use of a static method.
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    //static method to change the value of static variable
    static void change(){
    college = "BBDIT";
    }
    //constructor to initialize the variable
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object
public class TestStaticMethod{
   public static void main(String args[]){
   Student.change();//calling change method
   //creating objects
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   Student s3 = new Student(333,"Sonoo");
   //calling display method
   s1.display();
   s2.display();
   s3.display();
```

```
    }
}
```

Test it Now

Output:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

Another example of a static method that performs a normal calculation

```
//Java Program to get the cube of a given number using the static method

class Calculate{
  static int cube(int x){
  return x*x*x;
  }

  public static void main(String args[]){
  int result=Calculate.cube(5);
  System.out.println(result);
  }
}
```

Test it Now

Output:125

Restrictions for the static method

```
class A{
 int a=40;//non static

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

[Test it Now]

Output:Compile Time Error

Example of static block

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

[Test it Now]

Output:static block is invoked

Hello main

Can we execute a program without main() method?

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the main method.

```
class A3{
  static{
  System.out.println("static block is invoked");
  System.exit(0);
  }
}
```

[Test it Now]

Output:

static block is invoked

Since JDK 1.7 and above, output would be:

Error: Main method not found in class A3, please define the main method as:

public static void main(String[] args)

or a JavaFX application class must extend javafx.application.Application

---

this keyword in Java

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Test it Now

Output:

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Test it Now

Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

Program where this keyword is not required

```
class Student{
int rollno;
String name;
float fee;
Student(int r,String n,float f){
rollno=r;
name=n;
fee=f;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
```

```
 
class TestThis3{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Test it Now

Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

---

this: to invoke current class method

```
class A{
void m(){System.out.println("hello m");}
void n(){
System.out.println("hello n");
//m();//same as this.m()
this.m();
}
}
class TestThis4{
public static void main(String args[]){
A a=new A();
a.n();
}}
```

Test it Now

Output:

```
hello n
hello m
```

---

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Calling default constructor from parameterized constructor:

```
class A{
A(){System.out.println("hello a");}
A(int x){
this();
System.out.println(x);
}
}
class TestThis5{
public static void main(String args[]){
A a=new A(10);
}}
```

Test it Now

Output:

```
hello a
10
```

Calling parameterized constructor from default constructor:

```
class A{
A(){
this(5);
System.out.println("hello a");
}
A(int x){
System.out.println(x);
}
}
class TestThis6{
```

```
public static void main(String args[]){
A a=new A();
}}
```

[Test it Now]

Output:

```
5
hello a
```

Real usage of this() constructor call

The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float fee){
this(rollno,name,course);//reusing constructor
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
}
class TestThis7{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
```

```java
s2.display();
}}
```

Test it Now

Output:

```
111 ankit java 0.0
112 sumit java 6000.0
```

Rule: Call to this() must be the first statement in constructor.

```java
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float fee){
this.fee=fee;
this(rollno,name,course);//C.T.Error
}
void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
}
class TestThis8{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
s2.display();
}}
```

Test it Now

Output:

Compile Time Error: Call to this must be first statement in constructor

---

4) this: to pass as an argument in the method

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
class S2{
  void m(S2 obj){
  System.out.println("method is invoked");
  }
  void p(){
  m(this);
  }
  public static void main(String args[]){
  S2 s1 = new S2();
  s1.p();
  }
}
```

Test it Now

Output:

```
method is invoked
```

---

5) this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
class B{
  A4 obj;
  B(A4 obj){
    this.obj=obj;
  }
  void display(){
    System.out.println(obj.data);//using data member of A4 class
  }
}

class A4{
 int data=10;
 A4(){
  B b=new B(this);
  b.display();
 }
 public static void main(String args[]){
  A4 a=new A4();
 }
}
```

[Test it Now]

Output:10

Example of this keyword that you return as a statement from the method

```
class A{
A getA(){
return this;
}
void msg(){System.out.println("Hello java");}
}
class Test1{
public static void main(String args[]){
new A().getA().msg();
}
}
```

[Test it Now]

Output:

```
Hello java
```

Proving this keyword

Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same.

```
class A5{
void m(){
System.out.println(this);//prints same reference ID
}
public static void main(String args[]){
A5 obj=new A5();
System.out.println(obj);//prints the reference ID
obj.m();
}
}
```

Test it Now

Output:

```
A5@22b3ea59
A5@22b3ea59
```

Inheritance in Java

```
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);
}
}
```

[***Test it Now***]

Programmer salary is:40000.0

Bonus of programmer is:10000

Single Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

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

Multilevel Inheritance Example

File: TestInheritance2.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
```

```
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

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

.

Hierarchical Inheritance Example

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

[*Test it Now*]

Compile Time Error

.

File: TestInheritance3.java

```java
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

Q) Why multiple inheritance is not supported in java?

```java
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

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

```
 C obj=new C();
 obj.msg();//Now which msg() method would be invoked?
}
}
```

Test it Now

Compile Time Error

Aggregation in Java

```
class Operation{
 int square(int n){
  return n*n;
 }
}

class Circle{
 Operation op;//aggregation
 double pi=3.14;

double area(int radius){
  op=new Operation();
  int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).
  return pi*rsquare;
}



public static void main(String args[]){
  Circle c=new Circle();
  double result=c.area(5);
  System.out.println(result);
}
}
```

Test it Now

Output:78.5

Address.java

```
public class Address {
String city,state,country;

public Address(String city, String state, String country) {
    this.city = city;
    this.state = state;
    this.country = country;
}

}
```

Emp.java

```
public class Emp {
int id;
String name;
Address address;

public Emp(int id, String name,Address address) {
    this.id = id;
    this.name = name;
    this.address=address;
}

void display(){
System.out.println(id+" "+name);
System.out.println(address.city+" "+address.state+" "+address.country);
}

public static void main(String[] args) {
Address address1=new Address("gzb","UP","india");
Address address2=new Address("gno","UP","india");

Emp e=new Emp(111,"varun",address1);
Emp e2=new Emp(112,"arun",address2);

e.display();
e2.display();

}
}
```

[Test it Now]

```
Output:111 varun
gzb UP india
112 arun
gno UP india
```

Method Overloading: changing no. of arguments

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

[Test it Now]

```
Output:
22
33
```

---

2) Method Overloading: changing data type of arguments

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Test it Now

Output:

```
22
24.9
```

Q) Why Method Overloading is not possible by changing the return type of method only?

```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

Test it Now

Output:

```
Compile Time Error: method add(int,int) is already defined in class Adder
```

.

Can we overload java main() method?

the simple example:

```
class TestOverloading4{
public static void main(String[] args){System.out.println("main with String[]");}
public static void main(String args){System.out.println("main with String");}
public static void main(){System.out.println("main without args");}
}
```

[***Test it Now***]

Output:

```
main with String[])
```

Method Overloading and Type Promotion

Example of Method Overloading with TypePromotion

```
class OverloadingCalculation1{
  void sum(int a,long b){System.out.println(a+b);}
  void sum(int a,int b,int c){System.out.println(a+b+c);}

  public static void main(String args[]){
  OverloadingCalculation1 obj=new OverloadingCalculation1();
  obj.sum(20,20);//now second int literal will be promoted to long
  obj.sum(20,20,20);

  }
}
```

[***Test it Now***]

```
Output:40
60
```

Example of Method Overloading with Type Promotion if matching found

If there are matching type arguments in the method, type promotion is not performed.

```
class OverloadingCalculation2{
  void sum(int a,int b){System.out.println("int arg method invoked");}
  void sum(long a,long b){System.out.println("long arg method invoked");}

  public static void main(String args[]){
  OverloadingCalculation2 obj=new OverloadingCalculation2();
  obj.sum(20,20);//now int arg sum() method gets invoked
  }
}
```

Test it Now

Output:int arg method invoked

Example of Method Overloading with Type Promotion in case of ambiguity

Iclass OverloadingCalculation3{

```
  void sum(int a,long b){System.out.println("a method invoked");}
  void sum(long a,int b){System.out.println("b method invoked");}

  public static void main(String args[]){
  OverloadingCalculation3 obj=new OverloadingCalculation3();
  obj.sum(20,20);//now ambiguity
  }
}
```

Test it Now

Output:Compile Time Error

Method Overriding in Java

Understanding the problem without method overriding

Let's understand the problem that we may face in the program if we don't use method overriding.

```
//Java Program to demonstrate why we need method overriding
//Here, we are calling the method of parent class with child
//class object.
```

```
//Creating a parent class
class Vehicle{
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike extends Vehicle{
  public static void main(String args[]){
  //creating an instance of child class
  Bike obj = new Bike();
  //calling the method with child class instance
  obj.run();
  }
}
```

[*Test it Now*]

Output:

```
Vehicle is running
```

Example of method overriding

```
//Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

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

Test it Now

Output:

```
Bike is running safely
```

A real example of Java Method Overriding

```
//Java Program to demonstrate the real scenario of Java Method Overriding
//where three classes are overriding the method of a parent class.
//Creating a parent class.
class Bank{
int getRateOfInterest(){return 0;}
}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}

class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
//Test class to create objects and call the methods
class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}
```

Test it Now

Output:

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Covariant Return Type

Simple example of Covariant Return Type

FileName: B1.java

```
class A{
A get(){return this;}
}

class B1 extends A{
@Override
B1 get(){return this;}
void message(){System.out.println("welcome to covariant return type");}

public static void main(String args[]){
new B1().get().message();
}
}
```

Test it Now

Output:

```
welcome to covariant return type
```

FileName: CovariantExample.java

```
class A1
{
   A1 foo()
```

```
    {
        return this;
    }

    void print()
    {
        System.out.println("Inside the class A1");
    }
}


// A2 is the child class of A1
class A2 extends A1
{
    @Override
    A1 foo()
    {
        return this;
    }

    void print()
    {
        System.out.println("Inside the class A2");
    }
}

// A3 is the child class of A2
class A3 extends A2
{
    @Override
    A1 foo()
```

```java
    {
       return this;
    }

    @Override
    void print()
    {
       System.out.println("Inside the class A3");
    }
}

public class CovariantExample
{
    // main method
    public static void main(String argvs[])
    {
      A1 a1 = new A1();

      // this is ok
      a1.foo().print();

      A2 a2 = new A2();

      // we need to do the type casting to make it
      // more clear to reader about the kind of object created
      ((A2)a2.foo()).print();

      A3 a3 = new A3();

      // doing the type casting
      ((A3)a3.foo()).print();

    }
}
```

Output:

```
Inside the class A1
Inside the class A2
Inside the class A3
```

FileName: CovariantExample.java

```
class A1
{
    A1 foo()
    {
        return this;
    }

    void print()
    {
        System.out.println("Inside the class A1");
    }
}


// A2 is the child class of A1
class A2 extends A1
{
    @Override
    A2 foo()
    {
        return this;
    }

```

```java
    void print()
    {
        System.out.println("Inside the class A2");
    }
}

// A3 is the child class of A2
class A3 extends A2
{
    @Override
    A3 foo()
    {
        return this;
    }

    @Override
    void print()
    {
        System.out.println("Inside the class A3");
    }
}

public class CovariantExample
{
    // main method
    public static void main(String argvs[])
    {
       A1 a1 = new A1();

       a1.foo().print();

```

```
        A2 a2 = new A2();

        a2.foo().print();

        A3 a3 = new A3();

        a3.foo().print();

    }
}
```

Output:

```
Inside the class A1
Inside the class A2
Inside the class A3
```

Super Keyword in Java

```
lass Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

Test it Now

Output:

```
black
white
```

---

super can be used to invoke parent class method

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

Test it Now

Output:

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

---

super is used to invoke parent class constructor

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Test it Now

Output:

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

```
}
class Dog extends Animal{
Dog(){
System.out.println("dog is created");
}
}
class TestSuper4{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Test it Now

Output:

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

super example: real use

```
class Person{
int id;
String name;
Person(int id,String name){
```

```
this.id=id;
this.name=name;
}
}
class Emp extends Person{
float salary;
Emp(int id,String name,float salary){
super(id,name);//reusing parent constructor
this.salary=salary;
}
void display(){System.out.println(id+" "+name+" "+salary);}
}
class TestSuper5{
public static void main(String[] args){
Emp e1=new Emp(1,"ankit",45000f);
e1.display();
}}
```

Test it Now

Output:

```
1 ankit 45000
```

Instance initializer block

Example of instance initializer block

Let's see the simple example of instance initializer block that performs initialization.

```
class Bike7{
    int speed;

    Bike7(){System.out.println("speed is "+speed);}

    {speed=100;}

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

```
    Bike7 b1=new Bike7();
    Bike7 b2=new Bike7();
    }
}
```

Test it Now

Output:speed is 100

speed is 100

There are three places in java where you can perform operations:

1. *method*
2. *constructor*
3. *block*

What is invoked first, instance initializer block or constructor?

```
class Bike8{
    int speed;

    Bike8(){System.out.println("constructor is invoked");}

    {System.out.println("instance initializer block invoked");}

    public static void main(String args[]){
    Bike8 b1=new Bike8();
    Bike8 b2=new Bike8();
    }
}
```

Test it Now

```
Output:instance initializer block invoked
constructor is invoked
instance initializer block invoked
```

constructor is invoked

Program of instance initializer block that is invoked after super()

```
class A{
A(){
System.out.println("parent class constructor invoked");
}
}
class B2 extends A{
B2(){
super();
System.out.println("child class constructor invoked");
}

{System.out.println("instance initializer block is invoked");}

public static void main(String args[]){
B2 b=new B2();
}
}
```

[Test it Now]()

```
Output:parent class constructor invoked
instance initializer block is invoked
child class constructor invoked
```

Another example of instance block

```
class A{
A(){
System.out.println("parent class constructor invoked");
}
}

```

```
class B3 extends A{
B3(){
super();
System.out.println("child class constructor invoked");
}

B3(int a){
super();
System.out.println("child class constructor invoked "+a);
}

{System.out.println("instance initializer block is invoked");}

public static void main(String args[]){
B3 b1=new B3();
B3 b2=new B3(10);
}
}
```

Test it Now

```
parent class constructor invoked
instance initializer block is invoked
child class constructor invoked
parent class constructor invoked
instance initializer block is invoked
child class constructor invoked 10
```

Final Keyword In Java

Example of final variable

```
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
```

Test it Now

Output:Compile Time Error

---

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
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();
   }
}
```

Test it Now

Output:Compile Time Error

---

3) Java final class

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

Example of final class

```
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();
  }
}
```

Test it Now

Output:Compile Time Error

Q) Is final method inherited?

Ans) 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();
   }
}
```

Test it Now

Output:running...

Example of blank final variable

```
class Student{
int id;
String name;
final String PAN_CARD_NUMBER;
...
}
```

Que) Can we initialize blank final variable?

Yes, but only in constructor. For example:

```
class Bike10{
  final int speedlimit;//blank final variable

  Bike10(){
  speedlimit=70;
  System.out.println(speedlimit);
  }

  public static void main(String args[]){
    new Bike10();
 }
}
```

Test it Now

Output: 70

---

static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Example of static blank final variable

```
class A{
  static final int data;//static blank final variable
  static{ data=50;}
  public static void main(String args[]){
    System.out.println(A.data);
 }
}
```

Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

```
class Bike11{
 int cube(final int n){
  n=n+2;//can't be changed as n is final
  n*n*n;
 }
 public static void main(String args[]){
   Bike11 b=new Bike11();
   b.cube(5);
 }
}
```

Test it Now

Output: Compile Time Error

Polymorphism in Java

Example of Java Runtime Polymorphism

```
class Bike{
  void run(){System.out.println("running");}
}
class Splendor extends Bike{
  void run(){System.out.println("running safely with 60km");}

  public static void main(String args[]){
    Bike b = new Splendor();//upcasting
    b.run();
  }
}
```

Test it Now

Output:

```
running safely with 60km.
```

Java Runtime Polymorphism Example: Bank

```
class Bank{
```

```java
float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
float getRateOfInterest(){return 9.7f;}
}
class TestPolymorphism{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Test it Now

Output:

```
SBI Rate of Interest: 8.4
ICICI Rate of Interest: 7.3
AXIS Rate of Interest: 9.7
```

Java Runtime Polymorphism Example: Shape

```java
class Shape{
void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle...");}
}
class TestPolymorphism2{
public static void main(String args[]){
Shape s;
s=new Rectangle();
s.draw();
s=new Circle();
s.draw();
s=new Triangle();
s.draw();
}
}
```

[Test it Now](#)

Output:

```
drawing rectangle...
drawing circle...
drawing triangle...
```

Java Runtime Polymorphism Example: Animal

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
```

```
void eat(){System.out.println("eating bread...");}
}
class Cat extends Animal{
void eat(){System.out.println("eating rat...");}
}
class Lion extends Animal{
void eat(){System.out.println("eating meat...");}
}
class TestPolymorphism3{
public static void main(String[] args){
Animal a;
a=new Dog();
a.eat();
a=new Cat();
a.eat();
a=new Lion();
a.eat();
}}
```

Test it Now

Output: eating bread...

```
eating rat...
eating meat...
```

Java Runtime Polymorphism with Data Member

```
class Bike{
 int speedlimit=90;
}
class Honda3 extends Bike{
 int speedlimit=150;

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

```
 Bike obj=new Honda3();
 System.out.println(obj.speedlimit);//90
}
```

Test it Now

Output:

```
90
```

Java Runtime Polymorphism with Multilevel Inheritance

Let's see the simple example of Runtime Polymorphism with multilevel inheritance.

```
class Animal{
void eat(){System.out.println("eating");}
}
class Dog extends Animal{
void eat(){System.out.println("eating fruits");}
}
class BabyDog extends Dog{
void eat(){System.out.println("drinking milk");}
public static void main(String args[]){
Animal a1,a2,a3;
a1=new Animal();
a2=new Dog();
a3=new BabyDog();
a1.eat();
a2.eat();
a3.eat();
}
}
```

Test it Now

Output:

```
eating
eating fruits
```

drinking Milk

Try for Output

```
class Animal{
void eat(){System.out.println("animal is eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("dog is eating...");}
}
class BabyDog1 extends Dog{
public static void main(String args[]){
Animal a=new BabyDog1();
a.eat();
}}
```

[Test it Now]

Output:

Dog is eating

Static Binding and Dynamic Binding

Example of static binding

```
class Dog{
 private void eat(){System.out.println("dog is eating...");}

 public static void main(String args[]){
  Dog d1=new Dog();
  d1.eat();
 }
}
```

Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

Example of dynamic binding

```
class Animal{
 void eat(){System.out.println("animal is eating...");}
}

class Dog extends Animal{
 void eat(){System.out.println("dog is eating...");}

 public static void main(String args[]){
  Animal a=new Dog();
  a.eat();
 }
}
```

Test it Now

Output:dog is eating..

Java instanceof

Simple example of java instanceof

Let's see the simple example of instance operator where it tests the current class.

```
class Simple1{
 public static void main(String args[]){
 Simple1 s=new Simple1();
 System.out.println(s instanceof Simple1);//true
 }
}
```

Test it Now

Output:true

Another example of java instanceof operator

```
class Animal{}
class Dog1 extends Animal{//Dog inherits Animal

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

```
Dog1 d=new Dog1();
System.out.println(d instanceof Animal);//true
}
}
```

[Test it Now]

Output:true

---

instanceof in java with a variable that have null value

If we apply instanceof operator with a variable that have null value, it returns false. Let's see the example given below where we apply instanceof operator with the variable that have null value.

```
class Dog2{
 public static void main(String args[]){
  Dog2 d=null;
  System.out.println(d instanceof Dog2);//false
 }
}
```

[Test it Now]

Output:false

Possibility of downcasting with instanceof

Let's see the example, where downcasting is possible by instanceof operator.

```
class Animal { }

class Dog3 extends Animal {
  static void method(Animal a) {
    if(a instanceof Dog3){
       Dog3 d=(Dog3)a;//downcasting
       System.out.println("ok downcasting performed");
    }
  }

```

```
 public static void main (String [] args) {
  Animal a=new Dog3();
  Dog3.method(a);
 }

}
```

[Test it Now]()

Output:ok downcasting performed

Downcasting without the use of java instanceof

Downcasting can also be performed without the use of instanceof operator as displayed in the following example:

```
class Animal { }
class Dog4 extends Animal {
 static void method(Animal a) {
     Dog4 d=(Dog4)a;//downcasting
     System.out.println("ok downcasting performed");
 }
  public static void main (String [] args) {
  Animal a=new Dog4();
  Dog4.method(a);
 }
}
```

[Test it Now]()

Output:ok downcasting performed

Understanding Real use of instanceof in java

Let's see the real use of instanceof keyword by the example given below.

```
interface Printable{}
class A implements Printable{
public void a(){System.out.println("a method");}
}
```

```
class B implements Printable{
public void b(){System.out.println("b method");}
}

class Call{
void invoke(Printable p){//upcasting
if(p instanceof A){
A a=(A)p;//Downcasting
a.a();
}
if(p instanceof B){
B b=(B)p;//Downcasting
b.b();
}

}
}//end of Call class

class Test4{
public static void main(String args[]){
Printable p=new B();
Call c=new Call();
c.invoke(p);
}
}
```

Test it Now

Output: b method

Abstract class in Java

Example of Abstract class that has an abstract method

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

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

[Test it Now]

```
running safely
```

Understanding the real scenario of Abstract class

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
s.draw();
}
}
```

[Test it Now](#)

```
drawing circle
```

Another example of Abstract class in java

File: TestBank.java

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

[Test it Now](#)

```
Rate of Interest is: 7 %
Rate of Interest is: 8 %
```

Abstract class having constructor, data member and methods

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

File: TestAbstraction2.java

```
//Example of an abstract class that has abstract and non-abstract methods
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

Test it Now

```
bike is created
running safely..
gear changed
```

Rule: If there is an abstract method in a class, that class must be abstract.

```
class Bike12{
abstract void run();
}
```

Test it Now

```
compile time error
```

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
s.draw();
}
}
```

Test it Now

```
drawing circle
```

Another example of Abstract class in java

File: TestBank.java

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

```

```
class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

Test it Now

```
Rate of Interest is: 7 %
Rate of Interest is: 8 %
```

Abstract class having constructor, data member and methods

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

File: TestAbstraction2.java

```
//Example of an abstract class that has abstract and non-abstract methods
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
obj.changeGear();
 }
}
```

Test it Now

```
bike is created
running safely..
gear changed
```

Rule: If there is an abstract method in a class, that class must be abstract.

```
class Bike12{
abstract void run();
}
```

Test it Now

```
compile time error
```

Interface in Java

Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Test it Now

Output:

Hello

Java Interface Example: Drawable

```
//Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

Test it Now

Output:

```
drawing circle
```

Java Interface Example: Bank

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
```

```java
public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}}
```

Test it Now

Output:

```
ROI: 9.15
```

Multiple inheritance in Java by interface

```java
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Test it Now

Output:Hello

Welcome

Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

```
interface Printable{
void print();
}
interface Showable{
void print();
}

class TestInterface3 implements Printable, Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
 }
}
```

Test it Now

Output:

```
Hello
```

Interface inheritance

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

Test it Now

Output:

```
Hello
Welcome
```

Java 8 Default Method in Interface

```
interface Drawable{
void draw();
default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceDefault{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

Test it Now

Output:

```
drawing rectangle
default method
```

Java 8 Static Method in Interface

```
interface Drawable{
void draw();
static int cube(int x){return x*x*x;}
}
```

```
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}

class TestInterfaceStatic{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
System.out.println(Drawable.cube(3));
}}
```

Test it Now

Output:

```
drawing rectangle
27
```

Difference between abstract class and interface

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}

//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}

//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
```

```
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Test it Now

Output:

```
I am a
I am b
I am c
I am d
```

Java Package

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

Example of package that import the packagename.*

```
//save by A.java
```

```
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

Example of package by import package.classname

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

Output:Hello

Example of Subpackage

```
package com.javatpoint.core;
class Simple{
  public static void main(String args[]){
   System.out.println("Hello subpackage");
  }
}
```

```
Compile: javac -d . Simple.java
```

```
Run: java com.javatpoint.core.Simple
```

Output:Hello subpackage

Simple example of private access modifier

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

Example of default access modifier

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

Example of protected access modifier

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

Output:Hello

Example of public access modifier

```
//save by A.java

```

```
package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

Java Access Modifiers with Method Overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

```
class A{
protected void msg(){System.out.println("Hello java");}
}

public class Simple extends A{
void msg(){System.out.println("Hello java");}//C.T.Error
 public static void main(String args[]){
   Simple obj=new Simple();
   obj.msg();
   }
}
```

The default modifier is more restrictive than protected. That is why, there is a compile-time error.

Encapsulation in Java

```
//A Java class which is a fully encapsulated class.
//It has a private data member and getter and setter methods.
package com.javatpoint;
public class Student{
//private data member
private String name;
//getter method for name
public String getName(){
return name;
}
//setter method for name
public void setName(String name){
this.name=name
}
}
```

File: Test.java

```
//A Java class to test the encapsulated class.
package com.javatpoint;
class Test{
public static void main(String[] args){
//creating instance of the encapsulated class
Student s=new Student();
//setting value in the name member
s.setName("vijay");
//getting value of the name member
System.out.println(s.getName());
}
}
```

Compile By: javac -d . Test.java

Run By: java com.javatpoint.Test

Output:

```
Vijay
```

Another Example of Encapsulation in Java

Let's see another example of encapsulation that has only four fields with its setter and getter methods.

File: Account.java

```
//A Account class which is a fully encapsulated class.
//It has a private data member and getter and setter methods.
class Account {
//private data members
private long acc_no;
private String name,email;
private float amount;
//public getter and setter methods
public long getAcc_no() {
    return acc_no;
}
public void setAcc_no(long acc_no) {
    this.acc_no = acc_no;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
public float getAmount() {
    return amount;
}
public void setAmount(float amount) {
    this.amount = amount;
}

}
```

File: TestAccount.java

```
//A Java class to test the encapsulated class Account.
public class TestEncapsulation {
public static void main(String[] args) {
    //creating instance of Account class
    Account acc=new Account();
    //setting values through setter methods
    acc.setAcc_no(7560504000L);
    acc.setName("Sonoo Jaiswal");
    acc.setEmail("sonoojaiswal@javatpoint.com");
    acc.setAmount(500000f);
    //getting values through getter methods
    System.out.println(acc.getAcc_no()+" "+acc.getName()+" "+acc.getEmail()+" "+acc.getAmount());
}
}
```

Test it Now

Output:

```
7560504000 Sonoo Jaiswal sonoojaiswal@javatpoint.com 500000.0
```

Object Cloning in Java

Example of clone() method (Object cloning)

Let's see the simple example of object cloning

```
class Student18 implements Cloneable{
int rollno;
String name;

Student18(int rollno,String name){
this.rollno=rollno;
this.name=name;
}

public Object clone()throws CloneNotSupportedException{
return super.clone();
}

public static void main(String args[]){
try{
Student18 s1=new Student18(101,"amit");

Student18 s2=(Student18)s1.clone();

System.out.println(s1.rollno+" "+s1.name);
System.out.println(s2.rollno+" "+s2.name);

}catch(CloneNotSupportedException c){}

}
}
```

Test it Now

Output:101 amit

101 amit

Java Math Class

Example 1

JavaMathExample1.java

```
public class JavaMathExample1
{
    public static void main(String[] args)
    {
        double x = 28;
        double y = 4;

        // return the maximum of two numbers
        System.out.println("Maximum number of x and y is: " +Math.max(x, y));

        // return the square root of y
        System.out.println("Square root of y is: " + Math.sqrt(y));

        //returns 28 power of 4 i.e. 28*28*28*28
        System.out.println("Power of x and y is: " + Math.pow(x, y));

        // return the logarithm of given value
        System.out.println("Logarithm of x is: " + Math.log(x));
        System.out.println("Logarithm of y is: " + Math.log(y));

        // return the logarithm of given value when base is 10
        System.out.println("log10 of x is: " + Math.log10(x));
        System.out.println("log10 of y is: " + Math.log10(y));

        // return the log of x + 1
        System.out.println("log1p of x is: " +Math.log1p(x));

        // return a power of 2
        System.out.println("exp of a is: " +Math.exp(x));

        // return (a power of 2)-1
        System.out.println("expm1 of a is: " +Math.expm1(x));
    }
}
```

[Test it Now]

Output:

```
Maximum number of x and y is: 28.0
Square root of y is: 2.0
Power of x and y is: 614656.0
Logarithm of x is: 3.332204510175204
Logarithm of y is: 1.3862943611198906
log10 of x is: 1.4471580313422192
log10 of y is: 0.6020599913279624
log1p of x is: 3.367295829986474
exp of a is: 1.446257064291475E12
expm1 of a is: 1.446257064290475E12
```

Example 2

JavaMathExample2.java

```
public class JavaMathExample2
{
    public static void main(String[] args)
    {
        double a = 30;

        // converting values to radian
        double b = Math.toRadians(a);

        // return the trigonometric sine of a
        System.out.println("Sine value of a is: " +Math.sin(a));

        // return the trigonometric cosine value of a
        System.out.println("Cosine value of a is: " +Math.cos(a));

        // return the trigonometric tangent value of a
        System.out.println("Tangent value of a is: " +Math.tan(a));

        // return the trigonometric arc sine of a
        System.out.println("Sine value of a is: " +Math.asin(a));

        // return the trigonometric arc cosine value of a
        System.out.println("Cosine value of a is: " +Math.acos(a));

        // return the trigonometric arc tangent value of a
        System.out.println("Tangent value of a is: " +Math.atan(a));

        // return the hyperbolic sine of a
        System.out.println("Sine value of a is: " +Math.sinh(a));

        // return the hyperbolic cosine value of a
        System.out.println("Cosine value of a is: " +Math.cosh(a));

        // return the hyperbolic tangent value of a
        System.out.println("Tangent value of a is: " +Math.tanh(a));
    }
}
```

Test it Now

Output:

```
Sine value of a is: -0.9880316240928618
```

```
Cosine value of a is: 0.15425144988758405
Tangent value of a is: -6.405331196646276
Sine value of a is: NaN
Cosine value of a is: NaN
Tangent value of a is: 1.5374753309166493
Sine value of a is: 5.343237290762231E12
Cosine value of a is: 5.343237290762231E12
Tangent value of a is: 1.0
```

Filename: MathDemo.java

```java
// Java program for demonstrating the features and functionalities of Java Math class with methods.
public class MathDemo {
    public static void main(String[] args) {
        double x = 28;
        double y = 4;
        // Basic arithmetic operations
        System.out.println("Addition: " + (x + y));
        System.out.println("Subtraction: " + (x - y));
        System.out.println("Multiplication: " + (x * y));
        System.out.println("Division: " + (x / y));
        // Square root
        System.out.println("Square root of " + x + ": " + Math.sqrt(x));
        // Cube root
        System.out.println("Cube root of " + x + ": " + Math.cbrt(x));
        // Power
        System.out.println("Power of " + x + " to " + y + ": " + Math.pow(x, y));
        // Trigonometric functions
        double angle = 45.0;
        double radian = Math.toRadians(angle);
        System.out.println("Sine of " + angle + " degrees: " + Math.sin(radian));
        System.out.println("Cosine of " + angle + " degrees: " + Math.cos(radian));
        System.out.println("Tangent of " + angle + " degrees: " + Math.tan(radian));
        // Rounding
        double value = -123.456;
        System.out.println("Absolute value of " + value + ": " + Math.abs(value));
        System.out.println("Ceil value of " + value + ": " + Math.ceil(value));
        System.out.println("Floor value of " + value + ": " + Math.floor(value));
        System.out.println("Round value of " + value + ": " + Math.round(value));
        // Random numbers
        System.out.println("Random number between 0.0 and 1.0: " + Math.random());
        System.out.println("Random number between 0 and 100: " + (int) (Math.random() * 100));
        // Maximum and minimum
        double[] numbers = {10.5, 20.7, 5.2, 30.9};
        System.out.println("Maximum value: " + Math.max(numbers[0], Math.max(numbers[1], Math.max(numbers[2], numbers[3]))));
        System.out.println("Minimum value: " + Math.min(numbers[0], Math.min(numbers[1], Math.min(numbers[2], numbers[3]))));
        // Exponential and logarithmic functions
        System.out.println("e^" + x + ": " + Math.exp(x));
        System.out.println("Logarithm base 10 of " + x + ": " + Math.log10(x));
        System.out.println("Logarithm base e of " + x + ": " + Math.log(x));
        // Hypotenuse
        double side1 = 3.0;
        double side2 = 4.0;
        System.out.println("Hypotenuse of a right triangle with sides " + side1 + " and " + side2 + ": " + Math.hypot(side1, side2));
        // Trigonometric functions (inverse)
        double sinValue = 0.5;
        System.out.println("Arcsine of " + sinValue + ": " + Math.toDegrees(Math.asin(sinValue)));
        System.out.println("Arccosine of " + sinValue + ": " + Math.toDegrees(Math.acos(sinValue)));
        System.out.println("Arctangent of " + sinValue + ": " + Math.toDegrees(Math.atan(sinValue)));
        // Constants
        System.out.println("Value of PI: " + Math.PI);
        System.out.println("Value of E: " + Math.E);
    }
}
```

Output:

```
Addition: 32.0
Subtraction: 24.0
Multiplication: 112.0
Division: 7.0
Square root of 28.0: 5.291502622129181
Cube root of 28.0: 3.0365889718756627
Power of 28.0 to 4.0: 614656.0
Sine of 45.0 degrees: 0.7071067811865475
Cosine of 45.0 degrees: 0.7071067811865476
Tangent of 45.0 degrees: 0.9999999999999999
Absolute value of -123.456: 123.456
Ceil value of -123.456: -123.0
Floor value of -123.456: -124.0
Round value of -123.456: -123
Random number between 0.0 and 1.0: 0.40493356810101455
Random number between 0 and 100: 61
Maximum value: 30.9
Minimum value: 5.2
e^28.0: 1.446257064291475E12
Logarithm base 10 of 28.0: 1.4471580313422192
Logarithm base e of 28.0: 3.332204510175204
Hypotenuse of a right triangle with sides 3.0 and 4.0: 5.0
Arcsine of 0.5: 30.000000000000004
```

```
Arccosine of 0.5: 60.00000000000001
Arctangent of 0.5: 26.56505117707799
Value of PI: 3.141592653589793
Value of E: 2.718281828459045
```

Filename: JavaMath.java

```java
public class JavaMath {
    public static void main(String[] args) {
        double x = 28.5;
        double y = 4.2;
        // Basic arithmetic operations
        System.out.println("Addition: " + (x + y));
        System.out.println("Subtraction: " + (x - y));
        System.out.println("Multiplication: " + (x * y));
        System.out.println("Division: " + (x / y));
        // Absolute value
        System.out.println("Absolute value of " + x + ": " + Math.abs(x));
        System.out.println("Absolute value of " + y + ": " + Math.abs(y));
        // The Math.abs method returns the absolute value of a number, which is the number itself if it's positive or zero, and its negation if it's negative.
        // Ceiling and floor
        System.out.println("Ceil value of " + x + ": " + Math.ceil(x));
        System.out.println("Floor value of " + x + ": " + Math.floor(x));
        // The Math.ceil method returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.
        // The Math.floor method returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer.
        // Rounding
        System.out.println("Round value of " + x + ": " + Math.round(x));
        System.out.println("Round value of " + y + ": " + Math.round(y));
        // The Math.round method returns the closest long to the argument, with ties rounding to positive infinity.
        // Maximum and minimum
        System.out.println("Maximum of " + x + " and " + y + ": " + Math.max(x, y));
        System.out.println("Minimum of " + x + " and " + y + ": " + Math.min(x, y));
        // The Math.max method returns the greater of two double values.
        // The Math.min method returns the smaller of two double values.
        // Power
        System.out.println("Power of " + x + " to " + y + ": " + Math.pow(x, y));
        // The Math.pow method returns the value of the first argument raised to the power of the second argument.
        // Square root
        System.out.println("Square root of " + x + ": " + Math.sqrt(x));
        System.out.println("Square root of " + y + ": " + Math.sqrt(y));
        // The Math.sqrt method returns the correctly rounded positive square root of a double value.
        // Trigonometric functions
        double angle = 45.0;
        double radians = Math.toRadians(angle);
        System.out.println("Sine of " + angle + " degrees: " + Math.sin(radians));
        System.out.println("Cosine of " + angle + " degrees: " + Math.cos(radians));
        System.out.println("Tangent of " + angle + " degrees: " + Math.tan(radians));
        // The Math.sin, Math.cos, and Math.tan methods return the trigonometric sine, cosine, and tangent of an angle, respectively, given in radians.
        // Logarithmic functions
        System.out.println("Log base 10 of " + x + ": " + Math.log10(x));
        System.out.println("Natural log of " + x + ": " + Math.log(x));
        // The Math.log10 method returns the base 10 logarithm of a double value.
        // The Math.log method returns the natural logarithm (base e) of a double value.
        // Exponential function
        System.out.println("e raised to the power of " + x + ": " + Math.exp(x));
        // The Math.exp method returns Euler's number e raised to the power of a double value.
        // Random number generation
        System.out.println("Random number between 0.0 and 1.0: " + Math.random());
        // The Math.random method returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
        // Constants
        System.out.println("Value of PI: " + Math.PI);
        System.out.println("Value of E: " + Math.E);
        // The Math.PI constant represents the ratio of the circumference of a circle to its diameter, which is approximately 3.14159.
        // The Math.E constant represents Euler's number, the base of the natural logarithm, which is approximately 2.71828.
    }
}
```

Wrapper classes in Java

Wrapper class Example: Primitive to Wrapper

```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}}
```

Output:

```
20 20 20
```

Wrapper class Example: Wrapper to Primitive

```
//Java program to convert object into primitives
//Unboxing example of Integer to int
public class WrapperExample2{
```

```
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}}
```

Output:

```
3 3 3
```

Java Wrapper classes Example

```
//Java Program to convert all primitives into its corresponding
//wrapper objects and vice-versa
public class WrapperExample3{
public static void main(String args[]){
byte b=10;
short s=20;
int i=30;
long l=40;
float f=50.0F;
double d=60.0D;
char c='a';
boolean b2=true;

//Autoboxing: Converting primitives into objects
Byte byteobj=b;
Short shortobj=s;
Integer intobj=i;
Long longobj=l;
Float floatobj=f;
```

```
Double doubleobj=d;
Character charobj=c;
Boolean boolobj=b2;

//Printing objects
System.out.println("---Printing object values---");
System.out.println("Byte object: "+byteobj);
System.out.println("Short object: "+shortobj);
System.out.println("Integer object: "+intobj);
System.out.println("Long object: "+longobj);
System.out.println("Float object: "+floatobj);
System.out.println("Double object: "+doubleobj);
System.out.println("Character object: "+charobj);
System.out.println("Boolean object: "+boolobj);

//Unboxing: Converting Objects to Primitives
byte bytevalue=byteobj;
short shortvalue=shortobj;
int intvalue=intobj;
long longvalue=longobj;
float floatvalue=floatobj;
double doublevalue=doubleobj;
char charvalue=charobj;
boolean boolvalue=boolobj;

//Printing primitives
System.out.println("---Printing primitive values---");
System.out.println("byte value: "+bytevalue);
System.out.println("short value: "+shortvalue);
System.out.println("int value: "+intvalue);
System.out.println("long value: "+longvalue);
System.out.println("float value: "+floatvalue);
System.out.println("double value: "+doublevalue);
System.out.println("char value: "+charvalue);
System.out.println("boolean value: "+boolvalue);
}}
```

Output:

```
---Printing object values---
Byte object: 10
Short object: 20
Integer object: 30
Long object: 40
Float object: 50.0
Double object: 60.0
Character object: a
Boolean object: true
---Printing primitive values---
byte value: 10
short value: 20
int value: 30
long value: 40
float value: 50.0
double value: 60.0
char value: a
boolean value: true
```

Custom Wrapper class in Java

```
//Creating the custom wrapper class
class Javatpoint{
private int i;
Javatpoint(){}
Javatpoint(int i){
```

```
this.i=i;
}
public int getValue(){
return i;
}
public void setValue(int i){
this.i=i;
}
@Override
public String toString() {
  return Integer.toString(i);
}
}
//Testing the custom wrapper class
public class TestJavatpoint{
public static void main(String[] args){
Javatpoint j=new Javatpoint(10);
System.out.println(j);
}}
```

Output:

```
10
```

Call by Value and Call by Reference in Java

Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

```
class Operation{
 int data=50;

 void change(int data){
 data=data+100;//changes will be in the local variable only
 }

public static void main(String args[]){
  Operation op=new Operation();

  System.out.println("before change "+op.data);
  op.change(500);
  System.out.println("after change "+op.data);

 }
}
```

download this example

```
Output:before change 50
after change 50
```

Another Example of call by value in java

```
class Operation2{
 int data=50;

 void change(Operation2 op){
 op.data=op.data+100;//changes will be in the instance variable
 }


 public static void main(String args[]){
  Operation2 op=new Operation2();

  System.out.println("before change "+op.data);
  op.change(op);//passing object
  System.out.println("after change "+op.data);

 }
}
```

download this example

Output:before change 50

after change 150

Java Command Line Arguments

Simple example of command-line argument in java

```
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
```

```
compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo
```

Output: Your first argument is: sonoo

Example of command-line argument that prints all the values

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

for(int i=0;i<args.length;i++)
System.out.println(args[i]);

}
}
```

```
compile by > javac A.java
run by > java A sonoo jaiswal 1 3 abc
```

```
Output: sonoo
jaiswal
1
3
abc
```

Java String

Java String Example

StringExample.java

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by Java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating Java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

[Test it Now]

Output:

```
java
strings
example
```

Immutable String in Java

Testimmutablestring.java

```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
   System.out.println(s);//will print Sachin because strings are immutable objects
 }
}
```

[Test it Now]

Output:

```
Sachin
```

Testimmutablestring1.java

```
class Testimmutablestring1{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s);
 }
}
```

Test it Now

Output:

```
Sachin Tendulkar
```

Java String compare

StringComparisonUsingEqualsMethod.java

```
class StringComparisonUsingEqualsMethod{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   String s4="Saurav";
   System.out.println(s1.equals(s2));//true
   System.out.println(s1.equals(s3));//true
   System.out.println(s1.equals(s4));//false
 }
}
```

Output:

```
true
true
false
```

StringComparisonUsingequalsIgnoreCase.java

```
class  StringComparisonUsingequalsIgnoreCase {
    public static void main(String[] args) {
        String s1 = "Ram";
        String s2 = "rAm";
        // Using equals() method for case-sensitive comparison
        boolean equalsResult = s1.equals(s2);
        System.out.println("Using equals() method: " + equalsResult); // Output: false
        // Using equalsIgnoreCase() method for case-insensitive comparison
        boolean equalsIgnoreCaseResult = s1.equalsIgnoreCase(s2);
        System.out.println("Using equalsIgnoreCase() method: " + equalsIgnoreCaseResult); // Output: true
    }
}
```

Output:

```
false
true
```

By Using == Operator

StringCompare.java

```
class StringCompare {
    public static void main(String[] args) {
        String s1 = "Kohli";
        String s2 = "Kohli";
        String s3 = new String("Kohli");
        System.out.println(s1 == s2);            // true
        System.out.println(s1 == s3);            // false
    }
}
```

Output:

```
true
false
```

StringComparisonUsingComapreto.java

```
class StringComparisonUsingComapreto {
    public static void main(String[] args) {
       String str1 = "Sachin";
        String str2 = "Sachin";
        String str3 = "Ratan";
        System.out.println(str1.compareTo(str2));     // 0
        System.out.println(str1.compareTo(str3));     // 1 (str1 > str3)
        System.out.println(str3.compareTo(str1));     // -1 (str3 < str1)
    }
}
```

Output:

```
0
1
-1
```

Using startsWith() and endsWith() Methods

```
class StringCompare {
    public static void main(String[] args) {
        String str = "String Compare";
        System.out.println(str.startsWith("String")); // true
        System.out.println(str.endsWith("Compare"));  // true
    }
}
```

Output:

```
true
true
```

String Concatenation in Java

TestStringConcatenation1.java

```
class TestStringConcatenation1{
 public static void main(String args[]){
   String s="Sachin"+" Tendulkar";
   System.out.println(s);//Sachin Tendulkar
 }
}
```

Test it Now

Output:

```
Sachin Tendulkar
```

TestStringConcatenation2.java

```java
class TestStringConcatenation2{
 public static void main(String args[]){
   String s=50+30+"Sachin"+40+40;
   System.out.println(s);//80Sachin4040
 }
}
```

Test it Now

Output:

```
80Sachin4040
```

TestStringConcatenation3.java

```java
class TestStringConcatenation3{
 public static void main(String args[]){
   String s1="Sachin ";
   String s2="Tendulkar";
   String s3=s1.concat(s2);
   System.out.println(s3);//Sachin Tendulkar
  }
}
```

Test it Now

Output:

```
Sachin Tendulkar
```

The above Java program, concatenates two String objects s1 and s2 using concat() method and stores the result into s3 object.

String concatenation using StringBuilder class

StrBuilder.java

```java
public class StrBuilder
{
    /* Driver Code */
    public static void main(String args[])
    {
        StringBuilder s1 = new StringBuilder("Hello");    //String 1
        StringBuilder s2 = new StringBuilder(" World");    //String 2
        StringBuilder s = s1.append(s2);   //String 3 to store the result
          System.out.println(s.toString());  //Displays result
    }
}
```

Output:

```
Hello World
```

---

string concatenation using format() method

```
public class StrFormat
{
    /* Driver Code */
    public static void main(String args[])
    {
        String s1 = new String("Hello");    //String 1
        String s2 = new String(" World");    //String 2
        String s = String.format("%s%s",s1,s2);   //String 3 to store the result
          System.out.println(s.toString());  //Displays result
    }
}
```

Output:

```
Hello World
```

String concatenation using String.join() method (Java Version 8+)

StrJoin.java:

```
public class StrJoin
{
    /* Driver Code */
    public static void main(String args[])
    {
        String s1 = new String("Hello");    //String 1
        String s2 = new String(" World");    //String 2
        String s = String.join("",s1,s2);   //String 3 to store the result
          System.out.println(s.toString());  //Displays result
    }
}
```

Output:

```
Hello World
```

String concatenation using StringJoiner class (Java Version 8+)

```
public class StrJoiner
{
  /* Driver Code */
  public static void main(String args[])
  {
    StringJoiner s = new StringJoiner(", ");   //StringeJoiner object
    s.add("Hello");    //String 1
    s.add("World");    //String 2
    System.out.println(s.toString());  //Displays result
  }
}
```

Output:

```
Hello, World
```

String concatenation using Collectors.joining() method (Java (Java Version 8+)

ColJoining.java

```
import java.util.*;
import java.util.stream.Collectors;
public class ColJoining
{
  /* Driver Code */
  public static void main(String args[])
  {
    List<String> liststr = Arrays.asList("abc", "pqr", "xyz"); //List of String array
  String str = liststr.stream().collect(Collectors.joining(", ")); //performs joining operation
```

```
        System.out.println(str.toString()); //Displays result
    }
}
```

Output:

```
abc, pqr, xyz
```

Here, a list of String array is declared. And a String object str stores the result of Collectors.joining() method.

Substring in Java

Example of Java substring() method

TestSubstring.java

```
public class TestSubstring{
 public static void main(String args[]){
 String s="SachinTendulkar";
 System.out.println("Original String: " + s);
 System.out.println("Substring starting from index 6: " +s.substring(6));//Tendulkar
 System.out.println("Substring starting from index 0 to 6: "+s.substring(0,6)); //Sachin
 }
}
```

Output:

```
Original String: SachinTendulkar
Substring starting from index 6: Tendulkar
Substring starting from index 0 to 6: Sachin
```

Using String.split() method:

TestSubstring2.java

```
import java.util.*;

public class TestSubstring2
{
    /* Driver Code */
    public static void main(String args[])
    {
        String text= new String("Hello, My name is Sachin");
        /* Splits the sentence by the delimeter passed as an argument */
        String[] sentences = text.split("\\.");
        System.out.println(Arrays.toString(sentences));
    }
}
```

Output:

```
[Hello, My name is Sachin]
```

Java String Class Methods

Stringoperation1.java

```
public class Stringoperation1
{
public static void main(String ar[])
{
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
System.out.println(s);//Sachin(no change in original)
}
}
```

Test it Now

Output:

```
SACHIN
sachin
Sachin
```

Java String trim() method

The String class trim() method eliminates white spaces before and after the String.

Stringoperation2.java

```
public class Stringoperation2
{
```

```
public static void main(String ar[])
{
String s="  Sachin  ";
System.out.println(s);//  Sachin
System.out.println(s.trim());//Sachin
}
}
```

Test it Now

Output:

```
Sachin
Sachin
```

Java String startsWith() and endsWith() method

The method startsWith() checks whether the String starts with the letters passed as arguments and endsWith() method checks whether the String ends with the letters passed as arguments.

Stringoperation3.java

```
public class Stringoperation3
{
public static void main(String ar[])
{
String s="Sachin";
 System.out.println(s.startsWith("Sa"));//true
 System.out.println(s.endsWith("n"));//true
}
}
```

Test it Now

Output:

```
true
true
```

Java String charAt() Method

The String class charAt() method returns a character at specified index.

Stringoperation4.java

```
public class Stringoperation4
{
public static void main(String ar[])
{
String s="Sachin";
System.out.println(s.charAt(0));//S
System.out.println(s.charAt(3));//h
}
}
```

Test it Now

Output:

```
S
H
```

Java String length() Method

The String class length() method returns length of the specified String.

Stringoperation5.java

```
public class Stringoperation5
{
public static void main(String ar[])
{
String s="Sachin";
System.out.println(s.length());//6
}
}
```

Test it Now

Output:

```
6
```

Java String intern() Method

A pool of strings, initially empty, is maintained privately by the class String.

Stringoperation6.java

```
public class Stringoperation6
{
public static void main(String ar[])
{
String s=new String("Sachin");
String s2=s.intern();
System.out.println(s2);//Sachin
}
}
```

Test it Now

Output:

```
Sachin
```

Java String valueOf() Method

Stringoperation7.java

```
public class Stringoperation7
{
public static void main(String ar[])
{
int a=10;
String s=String.valueOf(a);
System.out.println(s+10);
}
}
```

Output:

```
1010
```

Java String replace() Method

Stringoperation8.java

```
public class Stringoperation8
{
public static void main(String ar[])
{
String s1="Java is a programming language. Java is a platform. Java is an Island.";
String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kav
a"
System.out.println(replaceString);
}
}
```

Output:

```
Kava is a programming language. Kava is a platform. Kava is an Island.
```

Java StringBuffer Class

StringBufferExample.java

```
class StringBufferExample{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

Output:

```
Hello Java
```

---

2) StringBuffer insert() Method

The insert() method inserts the given String with this string at the given position.

StringBufferExample2.java

```
class StringBufferExample2{
public static void main(String args[]){
```

```
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

Output:

```
HJavaello
```

---

3) StringBuffer replace() Method

The replace() method replaces the given String from the specified beginIndex and endIndex.

StringBufferExample3.java

```
class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

Output:

```
HJavalo
```

---

4) StringBuffer delete() Method

The delete() method of the StringBuffer class deletes the String from the specified beginIndex to endIndex.

StringBufferExample4.java

```
class StringBufferExample4{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

Output:

```
Hlo
```

5) StringBuffer reverse() Method

The reverse() method of the StringBuilder class reverses the current String.

StringBufferExample5.java

```
class StringBufferExample5{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

Output:

```
olleH
```

6) StringBuffer capacity() Method

The capacity() method of the StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

StringBufferExample6.java

```
class StringBufferExample6{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
}
}
```

Output:

```
16
16
34
```

7) StringBuffer ensureCapacity() method

StringBufferExample7.java

```
class StringBufferExample7{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
}
}
```

Output:

```
16
16
34
34
70
```

Java StringBuilder Class

StringBuilderExample.java

Advertisement

```
class StringBuilderExample{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

Output:

```
Hello Java
```

---

2) StringBuilder insert() method

The StringBuilder insert() method inserts the given string with this string at the given position.

StringBuilderExample2.java

```
class StringBuilderExample2{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

Output:

```
HJavaello
```

---

3) StringBuilder replace() method

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

StringBuilderExample3.java

```
class StringBuilderExample3{
```

```
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

Output:

```
HJavalo
```

4) StringBuilder delete() method

The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.

StringBuilderExample4.java

```
class StringBuilderExample4{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

Output:

```
Hlo
```

5) StringBuilder reverse() method

The reverse() method of StringBuilder class reverses the current string.

StringBuilderExample5.java

```
class StringBuilderExample5{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
```

```java
}
}
```

Output:

```
olleH
```

---

6) StringBuilder capacity() method

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

StringBuilderExample6.java

```java
class StringBuilderExample6{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("Java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
}
}
```

Output:

Advertisement

```
16
16
34
```

---

7) StringBuilder ensureCapacity() method

StringBuilderExample7.java

```java
class StringBuilderExample7{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("Java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
}
}
```

Output:

```
16
16
34
34
70
```

Difference between String and StringBuffer

ConcatTest.java

```java
public class ConcatTest{
  public static String concatWithString()    {
    String t = "Java";
    for (int i=0; i<10000; i++){
      t = t + "Tpoint";
    }
    return t;
  }
  public static String concatWithStringBuffer(){
    StringBuffer sb = new StringBuffer("Java");
    for (int i=0; i<10000; i++){
      sb.append("Tpoint");
```

```
        }
        return sb.toString();
    }
    public static void main(String[] args){
        long startTime = System.currentTimeMillis();
        concatWithString();
        System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis()-startTime)+"ms");
        startTime = System.currentTimeMillis();
        concatWithStringBuffer();
        System.out.println("Time taken by Concating with  StringBuffer: "+(System.currentTimeMillis()-startTime)+"ms");
    }
}
```

Output:

```
Time taken by Concating with String: 578ms
Time taken by Concating with StringBuffer: 0ms
```

String and StringBuffer HashCode TestInstanceTest.java

```
public class InstanceTest{
    public static void main(String args[]){
        System.out.println("Hashcode test of String:");
        String str="java";
        System.out.println(str.hashCode());
        str=str+"tpoint";
        System.out.println(str.hashCode());

        System.out.println("Hashcode test of StringBuffer:");
        StringBuffer sb=new StringBuffer("java");
        System.out.println(sb.hashCode());
        sb.append("tpoint");
        System.out.println(sb.hashCode());
    }
}
```

Output:

```
Hashcode test of String:
3254818
229541438
Hashcode test of StringBuffer:
118352462
118352462
```

Difference between StringBuffer and StringBuilder

StringBuffer Example

```
//Java Program to demonstrate the use of StringBuffer class.
public class BufferTest{
    public static void main(String[] args){
        StringBuffer buffer=new StringBuffer("hello");
        buffer.append("java");
        System.out.println(buffer);
    }
}
```

Output:

```
Hellojava
```

StringBuilder Example

BuilderTest.java

```
//Java Program to demonstrate the use of StringBuilder class.
public class BuilderTest{
    public static void main(String[] args){
        StringBuilder builder=new StringBuilder("hello");
        builder.append("java");
        System.out.println(builder);
    }
}
```

Output:

```
Hellojava
```

Performance Test of StringBuffer and StringBuilder

Let's see the code to check the performance of StringBuffer and StringBuilder classes.

ConcatTest.java

```
//Java Program to demonstrate the performance of StringBuffer and StringBuilder classes.

public class ConcatTest{
    public static void main(String[] args){
        long startTime = System.currentTimeMillis();
        StringBuffer sb = new StringBuffer("Java");
        for (int i=0; i<10000; i++){
            sb.append("Tpoint");
        }
        System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms");
        startTime = System.currentTimeMillis();
        StringBuilder sb2 = new StringBuilder("Java");
        for (int i=0; i<10000; i++){
            sb2.append("Tpoint");
        }
        System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");
    }
}
```

Output:

```
Time taken by StringBuffer: 16ms
Time taken by StringBuilder: 0ms
```

How to create Immutable class?

ImmutableDemo.java

```
public final class Employee
{
final String pancardNumber;
public Employee(String pancardNumber)
{
this.pancardNumber=pancardNumber;
}
public String getPancardNumber(){
return pancardNumber;
}
}
public class ImmutableDemo
{
public static void main(String ar[])
{
Employee e = new Employee("ABC123");
String s1 = e.getPancardNumber();
System.out.println("Pancard Number: " + s1);
}
}
```

Output:

```
Pancard Number: ABC123
```

Java toString() Method

Student.java

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
```

```
this.rollno=rollno;
this.name=name;
this.city=city;
}

public static void main(String args[]){
  Student s1=new Student(101,"Raj","lucknow");
  Student s2=new Student(102,"Vijay","ghaziabad");

  System.out.println(s1);//compiler writes here s1.toString()
  System.out.println(s2);//compiler writes here s2.toString()
}
}
```

Output:

```
Student@1fee6fc
Student@1eed786
```

Example of Java toString() method

Student.java

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }

 public String toString(){//overriding the toString() method
```

```
  return rollno+" "+name+" "+city;
 }
 public static void main(String args[]){
   Student s1=new Student(101,"Raj","lucknow");
   Student s2=new Student(102,"Vijay","ghaziabad");

   System.out.println(s1);//compiler writes here s1.toString()
   System.out.println(s2);//compiler writes here s2.toString()
 }
}
```

Output:

```
101 Raj lucknow
102 Vijay ghaziabad
```

StringTokenizer in Java

Example of StringTokenizer Class

Simple.java

```
import java.util.StringTokenizer;
public class Simple{
 public static void main(String args[]){
   StringTokenizer st = new StringTokenizer("my name is khan"," ");
     while (st.hasMoreTokens()) {
         System.out.println(st.nextToken());
     }
   }
}
```

Output:

```
my
name
is
khan
```

Example of StringTokenizer.nextToken(String delim) Method

Test.java

```java
import java.util.*;

public class Test {
  public static void main(String[] args) {
     StringTokenizer st = new StringTokenizer("my,name,is,khan");

    // printing next token
    System.out.println("Next token is : " + st.nextToken(","));
  }
}
```

Output:

```
Next token is : my
```

StringTokenizer1.java

```java
import java.util.StringTokenizer;
public class StringTokenizer1
{
 /* Driver Code */
 public static void main(String args[])
 {
   /* StringTokenizer object */
   StringTokenizer st = new StringTokenizer("Demonstrating methods from StringTokenizer class"," ");
    /* Checks if the String has any more tokens */
    while (st.hasMoreTokens())
    {
        System.out.println(st.nextToken());
    }
 }
}
```

Output:

Demonstrating

methods

from

StringTokenizer

class

Example of StringTokenizer.hasMoreElements() Method

StringTokenizer2.java

```
import java.util.StringTokenizer;
public class StringTokenizer2
{
 public static void main(String args[])
 {
   StringTokenizer st = new StringTokenizer("Hello everyone I am a Java developer"," ");
     while (st.hasMoreElements())
     {
        System.out.println(st.nextToken());
     }
 }
}
```

Output:

```
Hello
everyone
I
am
a
Java
developer
```

Example of StringTokenizer.nextElement() Method

StringTokenizer3.java

```
import java.util.StringTokenizer;
public class StringTokenizer3
```

```
{
 /* Driver Code */
 public static void main(String args[])
 {
   /* StringTokenizer object */
   StringTokenizer st = new StringTokenizer("Hello Everyone Have a nice day"," ");
     /* Checks if the String has any more tokens */
     while (st.hasMoreTokens())
     {
         /* Prints the elements from the String */
         System.out.println(st.nextElement());
     }
 }
}
```

Output:

```
Hello
Everyone
Have
a
nice
day
```

Example of StringTokenizer.countTokens() Method

StringTokenizer4.java

```
import java.util.StringTokenizer;
public class StringTokenizer3
{
 /* Driver Code */
 public static void main(String args[])
 {
   /* StringTokenizer object */
   StringTokenizer st = new StringTokenizer("Hello Everyone Have a nice day"," ");
```

```
        /* Prints the number of tokens present in the String */
        System.out.println("Total number of Tokens: "+st.countTokens());
 }
}
```

Output:

```
Total number of Tokens: 6
```

How to Reverse String in Java?

1) By Using StringBuilder / StringBuffer Class

File: StringFormatter.java

```
public class StringFormatter {
public static String reverseString(String str){
    StringBuilder sb=new StringBuilder(str);
    sb.reverse();
    return sb.toString();
}
}
```

File: TestStringFormatter.java

```
public class TestStringFormatter {
public static void main(String[] args) {
    System.out.println(StringFormatter.reverseString("my name is khan"));
    System.out.println(StringFormatter.reverseString("I am sonoo jaiswal"));
    }
}
```

Output:

```
nahk si eman ym
lawsiaj oonos ma I
```

---

2) By Using Reverse Iteration

File: StringFormatter.java

```
public class StringFormatter {
public static String reverseString(String str){
    char ch[]=str.toCharArray();
    String rev="";
    for(int i=ch.length-1;i>=0;i--){
        rev+=ch[i];
    }
    return rev;
}
}
```

File: TestStringFormatter.java

```
public class TestStringFormatter {
public static void main(String[] args) {
    System.out.println(StringFormatter.reverseString("my name is khan"));
    System.out.println(StringFormatter.reverseString("I am sonoo jaiswal"));
    }
}
```

Output:

```
nahk si eman ym
lawsiaj oonos ma I
```

Filename: StringReversal.java

```
public class StringReversal {
    // Approach 1: Using StringBuilder
    public static String reverseWithStringBuilder(String str) {
        // Step 1: Create a new StringBuilder object
        StringBuilder stringBuilder = new StringBuilder();
        // Step 2: Iterate over each character in the input string in reverse order
        for (int i = str.length() - 1; i >= 0; i--) {
            // Step 3: Append the current character to the StringBuilder
            stringBuilder.append(str.charAt(i));
        }
        // Step 4: Convert the StringBuilder object to a string and return it
        String reversedString = stringBuilder.toString();
        return reversedString;
    }
    // Approach 2: Using a char array
    public static String reverseWithCharArray(String str) {
        // Step 1: Convert the input string to a char array
        char[] charArray = str.toCharArray();
        // Step 2: Create an empty char array with the same length as the input string
        char[] reversedCharArray = new char[str.length()];
        // Step 3: Iterate over each character in the input char array
        for (int i = 0; i < str.length(); i++) {
            // Step 4: Copy each character to the reversed char array in reverse order
            reversedCharArray[i] = charArray[str.length() - 1 - i];
        }
        // Step 5: Convert the reversed char array to a string and return it
        String reversedString = new String(reversedCharArray);
        return reversedString;
    }
    // Approach 3: Using recursion
    public static String reverseWithRecursion(String str) {
        // Step 1: Base case - if the input string is empty, return it
        if (str.isEmpty()) {
            return str;
        }
        // Step 2: Recursive step - reverse the substring from the second character onwards and append the first character
        String reversedString = reverseWithRecursion(str.substring(1)) + str.charAt(0);
        return reversedString;
    }
    public static void main(String[] args) {
        // Step 1: Input string
        String input = "Hello, world!";
```

```java
        // Step 2: Print the original string
        System.out.println("Original: " + input);       // Step 3: Reverse the string using StringBuilder and print the result
        System.out.println("Reversed with StringBuilder: " + reverseWithStringBuilder(input));

        // Step 4: Reverse the string using char array and print the result
        System.out.println("Reversed with char array: " + reverseWithCharArray(input));
        // Step 5: Reverse the string using recursion and print the result
        System.out.println("Reversed with recursion: " + reverseWithRecursion(input));
    }
}
```

Output:

```
Original: Hello, world!
Reversed with StringBuilder: !dlrow ,olleH
Reversed with char array: !dlrow ,olleH
Reversed with recursion: !dlrow ,olleH
```

Example 2:

Filename: StringReversalExample.java

```java
public class StringReversalExample {
    // Approach 1: Using StringBuilder
    public static String reverseWithStringBuilder(String str) {
        // Create a StringBuilder object with the input string
        StringBuilder stringBuilder = new StringBuilder(str);
        // Use the reverse() method of StringBuilder to reverse the string
        stringBuilder.reverse();
        // Convert the reversed StringBuilder object back to a string and return it
        return stringBuilder.toString();
    }
    // Approach 2: Using a char array
    public static String reverseWithCharArray(String str) {
        // Convert the input string to a char array
        char[] charArray = str.toCharArray();
        // Initialize two pointers, one pointing to the start of the char array and the other to the end
        int start = 0;
        int end = str.length() - 1;
        // Iterate until the two pointers meet in the middle
        while (start < end) {
            // Swap the characters at the start and end positions
            char temp = charArray[start];
            charArray[start] = charArray[end];
            charArray[end] = temp;
            start++;
            end--;
        }
        // Convert the reversed char array back to a string and return it
        return new String(charArray);
    }
    // Approach 3: Using recursion
    public static String reverseWithRecursion(String str) {
        // Base case: if the input string is empty, return it
        if (str.isEmpty()) {
            return str;
        }
        // Recursive step: reverse the substring starting from the second character and concatenate the first character
        return reverseWithRecursion(str.substring(1)) + str.charAt(0);
    }
    public static void main(String[] args) {
        // Input string
        String input = "Java Programming";
        // Print the original string
        System.out.println("Original: " + input);
        // Reverse the string using StringBuilder and print the result
        System.out.println("Reversed with StringBuilder: " + reverseWithStringBuilder(input));

        // Reverse the string using char array and print the result
        System.out.println("Reversed with char array: " + reverseWithCharArray(input));
        // Reverse the string using recursion and print the result
        System.out.println("Reversed with recursion: " + reverseWithRecursion(input));
    }
}
```

Output:

```
Original: Java Programming
Reversed with StringBuilder: gnimmargorP avaJ
Reversed with char array: gnimmargorP avaJ
Reversed with recursion: gnimmargorP avaJ
```

Java String charAt()

FileName: CharAtExample.java

```
public class CharAtExample{
public static void main(String args[]){
String name="javatpoint";
char ch=name.charAt(4);//returns the char value at the 4th index
System.out.println(ch);
}}
```

[Test it Now]()

Output:

```
t
```

FileName: CharAtExample.java

```
public class CharAtExample{
public static void main(String args[]){
String name="javatpoint";
char ch=name.charAt(10);//returns the char value at the 10th index
System.out.println(ch);
}}
```

Output:

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException:
String index out of range: 10
at java.lang.String.charAt(String.java:658)
at CharAtExample.main(CharAtExample.java:4)
```

Accessing First and Last Character by Using the charAt() Method

Let's see a simple example where we are accessing first and last character from the provided string.

FileName: CharAtExample3.java

```
public class CharAtExample3 {
    public static void main(String[] args) {
    String str = "Welcome to Javatpoint portal";
    int strLength = str.length();
    // Fetching first character
    System.out.println("Character at 0 index is: "+ str.charAt(0));
    // The last Character is present at the string length-1 index
    System.out.println("Character at last index is: "+ str.charAt(strLength-1));
    }
}
```

Output:

```
Character at 0 index is: W
Character at last index is: l
```

Print Characters Presented at Odd Positions by Using the charAt() Method

Let's see an example where we are accessing all the elements present at odd index.

FileName: CharAtExample4.java

```
public class CharAtExample4 {
    public static void main(String[] args) {
        String str = "Welcome to Javatpoint portal";
        for (int i=0; i<=str.length()-1; i++) {
            if(i%2!=0) {
                System.out.println("Char at "+i+" place "+str.charAt(i));
            }
        }
    }
}
```

Output:

```
Char at 1 place e
Char at 3 place c
Char at 5 place m
Char at 7 place
Char at 9 place o
Char at 11 place J
Char at 13 place v
Char at 15 place t
Char at 17 place o
Char at 19 place n
Char at 21 place
Char at 23 place o
Char at 25 place t
Char at 27 place l
```

The position such as 7 and 21 denotes the space.

Counting Frequency of a character in a String by Using the charAt() Method

Let's see an example in which we are counting frequency of a character in the given string.

FileName: CharAtExample5.java

```
public class CharAtExample5 {
    public static void main(String[] args) {
        String str = "Welcome to Javatpoint portal";
        int count = 0;
        for (int i=0; i<=str.length()-1; i++) {
            if(str.charAt(i) == 't') {
                count++;
            }
        }
        System.out.println("Frequency of t is: "+count);
    }
}
```

Output:

```
Frequency of t is: 4
```

Counting the Number of Vowels in a String by Using the chatAt() Method

Let's see an example where we are counting the number of vowels present in a string with the help of the charAt() method.

FileName: CharAtExample6.java

```
// import statement
import java.util.*;

public class CharAtExample6
{
ArrayList<Character> al;

// constructor for creating and
// assigning values to the ArrayList al
CharAtExample6()
{
al = new ArrayList<Character>();
al.add('A'); al.add('E');
al.add('a'); al.add('e');
al.add('I'); al.add('O');
al.add('i'); al.add('o');
al.add('U'); al.add('u');
}

// a method that checks whether the character c is a vowel or not
private boolean isVowel(char c)
{
for(int i = 0; i < al.size(); i++)
{
if(c == al.get(i))
{
return true;
}
}
return false;
}
// a method that calculates vowels in the String s
public int countVowels(String s)
{
int countVowel = 0; // store total number of vowels
int size = s.length(); // size of string
for(int j = 0; j < size; j++)
{
char c = s.charAt(j);
if(isVowel(c))
{
// vowel found!
// increase the count by 1
countVowel = countVowel + 1;
}
}

```

```
return countVowel;
}

// main method
public static void main(String argvs[])
{
// creating an object of the class CharAtExample6
CharAtExample6 obj = new CharAtExample6();

String str = "Javatpoint is a great site for learning Java.";

int noOfVowel = obj.countVowels(str);

System.out.println("String: " + str);

System.out.println("Total number of vowels in the string are: "+ noOfVowel + "\n");

str = "One apple in a day keeps doctor away.";

System.out.println("String: " + str);

noOfVowel = obj.countVowels(str);

System.out.println("Total number of vowels in the string are: "+ noOfVowel);
}
}
```

Output:

```
String: Javatpoint is a great site for learning Java.
Total number of vowels in the string are: 16


String: One apple in a day keeps doctor away.
```

```
Total number of vowels in the string are: 13
```

Java String compareTo()

Java String compareTo() Method Example

CompareTo.java

```java
public class CompareTo{
public static void main(String args[]){
String s1="hello";
String s2="hello";
String s3="meklo";
String s4="hemlo";
String s5="flag";
System.out.println(s1.compareTo(s2));//0 because both are equal
System.out.println(s1.compareTo(s3));//-5 because "h" is 5 times lower than "m"
System.out.println(s1.compareTo(s4));//-1 because "l" is 1 times lower than "m"
System.out.println(s1.compareTo(s5));//2 because "h" is 2 times greater than "f"
}}
```

Test it Now

Output:

```
0
-5
-1
2
```

CompareToExample.java

```java
public class CompareToExample {
public static void main(String args[]){
String s1="hello";
String s2="";
String s3="me";
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s3));
}}
```

Output:

```
5
-2
```

CompareToExample.java

```
public class CompareToExample3
{
// main method
public static void main(String argvs[])
{

// input string in uppercase
String st1 = new String("INDIA IS MY COUNTRY");

// input string in lowercase
String st2 = new String("india is my country");

System.out.println(st1.compareTo(st2));
}
}
```

Output:

```
-32
```

CompareToExample.java

```
// import statement
import java.util.*;

class Players
{

private String name;

// constructor of the class
public Players(String str)
{
name = str;
}

}

public class CompareToExample4
{

// main method
public static void main(String[] args)
{

Players ronaldo = new Players("Ronaldo");
Players sachin = new Players("Sachin");
Players messi = new Players("Messi");
ArrayList<players> al = new ArrayList<>();

al.add(ronaldo);
al.add(sachin);
al.add(messi);

// performing binary search on the list al
Collections.binarySearch(al, "Sehwag", null);
}

}
</players>
```

Output:

```
Exception in thread "main" java.lang.ClassCastException: class Players cannot be cast to class java.lang.Comparable (Players is in the unnamed module of loader 'app'; java.lang.Comparable is in module java.base of loader 'bootstrap')
```

CompareTo.java

p>

```java
public class CompareTo
{
   public static void main(String[] args)
   {
      String str = null;

      // Invoking the compareTo method on a null object leads to NullPointerException
      int result = str.compareTo("India is my country.");

      System.out.println(result);
   }
}
```

Output:

```
Exception in thread "main" java.lang.NullPointerException
at CompareTo.main(CompareTo.java:8)
```

Java String concat

Java String concat() method example

FileName: ConcatExample.java

```java
public class ConcatExample{
public static void main(String args[]){
String s1="java string";
// The string s1 does not get changed, even though it is invoking the method
// concat(), as it is immutable. Therefore, the explicit assignment is required here.
s1.concat("is immutable");
System.out.println(s1);
s1=s1.concat(" is immutable so assign it explicitly");
System.out.println(s1);
}}
```

Test it Now

Output:

```
java string
java string is immutable so assign it explicitly
```

Java String concat() Method Example 2

Let's see an example where we are concatenating multiple string objects.

FileName: ConcatExample2.java

```
public class ConcatExample2 {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "Javatpoint";
        String str3 = "Reader";
        // Concatenating one string
        String str4 = str1.concat(str2);
        System.out.println(str4);
        // Concatenating multiple strings
        String str5 = str1.concat(str2).concat(str3);
        System.out.println(str5);
    }
}
```

Output:

```
HelloJavatpoint
HelloJavatpointReader
```

FileName: ConcatExample3.java

```
public class ConcatExample3 {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "Javatpoint";
        String str3 = "Reader";
        // Concatenating Space among strings
        String str4 = str1.concat(" ").concat(str2).concat(" ").concat(str3);
        System.out.println(str4);
        // Concatenating Special Chars
        String str5 = str1.concat("!!!");
        System.out.println(str5);
        String str6 = str1.concat("@").concat(str2);
        System.out.println(str6);
    }
}
```

Output:

```
Hello Javatpoint Reader
Hello!!!
<a href="/cdn-cgi/l/email-protection" class="__cf_email__" data-cfemail="afe7cac3c3c0efe5ced9cedbdfc0c6c1db">[email�protected]</a>
```

Java String concat() Method Example 4

FileName: ConcatExample4.java

```
// A Java program that shows how to add
// a string at the beginning of another string
public class ConcatExample4
{
// main method
public static void main(String argvs[])
{
String str = "Country";

// we have added the string "India is my" before the String str;
// Also, observe that a string literal can also invoke the concat() method
String s = "India is my ".concat(str);

// displaying the string
System.out.println(s);

}
}
```

Output:

```
India is my Country
```

Java String.contains() Method

Java String.contains() Method Example

File Name: ContainsExample.java

```
class ContainsExample{
public static void main(String args[]){
String name="what do you know about me";
System.out.println(name.contains("do you know"));
System.out.println(name.contains("about"));
System.out.println(name.contains("hello"));
}}
```

[*Test it Now*]

Output:

```
true
true
false
```

Java String.contains() Method Example 2

The contains() method searches case-sensitive char sequence. If the argument is not case sensitive, it returns false. Let's see an example.

FileName: ContainsExample2.java

```
public class ContainsExample2 {
    public static void main(String[] args) {
        String str = "Hello Javatpoint readers";
        boolean isContains = str.contains("Javatpoint");
        System.out.println(isContains);
        // Case Sensitive
        System.out.println(str.contains("javatpoint")); // false
    }
}
```

Output:

```
true
false
```

Java String.contains() Method Example 3

The contains() method is helpful to find a char-sequence in the string. We can use it in the control structure to produce the search-based result. Let's see an example.

FileName: ContainsExample3.java

```
public class ContainsExample3 {
    public static void main(String[] args) {
        String str = "To learn Java visit Javatpoint.com";
        if(str.contains("Javatpoint.com")) {
            System.out.println("This string contains javatpoint.com");
        }else
            System.out.println("Result not found");
    }
}
```

Output:

```
This string contains javatpoint.com
```

Java String.contains() Method Example 4

The contains() method raises the NullPointerException when one passes null in the parameter of the method. The following example shows the same.

FileName: ContainsExample4.java

```
public class ContainsExample4
{
// main method
public static void main(String argvs[])
{
String str = "Welcome to JavaTpoint!";

// comparing a string to null
if(str.contains(null))
{
System.out.println("Inside the if block");
}
else
{
System.out.println("Inside the else block");
}

}
}
```

Output:

```
Exception in thread "main" java.lang.NullPointerException
at java.base/java.lang.String.contains(String.java:2036)
at ContainsExample4.main(ContainsExample4.java:9)
```

Java String endsWith()

Java String endsWith() Method Example

FileName: EndsWithExample.java

```
public class EndsWithExample{
public static void main(String args[]){
String s1="java by javatpoint";
System.out.println(s1.endsWith("t"));
System.out.println(s1.endsWith("point"));
}}
```

Test it Now

Output:

```
true
```

```
true
false
String ends with .com
```

Java String endsWith() Method Example 3

The endsWith() method takes care of the case sensitiveness of the characters present in a string. The following program shows the same.

FileName: EndsWithExample3.java

```
public class EndsWithExample3
{
// main method
public static void main(String argvs[])
{
String str = "Welcome to JavaTpoint";

// the result of the following display statement
// shows endsWith() considers the case sensitiveness of
// the charaters of a string
System.out.println(str.endsWith("javaTpoint")); // false because J and j are different
System.out.println(str.endsWith("Javatpoint")); // false because T and t are different
System.out.println(str.endsWith("JavaTpoint")); // true because every character is same

}
}
```

Output:

```
false
false
true
false
```

Java String endsWith() Method Example 4

FileName: EndsWithExample4.java

```
public class EndsWithExample4
{
// main method
public static void main(String argvs[])
{
String str = "Welcome to JavaTpoint";

// prints true
System.out.println(str.endsWith(""));

// prints false as there is no white space after the string
System.out.println(str.endsWith(" "));

}
}
```

Output:

```
true
false
```

Java String endsWith() Method Example 5

The endsWith() method raises the NullPointerException if one passes null in the parameter of the method. The following example shows the same.

FileName: EndsWithExample5.java

```
public class EndsWithExample5
{
// main method
public static void main(String argvs[])
{
String str = "Welcome to JavaTpoint!";

// invoking the method endsWith with the parameter as null
if(str.endsWith(null))
{
System.out.println("Inside the if block");
}
else
{
System.out.println("Inside the else block");
}

}
}
```

Output:

```
Exception in thread "main" java.lang.NullPointerException
at java.base/java.lang.String.endsWith(String.java:1485)
at EndsWithExample5.main(EndsWithExample5.java:9)
```

Java String endsWith() Method Example 6

A String literal can also call the endsWith() method. The following program shows the same.

FileName: EndsWithExample6.java

```
public class EndsWithExample6
{
// main method
public static void main(String argvs[])
{
// invoking the method endsWith() using the string literal
if("Welcome to JavaTpoint".endsWith(""))
{
System.out.println("Inside the if block");
}
else
{
System.out.println("Inside the else block");
}
// invoking the method endsWith() using the string literal
if("Welcome to JavaTpoint".endsWith("point"))
{
System.out.println("Inside the if block");
}
else
{
System.out.println("Inside the else block");
}
// invoking the method endsWith() using the string literal
if("Welcome to JavaTpoint".endsWith("xyz"))
{
System.out.println("Inside the if block");
}
else
{
System.out.println("Inside the else block");
}
}
}
```

Output:

```
Inside the if block
Inside the if block
Inside the else block
```

Java String equals()

FileName: EqualsExample.java

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

```
String s1="javatpoint";
String s2="javatpoint";
String s3="JAVATPOINT";
String s4="python";
System.out.println(s1.equals(s2));//true because content and case is same
System.out.println(s1.equals(s3));//false because case is not same
System.out.println(s1.equals(s4));//false because content is not same
}}
```

[Test it Now]()

Output:

```
true
false
false
```

Java String equals() Method Example 2

```
public class EqualsExample2 {
    public static void main(String[] args) {
        String s1 = "javatpoint";
        String s2 = "javatpoint";
        String s3 = "Javatpoint";
        System.out.println(s1.equals(s2)); // True because content is same
        if (s1.equals(s3)) {
            System.out.println("both strings are equal");
        }else System.out.println("both strings are unequal");
    }
}
```

Output:

```
true
both strings are unequal
```

FileName: EqualsExample3.java

```
import java.util.ArrayList;
public class EqualsExample3 {
    public static void main(String[] args) {
        String str1 = "Mukesh";
        ArrayList<string> list = new ArrayList<>();
        list.add("Ravi");
        list.add("Mukesh");
        list.add("Ramesh");
        list.add("Ajay");
        for (String str : list) {
            if (str.equals(str1)) {
                System.out.println("Mukesh is present");
            }
        }
    }
}
</string>
```

Output:

```
Mukesh is present
```

FileName: EqualsExample4.java

```
public class EqualsExample4
{
// main method
public static void main(String argvs[])
{
// Strings
String str = "a";
String str1 = "123";
String str2 = "45.89";
String str3 = "false";
Character c = new Character('a');
Integer i = new Integer(123);
Float f = new Float(45.89);
Boolean b = new Boolean(false);
// reference of the Character object is passed
System.out.println(str.equals(c));
// reference of the Integer object is passed
System.out.println(str1.equals(i));
// reference of the Float object is passed
System.out.println(str2.equals(f));
// reference of the Boolean object is passed
System.out.println(str3.equals(b));
// the above print statements show a false value because
// we are comparing a String with different data types
// To achieve the true value, we have to convert
// the different data types into the string using the toString() method
System.out.println(str.equals(c.toString()));
System.out.println(str1.equals(i.toString()));
System.out.println(str2.equals(f.toString()));
System.out.println(str3.equals(b.toString()));
}
}
```

Output:

```
false
false
false
false
true
true
true
true
```

Java String equalsIgnoreCase()

Java String equalsIgnoreCase() Method Example

FileName: EqualsIgnoreCaseExample.java

```
public class EqualsIgnoreCaseExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="javatpoint";
String s3="JAVATPOINT";
String s4="python";
System.out.println(s1.equalsIgnoreCase(s2));//true because content and case both are same
System.out.println(s1.equalsIgnoreCase(s3));//true because case is ignored
System.out.println(s1.equalsIgnoreCase(s4));//false because content is not same
}}
```

[Test it Now]

Output:

```
true
true
false
```

Java String equalsIgnoreCase() Method Example 2

FileName: EqualsIgnoreCaseExample2.java

```
import java.util.ArrayList;
public class EqualsIgnoreCaseExample2 {
    public static void main(String[] args) {
        String str1 = "Mukesh Kumar";
        ArrayList<String> list = new ArrayList<>();
        list.add("Mohan");
        list.add("Mukesh");
        list.add("RAVI");
        list.add("MuKesH kuMar");
        list.add("Suresh");
        for (String str : list) {
            if (str.equalsIgnoreCase(str1)) {
                System.out.println("Mukesh kumar is present");
            }
        }
    }
}
```

Output:

```
Mukesh kumar is present
```

Java String format()

Java String format() method example

```
public class FormatExample{
public static void main(String args[]){
String name="sonoo";
String sf1=String.format("name is %s",name);
String sf2=String.format("value is %f",32.33434);
String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional part filling with 0

System.out.println(sf1);
System.out.println(sf2);
System.out.println(sf3);
}}
```

Test it Now

```
name is sonoo
value is 32.334340
value is 32.334340000000
```

Java String format() Method Example 2

This method supports various data types and formats them into a string type. Let us see an example.

```
public class FormatExample2 {
    public static void main(String[] args) {
        String str1 = String.format("%d", 101);          // Integer value
        String str2 = String.format("%s", "Amar Singh"); // String value
        String str3 = String.format("%f", 101.00);      // Float value
        String str4 = String.format("%x", 101);         // Hexadecimal value
        String str5 = String.format("%c", 'c');         // Char value
        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
        System.out.println(str5);
    }

}
```

[Test it Now]

```
101
Amar Singh
101.000000
65
c
```

Java String format() Method Example 3

Apart from formatting, we can set width, padding etc. of any value. Let us see an example where we are setting width and padding for an integer value.

```
public class FormatExample3 {
    public static void main(String[] args) {
        String str1 = String.format("%d", 101);
        String str2 = String.format("|%10d|", 101);  // Specifying length of integer
        String str3 = String.format("|%-10d|", 101); // Left-
justifying within the specified width
        String str4 = String.format("|% d|", 101);
        String str5 = String.format("|%010d|", 101); // Filling with zeroes
        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
        System.out.println(str5);
    }
}
```

Test it Now

```
101
| 101|
|101 |
| 101|
|0000000101|
```

Java String getBytes()

FileName: StringGetBytesExample.java

```
public class StringGetBytesExample{
public static void main(String args[]){
String s1="ABCDEFG";
byte[] barr=s1.getBytes();
for(int i=0;i<barr.length;i++){
System.out.println(barr[i]);
}
}}
```

Test it Now

Output:

```
65
66
67
68
69
70
71
```

Java String class getBytes() Method Example 2

FileName: StringGetBytesExample2.java

```
public class StringGetBytesExample2 {
    public static void main(String[] args) {
        String s1 = "ABCDEFG";
        byte[] barr = s1.getBytes();
        for(int i=0;i<barr.length;i++){
            System.out.println(barr[i]);
        }
        // Getting string back
        String s2 = new String(barr);
        System.out.println(s2);
    }
}
```

Test it Now

Output:

```
65
66
67
68
69
70
71
ABCDEFG
```

Java String class getBytes() Method Example 3

The following example shows the encoding into a different charset.

FileName: StringGetBytesExample3.java

```
// Import statement
import java.io.*;

public class StringGetBytesExample3
{
```

```
// main method
public static void main(String argvs[])
{
// input string
String str = "Welcome to JavaTpoint.";
System.out.println("The input String is : ");
System.out.println(str + "\n");

// inside try block encoding is
// being done using different charsets
try
{
16 - bit UCS Transformation format
byte[] byteArr = str.getBytes("UTF-16");
System.out.println("After converted into UTF-16 the String is : ");

for (int j = 0; j < byteArr.length; j++)
{
System.out.print(byteArr[j]);
}

System.out.println("\n");

// Big Endian byte order, 16 - bit UCS Transformation format
byte[] byteArr1 = str.getBytes("UTF-16BE");
System.out.println("After converted into UTF-16BE the String is : ");

for (int j = 0; j < byteArr1.length; j++)
{
System.out.print(byteArr1[j]);
}

System.out.println("\n");

// ISO Latin Alphabet
byte[] byteArr2 = str.getBytes("ISO-8859-1");
System.out.println("After converted into ISO-8859-1 the String is : ");

for (int j = 0; j < byteArr2.length; j++)
{
System.out.print(byteArr2[j]);
}

System.out.println("\n");

// Little Endian byte order, 16 - bit UCS Transformation format
byte[] byteArr3 = str.getBytes("UTF-16LE");
System.out.println("After converted into UTF-16LE the String is : ");

for (int j = 0; j < byteArr3.length; j++)
{
System.out.print(byteArr3[j]);
}

}
catch (UnsupportedEncodingException g)
{
System.out.println("Unsupported character set" + g);
}


}
```

```java
}
```

Output:

```
/StringGetBytesExample4.java:11: error: unreported exception UnsupportedEncodingException; must be caught or declared to be thrown
byte[] byteArr = str.getBytes("UTF-17");
^
1 error
```

Java String getChars()

Java String getChars() Method Example

FileName: StringGetCharsExample.java

```java
public class StringGetCharsExample{
public static void main(String args[]){
 String str = new String("hello javatpoint how r u");
      char[] ch = new char[10];
      try{
         str.getChars(6, 16, ch, 0);
         System.out.println(ch);
      }catch(Exception ex){System.out.println(ex);}
}}
```

[Test it Now]

Output:

```
javatpoint
```

Java String getChars() Method Example 2

The method throws an exception if index value exceeds array range. Let's see an example.

FileName: StringGetCharsExample2.java

```java
public class StringGetCharsExample2 {
    public static void main(String[] args) {
        String str = new String("Welcome to Javatpoint");
        char[] ch  = new char[20];
        try {
            str.getChars(1, 26, ch, 0);
            System.out.println(ch);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Output:

```
java.lang.StringIndexOutOfBoundsException: offset 10, count 14, length 20
```

Java String getChars() Method Example 3

FileName: StringGetCharsExample3.java

```
public class StringGetCharsExample3
{
// main method
public static void main(String argvs[])
{
String str = "Welcome to JavaTpoint!";

// creating a char arry of size 25
char[] chArr = new char[25];

// start and end indices are same
int srcBeginIndex = 11;
int srcEndIndex = 11;
int dstBeginIndex = 2;

try
{
// invoking the method getChars()
str.getChars(srcBeginIndex, srcEndIndex, chArr, dstBeginIndex);
System.out.println(chArr);
}
catch(Exception excpn)
{
System.out.println(excpn);
}
System.out.println("The getChars() method prints nothing as start and end indices are equal.");
}
}
```

Output:

```
The getChars() method prints nothing as start and end indices are equal.
```

Java String indexOf()

Java String indexOf() Method Example

FileName: IndexOfExample.java

```
public class IndexOfExample{
public static void main(String args[]){
String s1="this is index of example";
//passing substring
int index1=s1.indexOf("is");//returns the index of is substring
int index2=s1.indexOf("index");//returns the index of index substring
System.out.println(index1+"  "+index2);//2 8

//passing substring with from index
int index3=s1.indexOf("is",4);//returns the index of is substring after 4th index
System.out.println(index3);//5 i.e. the index of another is

//passing char value
int index4=s1.indexOf('s');//returns the index of s char value
System.out.println(index4);//3
}}
```

Test it Now

Output:

```
2 8
5
3
```

FileName: IndexOfExample5.java

```
public class IndexOfExample5
{
// main method
public static void main(String argvs[])
{

String str = "Welcome to JavaTpoint";
int count = 0;
int startFrom = 0;
for(; ;)
{

int index = str.indexOf('o', startFrom);

if(index >= 0)
{
// match found. Hence, increment the count
count = count + 1;

// start looking after the searched index
startFrom = index + 1;
}

else
{
```

```
// the value of index is - 1 here. Therefore, terminate the loop
break;
}

}

System.out.println("In the String: "+ str);
System.out.println("The 'o' character has come "+ count + " times");
}
}
```

Output:

```
In the String: Welcome to JavaTpoint
The 'o' character has come 3 times
```

Java String indexOf(String substring) Method Example

The method takes substring as an argument and returns the index of the first character of the substring.

FileName: IndexOfExample2.java

```
public class IndexOfExample2 {
    public static void main(String[] args) {
        String s1 = "This is indexOf method";
        // Passing Substring
        int index = s1.indexOf("method"); //Returns the index of this substring
        System.out.println("index of substring "+index);
    }

}
```

[Test it Now]

Output:

```
index of substring 16
```

Java String indexOf(String substring, int fromIndex) Method Example

The method takes substring and index as arguments and returns the index of the first character that occurs after the given fromIndex.

FileName: IndexOfExample3.java

```
public class IndexOfExample3 {
    public static void main(String[] args) {
        String s1 = "This is indexOf method";
        // Passing substring and index
        int index = s1.indexOf("method", 10); //Returns the index of this substring
        System.out.println("index of substring "+index);
        index = s1.indexOf("method", 20); // It returns -1 if substring does not found
        System.out.println("index of substring "+index);
    }
}
```

Test it Now

Output:

```
index of substring 16
index of substring -1
```

Java String indexOf(int char, int fromIndex) Method Example

The method takes char and index as arguments and returns the index of the first character that occurs after the given fromIndex.

FileName: IndexOfExample4.java

```
public class IndexOfExample4 {
    public static void main(String[] args) {
        String s1 = "This is indexOf method";
        // Passing char and index from
        int index = s1.indexOf('e', 12); //Returns the index of this char
        System.out.println("index of char "+index);
    }
}
```

[Test it Now]()

Output:

```
index of char 17
```

Java String intern()

Java String intern() Method Example

FileName: InternExample.java

```
public class InternExample{
public static void main(String args[]){
String s1=new String("hello");
String s2="hello";
String s3=s1.intern();//returns string from pool, now it will be same as s2
System.out.println(s1==s2);//false because reference variables are pointing to different instance
System.out.println(s2==s3);//true because reference variables are pointing to same instance
}}
```

[Test it Now]()

Output:

```
false
true
```

Java String intern() Method Example 2

Let's see one more example to understand the string intern concept.

FileName: InternExample2.java

```
public class InternExample2 {
    public static void main(String[] args) {
        String s1 = "Javatpoint";
        String s2 = s1.intern();
        String s3 = new String("Javatpoint");
        String s4 = s3.intern();
        System.out.println(s1==s2); // True
        System.out.println(s1==s3); // False
        System.out.println(s1==s4); // True
        System.out.println(s2==s3); // False
        System.out.println(s2==s4); // True
        System.out.println(s3==s4); // False
    }
}
```

Test it Now

Output:

```
true
false
true
false
true
false
```

Java String isEmpty()

Java String isEmpty() method example

FileName: StringIsEmptyExample.java

```
public class IsEmptyExample{
public static void main(String args[]){
String s1="";
String s2="javatpoint";

System.out.println(s1.isEmpty());
System.out.println(s2.isEmpty());
}}
```

Test it Now

Output:

```
true
false
```

Java String isEmpty() Method Example 2

FileName: StringIsEmptyExample2.java

```
public class IsEmptyExample2 {
    public static void main(String[] args) {
        String s1="";
        String s2="Javatpoint";
        // Either length is zero or isEmpty is true
        if(s1.length()==0 || s1.isEmpty())
            System.out.println("String s1 is empty");
        else System.out.println("s1");
        if(s2.length()==0 || s2.isEmpty())
            System.out.println("String s2 is empty");
        else System.out.println(s2);
    }
}
```

Output:

```
String s1 is empty
Javatpoint
```

Empty Vs. Null Strings

FileName: StringIsEmptyExample3.java

```
public class StringIsEmptyExample3
{
// main method
public static void main(String argvs[])
{
String str = null;
if(str.isEmpty())
{
System.out.println("The string is null.");
}
11. else
12. {
13. System.out.println("The string is not null.");
14. }
15. }
16. }
```

Output:

```
Exception in thread "main" java.lang.NullPointerException
at StringIsEmptyExample3.main(StringIsEmptyExample3.java:7)
```

Here, we can use the == operator to check for the null strings.

FileName: StringIsEmptyExample4.java

```
1.  class StringIsEmptyExample4
2.  {
3.  // main method
4.  public static void main(String argvs[])
5.  {
6.  String str = null;
7.  if(str == null)
8.  {
9.  System.out.println("The string is null.");
10. }
11. else
12. {
13. System.out.println("The string is not null.");
14. }
15. }
16. }
```

Output:

```
The string is null.
```

Blank Strings

FileName: StringIsEmptyExample5.java

```
public class StringIsEmptyExample5
{
// main method
public static void main(String argvs[])
{
// a blank string
String str = "     ";
int size = str.length();

// trim the white spaces and after that
// if the string results in the empty string
// then the string is blank; otherwise, not.
if(size == 0)
{
System.out.println("The string is empty. \n");
}
else if(size > 0 && str.trim().isEmpty())
{
System.out.println("The string is blank. \n");
}
else
{
System.out.println("The string is not blank. \n");
}

str = " Welcome to JavaTpoint.  ";
size = str.length();
if(size == 0)
{
System.out.println("The string is empty. \n");
}
if(size > 0 && str.trim().isEmpty())
{
System.out.println("The string is blank. \n");
}
else
{
System.out.println("The string is not blank. \n");
}
}
}
```

Output:

```
The string is blank.
The string is not blank.
```

Java String.join() Method

Java String.join() Method Example

FileName: StringJoinExample.java

Advertisement

```
public class StringJoinExample{
public static void main(String args[]){
String joinString1=String.join("-","welcome","to","javatpoint");
System.out.println(joinString1);
}}
```

[Test it Now]

Output:

```
welcome-to-javatpoint
```

Java String.join() Method Example 2

We can use a delimiter to format the string as we did in the below example to show the date and time.

FileName: StringJoinExample2.java

```java
public class StringJoinExample2 {
    public static void main(String[] args) {
        String date = String.join("/","25","06","2018");
        System.out.print(date);
        String time = String.join(":", "12","10","10");
        System.out.println(" "+time);
    }
}
```

Output:

```
25/06/2018 12:10:10
```

Java String.join() Method Example 3

In the case of using null as a delimiter, we get the null pointer exception. The following example confirms the same.

FileName: StringJoinExample3.java

```java
1.  public class StringJoinExample3
2.  {
3.  // main method
4.  public static void main(String argvs[])
5.  {
6.  String str = null;
7.  str = String.join(null, "abc", "bcd", "apple");
8.  System.out.println(str);
9.  }
10. }
```

Output:

```
Exception in thread "main" java.lang.NullPointerException
at java.base/java.util.Objects.requireNonNull(Objects.java:221)
at java.base/java.lang.String.join(String.java:2393)
at StringJoinExample3.main(StringJoinExample3.java:7)
```

FileName: StringJoinExample4.java

```java
public class StringJoinExample4
```

```
2. {
3. // main method
4. public static void main(String argvs[])
5. {
6. String str = null;
7. str = String.join("India", null);
8. System.out.println(str);
9. }
10. }
```

Output:

```
/StringJoinExample4.java:7: error: reference to join is ambiguous
str = String.join("India", null);
^
both method join(CharSequence,CharSequence...) in String and method
join(CharSequence,Iterable<? extends CharSequence>) in String match
/StringJoinExample4.java:7: warning: non-varargs call of varargs method with inexact argument
type for last parameter;
str = String.join("India", null);
^
cast to CharSequence for a varargs call
cast to CharSequence[] for a non-varargs call and to suppress this warning
1 error
1 warning
```

Java String join() Method Example 4

If the elements that have to be attached with the delimiter have some strings, in which a few of them are null, then the null elements are treated as a normal string, and we do not get any exception or error. Let's understand it through an example.

FileName: StringJoinExample5.java

```
public class StringJoinExample5
{
// main method
public static void main(String argvs[])
{
String str = null;

// one of the element is null however it will be treated as normal string
str = String.join("-", null, " wake up ", " eat ", " write content for JTP ", " eat ", " sleep ");
System.out.println(str);
}
}
```

Output:

```
null- wake up - eat - write content for JTP - eat – sleep
```

Java String lastIndexOf()

Java String lastIndexOf() method example

FileName: LastIndexOfExample.java

```
public class LastIndexOfExample{
public static void main(String args[]){
String s1="this is index of example";//there are 2 's' characters in this sentence
int index1=s1.lastIndexOf('s');//returns last index of 's' char value
System.out.println(index1);//6
}}
```

Test it Now

Output:

```
6
```

Java String lastIndexOf(int ch, int fromIndex) Method Example

Here, we are finding the last index from the string by specifying fromIndex.

FileName: LastIndexOfExample2.java

```
public class LastIndexOfExample2 {
    public static void main(String[] args) {
        String str = "This is index of example";
        int index = str.lastIndexOf('s',5);
        System.out.println(index);
    }
}
```

[Test it Now]

Output: 3

Java String lastIndexOf(String substring) Method Example

It returns the last index of the substring.

FileName: LastIndexOfExample3.java

```
public class LastIndexOfExample3 {
    public static void main(String[] args) {
        String str = "This is last index of example";
        int index = str.lastIndexOf("of");
        System.out.println(index);
    }
}
```

[Test it Now]

Output: 19

Java String lastIndexOf(String substring, int fromIndex) Method Example

It returns the last index of the substring from the fromIndex.

FileName: LastIndexOfExample4.java

```
public class LastIndexOfExample4 {
    public static void main(String[] args) {
        String str = "This is last index of example";
        int index = str.lastIndexOf("of", 25);
        System.out.println(index);
        index = str.lastIndexOf("of", 10);
        System.out.println(index); // -1, if not found
    }
}
```

[Test it Now]

Output:

```
19
-1
```

Java String.length() Method

Java String.length() Method Example

FileName: LengthExample.java

```java
public class LengthExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="python";
System.out.println("string length is: "+s1.length());//10 is the length of javatpoint string
System.out.println("string length is: "+s2.length());//6 is the length of python string
}}
```

Test it Now

Output:

```
string length is: 10
string length is: 6
```

FileName: LengthExample2.java

```java
public class LengthExample2 {
    public static void main(String[] args) {
        String str = "Javatpoint";
        if(str.length()>0) {
            System.out.println("String is not empty and length is: "+str.length());
        }
        str = "";
        if(str.length()==0) {
            System.out.println("String is empty now: "+str.length());
        }
    }
}
```

Output:

```
String is not empty and length is: 10

String is empty now: 0
```

Java String.length() Method Example 3

The length() method is also used to reverse the string.

FileName: LengthExample3.java

```java
class LengthExample3
{
// main method
public static void main(String argvs[])
{
String str = "Welcome To JavaTpoint";
int size = str.length();

System.out.println("Reverse of the string: " + "'" + str + "'" + " is");

for(int i = 0; i < size; i++)
{
// printing in reverse order
System.out.print(str.charAt(str.length() - i - 1));
}

}
}
```

Output:

```
Reverse of the string: 'Welcome To JavaTpoint' is

tniopTavaJ oT emocleW
```

Java String.length() Method Example 4

The length() method can also be used to find only the white spaces present in the string. Observe the following example.

File Name: LengthExample4.java

```
public class LengthExample4
{
// main method
public static void main(String argvs[])
{
String str = " Welcome To JavaTpoint ";
int sizeWithWhiteSpaces = str.length();

System.out.println("In the string: " + "'" + str + "'");

str = str.replace(" ", "");
int sizeWithoutWhiteSpaces = str.length();

// calculating the white spaces
int noOfWhieSpaces = sizeWithWhiteSpaces - sizeWithoutWhiteSpaces;

System.out.print("Total number of whitespaces present are: " + noOfWhieSpaces);
}
}
```

Output:

```
In the string: ' Welcome To JavaTpoint '
Total number of whitespaces present are: 4
```

Java String replace()

File Name: StringReplaceDemo.java

```
public class StringReplaceDemo {
    public static void main(String[] args) {
        // Feature 1: Case-Sensitivity
        String str1 = "Hello World";
        String replaced1 = str1.replace("o", "*");
        System.out.println("Case-Sensitivity:");
        System.out.println("Original String: " + str1);
        System.out.println("Replaced String: " + replaced1); // Output: Hell* W*rld
        System.out.println();
        // Feature 2: Replacing Substrings
        String str2 = "Java is awesome";
        String replaced2 = str2.replace("awesome", "fantastic");
        System.out.println("Replacing Substrings:");
        System.out.println("Original String: " + str2);
        System.out.println("Replaced String: " + replaced2); // Output: Java is fantastic
        System.out.println();
        // Feature 3: Replacing with an Empty String
        String str3 = "Remove these spaces";
        String replaced3 = str3.replace(" ", "");
        System.out.println("Replacing with an Empty String:");
        System.out.println("Original String: " + str3);
        System.out.println("Replaced String: " + replaced3); // Output: Removethesespaces
    }
}
```

Output:

```
Case-Sensitivity:
Original String: Hello World
Replaced String: Hell* W*rld

Replacing Substrings:
Original String: Java is awesome
Replaced String: Java is fantastic

Replacing with an Empty String:
Original String: Remove these spaces
Replaced String: Removethesespaces
```

Java String replace(char old, char new) Method Example

FileName: ReplaceExample1.java

```
public class ReplaceExample1{
public static void main(String args[]){
```

```
String s1="javatpoint is a very good website";
String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'
System.out.println(replaceString);
}}
```

Test it Now

Output:

```
jevetpoint is e very good website
```

Java String replace(CharSequence target, CharSequence replacement) Method Example

FileName: ReplaceExample2.java

```
public class ReplaceExample2{
public static void main(String args[]){
String s1="my name is khan my name is java";
String replaceString=s1.replace("is","was");//replaces all occurrences of "is" to "was"
System.out.println(replaceString);
}}
```

Test it Now

Output:

```
my name was khan my name was java
```

Java String replace() Method Example 3

FileName: ReplaceExample3.java

```
public class ReplaceExample3 {
    public static void main(String[] args) {
        String str = "oooooo-hhhh-oooooo";
        String rs = str.replace("h","s"); // Replace 'h' with 's'
        System.out.println(rs);
        rs = rs.replace("s","h"); // Replace 's' with 'h'
        System.out.println(rs);
    }
}
```

Output:

```
oooooo-ssss-oooooo
oooooo-hhhh-oooooo
```

Java String replace() Method Example 4

The replace() method throws the NullPointerException when the replacement or target is null. The following example confirms the same.

FileName: ReplaceExample4.java

```
public class ReplaceExample4
{
// main method
public static void main(String argvs[])
{

String str = "For learning Java, JavaTpoint is a very good site.";
int size = str.length();

System.out.println(str);
String target = null;

// replacing null with JavaTpoint. Hence, the NullPointerException is raised.
str = str.replace(target, "JavaTpoint ");

System.out.println(str);

}
}
```

Output:

```
For learning Java, JavaTpoint is a very good site.
Exception in thread "main" java.lang.NullPointerException
at java.base/java.lang.String.replace(String.java:2142)
```

at ReplaceExample4.main(ReplaceExample4.java:12)

Java String replaceAll()

Java String replaceAll() example: replace character

Let's see an example to replace all the occurrences of a single character.

FileName: ReplaceAllExample1.java

```
public class ReplaceAllExample1{
public static void main(String args[]){
String s1="javatpoint is a very good website";
String replaceString=s1.replaceAll("a","e");//replaces all occurrences of "a" to "e"
System.out.println(replaceString);
}}
```

[*Test it Now*]

Output:

```
jevetpoint is e very good website
```

Java String replaceAll() example: replace word

Let's see an example to replace all the occurrences of a single word or set of words.

FileName: ReplaceAllExample2.java

```
public class ReplaceAllExample2{
public static void main(String args[]){
String s1="My name is Khan. My name is Bob. My name is Sonoo.";
String replaceString=s1.replaceAll("is","was");//replaces all occurrences of "is" to "was"
System.out.println(replaceString);
}}
```

[*Test it Now*]

Output:

```
My name was Khan. My name was Bob. My name was Sonoo.
```

Java String replaceAll() example: remove white spaces

Let's see an example to remove all the occurrences of white spaces.

FileName: ReplaceAllExample3.java

```
public class ReplaceAllExample3{
public static void main(String args[]){
String s1="My name is Khan. My name is Bob. My name is Sonoo.";
String replaceString=s1.replaceAll("\\s","");
System.out.println(replaceString);
}}
```

Test it Now

Output:

```
MynameisKhan.MynameisBob.MynameisSonoo.
```

Java String replaceAll() Method Example 4

The replaceAll() method throws the PatternSyntaxException when there is an improper regular expression. Look at the following example.

FileName: ReplaceAllExample4.java

```
public class ReplaceAllExample4
{
// main method
public static void main(String argvs[])
{

// input string
String str = "For learning Java, JavaTpoint is a very good site.";

System.out.println(str);

String regex = "\\"; // the regular expression is not valid.

// invoking the replaceAll() method raises the PatternSyntaxException
str = str.replaceAll(regex, "JavaTpoint ");

System.out.println(str);
```

```java

}
}
```

Output:

```
For learning Java, JavaTpoint is a very good site.


Exception in thread "main" java.util.regex.PatternSyntaxException: Unexpected internal error near index 1
\
at java.base/java.util.regex.Pattern.error(Pattern.java:2015)
at java.base/java.util.regex.Pattern.compile(Pattern.java:1784)
at java.base/java.util.regex.Pattern.(Pattern.java:1427)
at java.base/java.util.regex.Pattern.compile(Pattern.java:1068)
at java.base/java.lang.String.replaceAll(String.java:2126)
at ReplaceExample4.main(ReplaceExample4.java:12)
```

Java String replaceAll() Method Example 5

The replaceAll() method can also be used to insert spaces between characters.

FileName: ReplaceAllExample5.java

```java
public class ReplaceAllExample5
{
// main method
public static void main(String argvs[])
{

// input string
String str = "JavaTpoint";
System.out.println(str);

String regex = "";
// adding a white space before and after every character of the input string.
str = str.replaceAll(regex, " ");

System.out.println(str);

}
}
```

Output:

Advertisement

```
JavaTpoint
J a v a T p o i n t
```

Java String replaceAll() Method Example 6

Even the null regular expression is also not accepted by the replaceAll() method as the NullPointerException is raised.

FileName: ReplaceAllExample6.java

```
public class ReplaceAllExample6
{
// main method
public static void main(String argvs[])
{

// input string
String str = "JavaTpoint";
System.out.println(str);

String regex = null; // regular expression is null

str = str.replaceAll(regex, " ");

System.out.println(str);

}
}
```

Output:

```
JavaTpoint

Exception in thread "main" java.lang.NullPointerException
at java.base/java.util.regex.Pattern.(Pattern.java:1426)
at java.base/java.util.regex.Pattern.compile(Pattern.java:1068)
at java.base/java.lang.String.replaceAll(String.java:2126)
at ReplaceAllExample6.main(ReplaceAllExample6.java:13)
```

Java String split()

Java String split() method example

The given example returns total number of words in a string excluding space only. It also includes special characters.

```
public class SplitExample{
public static void main(String args[]){
String s1="java string split method by javatpoint";
String[] words=s1.split("\\s");//splits the string based on whitespace
//using java foreach loop to print elements of string array
for(String w:words){
System.out.println(w);
}
}}
```

[Test it Now]

```
java
string
split
method
by
javatpoint
```

Java String split() method with regex and length example

```
public class SplitExample2{
public static void main(String args[]){
String s1="welcome to split world";
System.out.println("returning words:");
for(String w:s1.split("\\s",0)){
System.out.println(w);
}
System.out.println("returning words:");
for(String w:s1.split("\\s",1)){
System.out.println(w);
}
System.out.println("returning words:");
for(String w:s1.split("\\s",2)){
System.out.println(w);
}

}}
```

Test it Now

```
returning words:
welcome
to
split
world
returning words:
welcome to split world
returning words:
welcome
to split world
```

Java String split() method with regex and length example 2

Here, we are passing split limit as a second argument to this function. This limits the number of splitted strings.

```java
public class SplitExample3 {
    public static void main(String[] args) {
        String str = "Javatpointtt";
        System.out.println("Returning words:");
        String[] arr = str.split("t", 0);
        for (String w : arr) {
            System.out.println(w);
        }
        System.out.println("Split array length: "+arr.length);
    }
}
```

```
Returning words:
Java
poin
Split array length: 2
```

Java String startsWith()

Java String startsWith() method example

The startsWith() method considers the case-sensitivity of characters. Consider the following example.

FileName: StartsWithExample.java

```java
public class StartsWithExample
{
// main method
public static void main(String args[])
{
// input string
String s1="java string split method by javatpoint";
System.out.println(s1.startsWith("ja"));  // true
System.out.println(s1.startsWith("java string"));   // true
System.out.println(s1.startsWith("Java string"));  // false as 'j' and 'J' are different
```

```
    }
}
```

Output:

```
true
true
false
```

Java String startsWith(String prefix, int offset) Method Example

It is an overloaded method of the startWith() method that is used to pass an extra argument (offset) to the function. The method works from the passed offset. Let's see an example.

FileName: StartsWithExample2.java

Advertisement

```
public class StartsWithExample2 {
    public static void main(String[] args) {
        String str = "Javatpoint";
        // no offset mentioned; hence, offset is 0 in this case.
        System.out.println(str.startsWith("J")); // True

         // no offset mentioned; hence, offset is 0 in this case.
        System.out.println(str.startsWith("a")); // False
        // offset is 1
        System.out.println(str.startsWith("a",1)); // True
    }
}
```

Output:

```
true
false
true
```

Java String startsWith() Method Example - 3

If we adding an empty string at the beginning of a string, then it has no impact at all on the string.

"" + "Tokyo Olympics" = "Tokyo Olympics"s

It means one can say that a string in Java always starts with the empty string. Let's confirm the same with the help of Java code.

FileName: StartsWithExample3.java

```
public class StartsWithExample3
{
// main method
public static void main(String argvs[])
{
// input string
String str = "Tokyo Olympics";

if(str.startsWith(""))
{
System.out.println("The string starts with the empty string.");
}
else
{
System.
out.println("The string does not start with the empty string.");
}

}
}
```

Output:

```
The string starts with the empty string.
```

Java String substring()

Java substring() Method Example

FileName: SubstringExample.java

```
public class SubstringExample{
public static void main(String args[]){
String s1="javatpoint";
System.out.println(s1.substring(2,4));//returns va
System.out.println(s1.substring(2));//returns vatpoint
}}
```

Test it Now

Output:

```
va
vatpoint
```

---

substring() Method Example 2

FileName: SubstringExample2.java

```
public class SubstringExample2 {
    public static void main(String[] args) {
        String s1="Javatpoint";
        String substr = s1.substring(0); // Starts with 0 and goes to end
        System.out.println(substr);
        String substr2 = s1.substring(5,10); // Starts from 5 and goes to 10
        System.out.println(substr2);
        String substr3 = s1.substring(5,15); // Returns Exception
    }
}
```

Output:

```
Javatpoint
point
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: begin 5, end 15, length
10
```

FileName: SubstringExample3.java

```
public class SubstringExample3
{
// main method
public static void main(String argvs[])
{
```

```
String str[] =
{
"Praveen Kumar",
"Yuvraj Singh",
"Harbhajan Singh",
"Gurjit Singh",
"Virat Kohli",
"Rohit Sharma",
"Sandeep Singh",
"Milkha Singh"
};

String surName = "Singh";
int surNameSize = surName.length();

int size = str.length;

for(int j = 0; j < size; j++)
{
   int length = str[j].length();
   // extracting the surname
   String subStr = str[j].substring(length - surNameSize);

   // checks whether the surname is equal to "Singh" or not
   if(subStr.equals(surName))
   {
      System.out.println(str[j]);
   }
}

}
```

```java
}
```

Output:

```
Yuvraj Singh
Harbhajan Singh
Gurjit Singh
Sandeep Singh
Milkha Singh
```

FileName: SubstringExample4.java

```java
public class SubstringExample4
{
public boolean isPalindrome(String str)
{
int size = str.length();

// handling the base case
if(size == 0 || size == 1)
{
// an empty string
// or a string of only one character
// is always a palindrome
return true;
}
String f = str.substring(0, 1);
String l = str.substring(size - 1);
// comparing first and the last character of the string
if(l.equals(f))
{
// recursively finding the solution using the substring() method
// reducing the number of characters of the by 2 for the next recursion
return isPalindrome(str.substring(1, size - 1));
}
return false;
}
// main method
public static void main(String argvs[])
{
// instantiating the class SubstringExample4
SubstringExample4 obj = new SubstringExample4();
String str[] =
{
"madam",
"rock",
"eye",
"noon",
"kill"
};
int size = str.length;

for(int j = 0; j < size; j++)
{
if(obj.isPalindrome(str[j]))
{
System.out.println(str[j] + " is a palindrome.");
}
else
{
System.out.println(str[j] + " is not a palindrome.");
}
}
}
}
```

Output:

```
madam is a palindrome.
rock is not a palindrome.
eye is a palindrome.
noon is a palindrome.
kill is not a palindrome.
```

Java String toCharArray()

Java String toCharArray() method example

```java
public class StringToCharArrayExample{
public static void main(String args[]){
String s1="hello";
char[] ch=s1.toCharArray();
for(int i=0;i<ch.length;i++){
System.out.print(ch[i]);
}
}}
```

Test it Now

Output:

```
Hello
```

Java String toCharArray() Method Example 2

Let's see one more example of char array. It is useful method which returns char array from the string without writing any custom code.

```java
public class StringToCharArrayExample2 {
    public static void main(String[] args) {
        String s1 = "Welcome to Javatpoint";
        char[] ch = s1.toCharArray();
        int len = ch.length;
        System.out.println("Char Array length: " + len);
        System.out.println("Char Array elements: ");
        for (int i = 0; i < len; i++) {
            System.out.println(ch[i]);
        }
    }
}
```

Output:

```
Char Array length: 21
Char Array elements:
W
e
l
c
o
m
e
 
t
o
J
a
v
a
t
p
o
i
n
t
```

Java String toLowerCase()

Java String toLowerCase() method example

```
public class StringLowerExample{
public static void main(String args[]){
String s1="JAVATPOINT HELLO stRIng";
String s1lower=s1.toLowerCase();
System.out.println(s1lower);
}}
```

Test it Now

Output:

```
javatpoint hello string
```

Java String toLowerCase(Locale locale) Method Example 2

This method allows us to pass locale too for the various langaugues. Let's see an example below where we are getting string in english and turkish both.

```
import java.util.Locale;
public class StringLowerExample2 {
   public static void main(String[] args) {
      String s = "JAVATPOINT HELLO stRIng";
      String eng = s.toLowerCase(Locale.ENGLISH);
      System.out.println(eng);
      String turkish = s.toLowerCase(Locale.forLanguageTag("tr")); // It shows i without dot
      System.out.println(turkish);
   }
}
```

Output:

```
javatpoint hello string
javatpo?nt hello str?ng
```

Java String toUpperCase()

Java String toUpperCase() method example

```
public class StringUpperExample{
public static void main(String args[]){
String s1="hello string";
String s1upper=s1.toUpperCase();
System.out.println(s1upper);
}}
```

Test it Now

Output:

```
HELLO STRING
```

Java String toUpperCase(Locale locale) Method Example 2

```
import java.util.Locale;
public class StringUpperExample2 {
    public static void main(String[] args) {
        String s = "hello string";
        String turkish = s.toUpperCase(Locale.forLanguageTag("tr"));
        String english = s.toUpperCase(Locale.forLanguageTag("en"));
        System.out.println(turkish);//will print I with dot on upper side
        System.out.println(english);
    }
}
```

Output:

```
HELLO STR?NG
HELLO STRING
```

Java String.trim() Method

Java String.trim() Method Example

File Name: StringTrimExample.java

```
public class StringTrimExample{
public static void main(String args[]){
String s1="  hello string   ";
System.out.println(s1+"javatpoint");//without trim()
System.out.println(s1.trim()+"javatpoint");//with trim()
}}
```

Test it Now

Output

```
hello string javatpoint
hello stringjavatpoint
```

Java String.trim() Method Example 2

The example demonstrates the use of the trim() method. This method removes all the trailing spaces so the length of the string also reduces. Let's see an example.

FileName: StringTrimExample2.java

```java
public class StringTrimExample2 {
   public static void main(String[] args) {
      String s1 ="  hello java string   ";
      System.out.println(s1.length());
      System.out.println(s1); //Without trim()
      String tr = s1.trim();
      System.out.println(tr.length());
      System.out.println(tr); //With trim()
   }
}
```

Output

```
22
hello java string
17
hello java string
```

Java String.trim() Method Example 3

The trim() can be used to check whether the string only contains white spaces or not. The following example shows the same.

FileName: TrimExample3.java

```java
public class TrimExample3
{
// main method
public static void main(String argvs[])
{
```

```java

String str = " abc ";

if((str.trim()).length() > 0)
{
System.out.println("The string contains characters other than white spaces \n");
}
else
{
System.out.println("The string contains only white spaces \n");
}

str = "    ";

if((str.trim()).length() > 0)
{
System.out.println("The string contains characters other than white spaces \n");
}
else
{
System.out.println("The string contains only white spaces \n");
}

}
}
```

Output

```
The string contains characters other than white spaces


The string contains only white spaces
```

Java String.trim() Method Example 4

FileName: TrimExample4.java

```
public class TrimExample4
{
// main method
public static void main(String argvs[])
{

// the string contains white spaces
// therefore, trimming the spaces leads to the
// generation of new string
String str = " abc ";

// str1 stores a new string
String str1 = str.trim();

// the hashcode of str and str1 is different
System.out.println(str.hashCode());
System.out.println(str1.hashCode() + "\n");

// no white space present in the string s
// therefore, the reference of the s is returned
// when the trim() method is invoked
String s = "xyz";
String s1 = s.trim();

// the hashcode of s and s1 is the same
System.out.println(s.hashCode());
System.out.println(s1.hashCode());

}
}
```

Output

The string contains characters other than white spaces

The string contains only white spaces

Exception Handling in Java

JavaExceptionExample.java

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

[***Test it Now***]

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Java try-catch block

Example 1

TryCatchExample1.java

```
public class TryCatchExample1 {

    public static void main(String[] args) {

        int data=50/0; //may throw exception

        System.out.println("rest of the code");

    }

}
```

[Test it Now]()

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Example 2

TryCatchExample2.java

```java
public class TryCatchExample2 {

    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception
        }
            //handling the exception
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }
        System.out.println("rest of the code");
    }

}
```

[Test it Now]()

Output:

```
java.lang.ArithmeticException: / by zero
rest of the code
```

TryCatchExample3.java

```java
public class TryCatchExample3 {

    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception
                        // if exception occurs, the remaining statement will not exceute
        System.out.println("rest of the code");
        }
            // handling the exception
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }

    }

}
```

[Test it Now]()

Output:

```
java.lang.ArithmeticException: / by zero
```

Example 4

Here, we handle the exception using the parent class exception.

TryCatchExample4.java

```java
public class TryCatchExample4 {

    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception
        }
            // handling the exception by using Exception class
        catch(Exception e)
        {
          System.out.println(e);
        }
        System.out.println("rest of the code");
    }

}
```

Test it Now

Output:

```
java.lang.ArithmeticException: / by zero
rest of the code
```

Example 5

Let's see an example to print a custom message on exception.

TryCatchExample5.java

```
public class TryCatchExample5 {

    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception
        }
            // handling the exception
        catch(Exception e)
        {
                 // displaying the custom message
          System.out.println("Can't divided by zero");
        }
    }

}
```

Test it Now

Output:

```
Can't divided by zero
```

Example 6

Let's see an example to resolve the exception in a catch block.

TryCatchExample6.java

```
public class TryCatchExample6 {

    public static void main(String[] args) {
        int i=50;
        int j=0;
        int data;
        try
        {
        data=i/j; //may throw exception
        }
            // handling the exception
        catch(Exception e)
        {
             // resolving the exception in catch block
            System.out.println(i/(j+2));
        }
    }
}
```

Test it Now

Output:

```
25
```

Example 7

In this example, along with try block, we also enclose exception code in a catch block.

TryCatchExample7.java

```java
public class TryCatchExample7 {

  public static void main(String[] args) {

    try
    {
    int data1=50/0; //may throw exception

    }
        // handling the exception
    catch(Exception e)
    {
       // generating the exception in catch block
    int data2=50/0; //may throw exception

    }
  System.out.println("rest of the code");
  }
}
```

Test it Now

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

```java
TryCatchExample8.java
  public static void main(String[] args) {
    try
    {
    int data=50/0; //may throw exception

    }
       // try to handle the ArithmeticException using ArrayIndexOutOfBoundsException
```

```
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println(e);
        }
        System.out.println("rest of the code");
    }

}
```

Test it Now

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Example 9

Let's see an example to handle another unchecked exception.

TryCatchExample9.java

```
public class TryCatchExample9 {

    public static void main(String[] args) {
        try
        {
        int arr[]= {1,3,5,7};
        System.out.println(arr[10]); //may throw exception
        }
            // handling the array exception
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println(e);
        }
        System.out.println("rest of the code");
    }

}
```

[Test it Now]

Output:

```
java.lang.ArrayIndexOutOfBoundsException: 10
rest of the code
```

Example 10

Let's see an example to handle checked exception.

TryCatchExample10.java

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class TryCatchExample10 {

    public static void main(String[] args) {


        PrintWriter pw;
        try {
            pw = new PrintWriter("jtp.txt"); //may throw exception
            pw.println("saved");
        }
// providing the checked exception handler
 catch (FileNotFoundException e) {

            System.out.println(e);
        }
    System.out.println("File saved successfully");
    }
}
```

[Test it Now]

Output:

File saved successfully

Java Catch Multiple Exceptions

Example 1

Let's see a simple example of java multi-catch block.

MultipleCatchBlock1.java

```
public class MultipleCatchBlock1 {

   public static void main(String[] args) {

         try{
              int a[]=new int[5];
              a[5]=30/0;
             }
             catch(ArithmeticException e)
               {
                System.out.println("Arithmetic Exception occurs");
               }
             catch(ArrayIndexOutOfBoundsException e)
               {
                System.out.println("ArrayIndexOutOfBounds Exception occurs");
               }
             catch(Exception e)
               {
                System.out.println("Parent Exception occurs");
               }
             System.out.println("rest of the code");
   }
}
```

Test it Now

Output:

```
Arithmetic Exception occurs
rest of the code
```

Example 2

MultipleCatchBlock2.java

```
public class MultipleCatchBlock2 {

   public static void main(String[] args) {

         try{
             int a[]=new int[5];

             System.out.println(a[10]);
            }
            catch(ArithmeticException e)
              {
               System.out.println("Arithmetic Exception occurs");
              }
            catch(ArrayIndexOutOfBoundsException e)
              {
               System.out.println("ArrayIndexOutOfBounds Exception occurs");
              }
            catch(Exception e)
              {
               System.out.println("Parent Exception occurs");
              }
            System.out.println("rest of the code");
   }
}
```

[Test it Now]

Output:

```
ArrayIndexOutOfBounds Exception occurs
rest of the code
```

MultipleCatchBlock3.java

```java
public class MultipleCatchBlock3 {

    public static void main(String[] args) {

          try{
               int a[]=new int[5];
               a[5]=30/0;
               System.out.println(a[10]);
              }
              catch(ArithmeticException e)
                {
                 System.out.println("Arithmetic Exception occurs");
                }
              catch(ArrayIndexOutOfBoundsException e)
                {
                 System.out.println("ArrayIndexOutOfBounds Exception occurs");
                }
              catch(Exception e)
                {
                 System.out.println("Parent Exception occurs");
                }
              System.out.println("rest of the code");
    }
}
```

Test it Now

Output:

```
Arithmetic Exception occurs
```

rest of the code

Example 4

In this example, we generate NullPointerException, but didn't provide the corresponding exception type. In such case, the catch block containing the parent exception class Exception will invoked.

MultipleCatchBlock4.java

```
public class MultipleCatchBlock4 {

    public static void main(String[] args) {

           try{
               String s=null;
               System.out.println(s.length());
              }
              catch(ArithmeticException e)
                {
                 System.out.println("Arithmetic Exception occurs");
                }
              catch(ArrayIndexOutOfBoundsException e)
                {
                 System.out.println("ArrayIndexOutOfBounds Exception occurs");
                }
              catch(Exception e)
                {
                 System.out.println("Parent Exception occurs");
                }
              System.out.println("rest of the code");
    }
}
```

[*Test it Now*]

Output:

Parent Exception occurs

rest of the code

Example 5

Let's see an example, to handle the exception without maintaining the order of exceptions (i.e. from most specific to most general).

MultipleCatchBlock5.java

```
class MultipleCatchBlock5{
 public static void main(String args[]){
  try{
   int a[]=new int[5];
   a[5]=30/0;
  }
  catch(Exception e){System.out.println("common task completed");}
  catch(ArithmeticException e){System.out.println("task1 is completed");}
  catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
  System.out.println("rest of the code...");
 }
}
```

Test it Now

Output:

```
Compile-time error
```

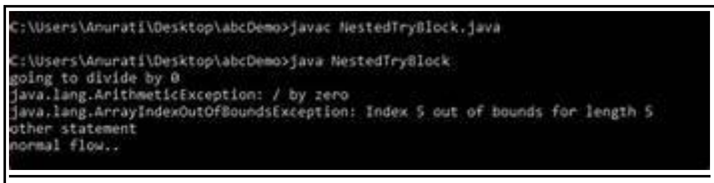
Java Nested try block

NestedTryBlock.java

```
public class NestedTryBlock{
 public static void main(String args[]){
 //outer try block
  try{
  //inner try block 1
   try{
    System.out.println("going to divide by 0");
```

```
     int b =39/0;
    }
    //catch block of inner try block 1
    catch(ArithmeticException e)
    {
      System.out.println(e);
    }


    //inner try block 2
    try{
    int a[]=new int[5];

    //assigning the value out of array bounds
     a[5]=4;
     }

    //catch block of inner try block 2
    catch(ArrayIndexOutOfBoundsException e)
    {
       System.out.println(e);
    }


    System.out.println("other statement");
  }
  //catch block of outer try block
  catch(Exception e)
  {
    System.out.println("handled the exception (outer catch)");
  }

  System.out.println("normal flow..");
 }
}
```

Output:

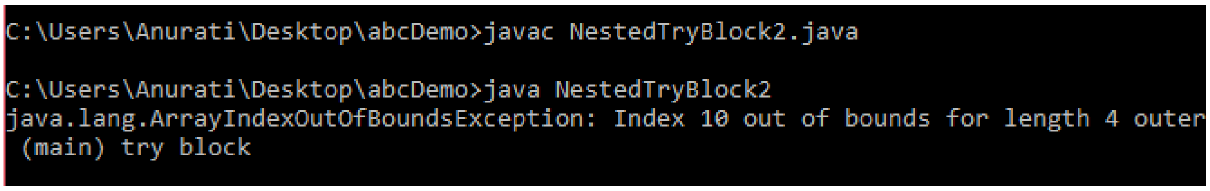


```
C:\Users\Anurati\Desktop\abcDemo>javac NestedTryBlock.java

C:\Users\Anurati\Desktop\abcDemo>java NestedTryBlock
going to divide by 0
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
other statement
normal flow..
```

Example 2

```
public class NestedTryBlock2 {

    public static void main(String args[])
    {
        // outer (main) try block
        try {

            //inner try block 1
            try {

                // inner try block 2
                try {
                    int arr[] = { 1, 2, 3, 4 };

                    //printing the array element out of its bounds
                    System.out.println(arr[10]);
                }

                // to handles ArithmeticException
                catch (ArithmeticException e) {
                    System.out.println("Arithmetic exception");
                    System.out.println(" inner try block 2");
                }
            }

            // to handle ArithmeticException
            catch (ArithmeticException e) {
                System.out.println("Arithmetic exception");
                System.out.println("inner try block 1");
            }
        }

        // to handle ArrayIndexOutOfBoundsException
        catch (ArrayIndexOutOfBoundsException e4) {
            System.out.print(e4);
            System.out.println(" outer (main) try block");
        }
        catch (Exception e5) {
            System.out.print("Exception");
            System.out.println(" handled in main try-block");
        }
    }
}
```

Output:



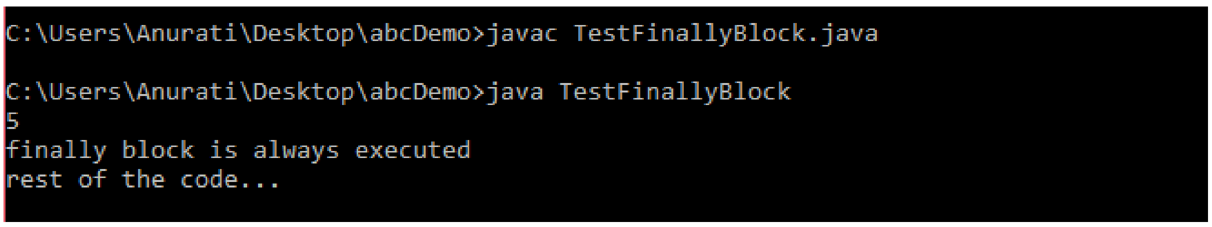
```
C:\Users\Anurati\Desktop\abcDemo>javac NestedTryBlock2.java

C:\Users\Anurati\Desktop\abcDemo>java NestedTryBlock2
java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 4 outer
 (main) try block
```

Java finally block

TestFinallyBlock.java

```
class TestFinallyBlock {
  public static void main(String args[]){
  try{
//below code do not throw any exception
   int data=25/5;
   System.out.println(data);
  }
//catch won't be executed
  catch(NullPointerException e){
System.out.println(e);
}
//executed regardless of exception occurred or not
 finally {
System.out.println("finally block is always executed");
}

System.out.println("rest of phe code...");
  }
}
```

Output:

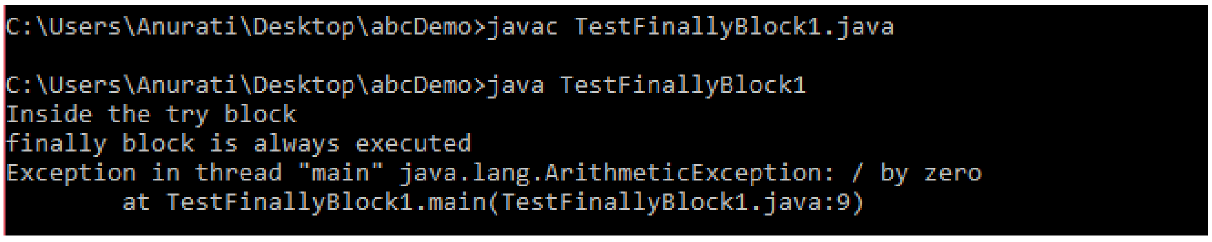


```
C:\Users\Anurati\Desktop\abcDemo>javac TestFinallyBlock.java

C:\Users\Anurati\Desktop\abcDemo>java TestFinallyBlock
5
finally block is always executed
rest of the code...
```

Case 2: When an exception occurr but not handled by the catch block TestFinallyBlock1.java

```
public class TestFinallyBlock1{
      public static void main(String args[]){

      try {

         System.out.println("Inside the try block");

         //below code throws divide by zero exception
        int data=25/0;
        System.out.println(data);
      }
      //cannot handle Arithmetic type exception
      //can only accept Null Pointer type exception
      catch(NullPointerException e){
        System.out.println(e);
      }

      //executes regardless of exception occured or not
      finally {
        System.out.println("finally block is always executed");
      }

      System.out.println("rest of the code...");
      }
    }
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestFinallyBlock1.java

C:\Users\Anurati\Desktop\abcDemo>java TestFinallyBlock1
Inside the try block
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at TestFinallyBlock1.main(TestFinallyBlock1.java:9)
```

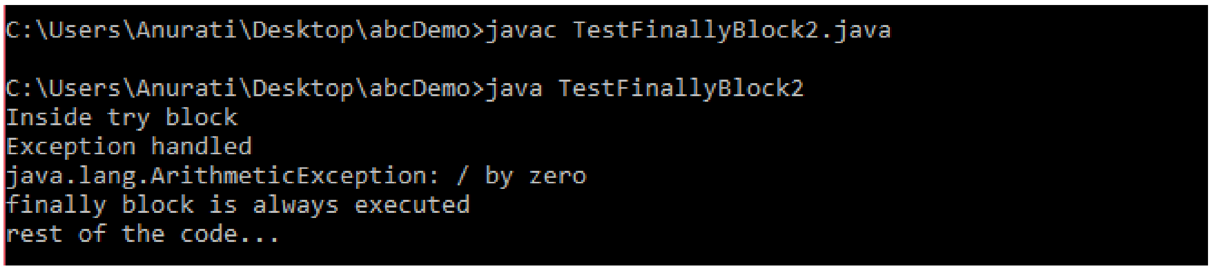
Case 3: When an exception occurs and is handled by the catch block

Example:

TestFinallyBlock2.java

```
public class TestFinallyBlock2{
    public static void main(String args[]){

    try {

      System.out.println("Inside try block");

      //below code throws divide by zero exception
     int data=25/0;
     System.out.println(data);
    }

    //handles the Arithmetic Exception / Divide by zero exception
    catch(ArithmeticException e){
      System.out.println("Exception handled");
      System.out.println(e);
    }

    //executes regardless of exception occured or not
    finally {
      System.out.println("finally block is always executed");
    }

    System.out.println("rest of the code...");
    }
  }
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestFinallyBlock2.java

C:\Users\Anurati\Desktop\abcDemo>java TestFinallyBlock2
Inside try block
Exception handled
java.lang.ArithmeticException: / by zero
finally block is always executed
rest of the code...
```

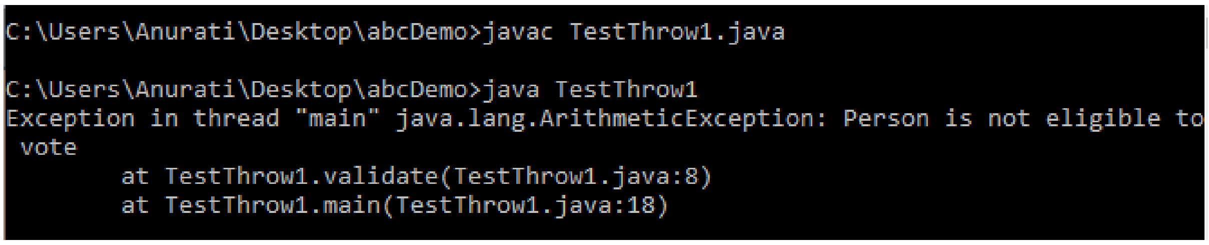
Rule: For each try block there can be zero or more catch blocks, but only one finally block.

Java throw Exception

TestThrow1.java

```
1.  public class TestThrow1 {
2.     //function to check if person is eligible to vote or not
3.     public static void validate(int age) {
4.        if(age<18) {
5.           //throw Arithmetic exception if not eligible to vote
6.           throw new ArithmeticException("Person is not eligible to vote");
7.        }
8.        else {
9.           System.out.println("Person is eligible to vote!!");
10.       }
11.    }
12.    //main method
13.    public static void main(String args[]){
14.       //calling the function
15.       validate(13);
16.       System.out.println("rest of the code...");
17.   }
18. }
```

Output:

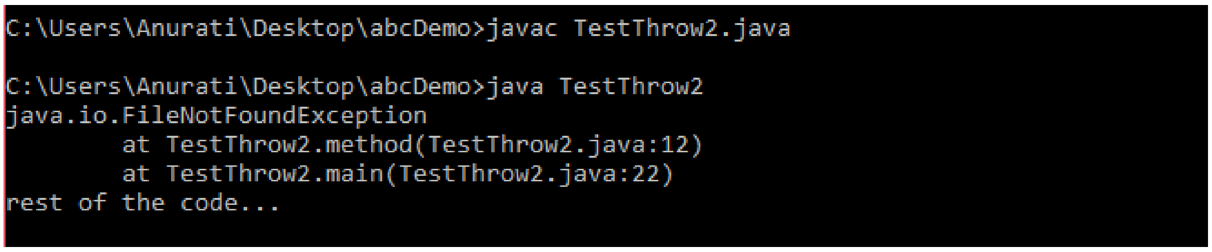


```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow1.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrow1
Exception in thread "main" java.lang.ArithmeticException: Person is not eligible to
 vote
        at TestThrow1.validate(TestThrow1.java:8)
        at TestThrow1.main(TestThrow1.java:18)
```

TestThrow2.java

```
import java.io.*;

public class TestThrow2 {

  //function to check if person is eligible to vote or not
  public static void method() throws FileNotFoundException {

    FileReader file = new FileReader("C:\\Users\\Anurati\\Desktop\\abc.txt");
    BufferedReader fileInput = new BufferedReader(file);


    throw new FileNotFoundException();

  }
  //main method
  public static void main(String args[]){
    try
    {
      method();
    }
    catch (FileNotFoundException e)
    {
      e.printStackTrace();
    }
    System.out.println("rest of the code...");
 }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow2.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrow2
java.io.FileNotFoundException
        at TestThrow2.method(TestThrow2.java:12)
        at TestThrow2.main(TestThrow2.java:22)
rest of the code...
```

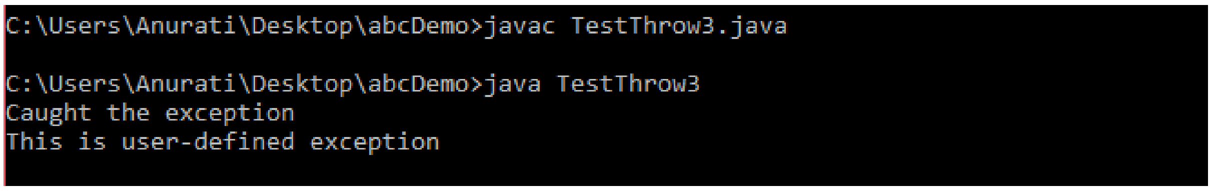
Example 3: Throwing User-defined Exception

exception is everything else under the Throwable class.

TestThrow3.java

```
// class represents user-defined exception
class UserDefinedException extends Exception
{
    public UserDefinedException(String str)
    {
        // Calling constructor of parent Exception
        super(str);
    }
}
// Class that uses above MyException
public class TestThrow3
{
    public static void main(String args[])
    {
        try
        {
            // throw an object of user defined exception
            throw new UserDefinedException("This is user-defined exception");
        }
        catch (UserDefinedException ude)
        {
            System.out.println("Caught the exception");
            // Print the message from MyException object
            System.out.println(ude.getMessage());
        }
    }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow3.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrow3
Caught the exception
This is user-defined exception
```

Java Exception Propagation

Exception Propagation Example

TestExceptionPropagation1.java

```
class TestExceptionPropagation1{
  void m(){
    int data=50/0;
  }
  void n(){
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   TestExceptionPropagation1 obj=new TestExceptionPropagation1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

Test it Now

Output:

```
exception handled
normal flow...
```

Exception Propagation Example

TestExceptionPropagation1.java

```
class TestExceptionPropagation2{
  void m(){
    throw new java.io.IOException("device error");//checked exception
  }
  void n(){
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handeled");}
  }
  public static void main(String args[]){
   TestExceptionPropagation2 obj=new TestExceptionPropagation2();
   obj.p();
   System.out.println("normal flow");
  }
}
```

Test it Now

Output:

```
Compile Time Error
```

Java throws keyword

Java throws Example

Let's see the example of Java throws clause which describes that checked exceptions can be propagated by throws keyword.

Testthrows1.java

```
import java.io.IOException;
class Testthrows1{
 void m()throws IOException{
   throw new IOException("device error");//checked exception
 }
 void n()throws IOException{
   m();
 }
 void p(){
  try{
   n();
  }catch(Exception e){System.out.println("exception handled");}
 }
 public static void main(String args[]){
  Testthrows1 obj=new Testthrows1();
  obj.p();
  System.out.println("normal flow...");
 }
}
```

Test it Now

Output:

```
exception handled
normal flow...
```

Testthrows2.java

```
import java.io.*;
class M{
 void method()throws IOException{
  throw new IOException("device error");
 }
}
```

```
public class Testthrows2{
  public static void main(String args[]){
   try{
    M m=new M();
    m.method();
   }catch(Exception e){System.out.println("exception handled");}

   System.out.println("normal flow...");
 }
}
```

Test it Now

Output:

```
exception handled
normal flow...
```

Case 2: Declare Exception

Let's see examples for both the scenario.

A) If exception does not occur

Testthrows3.java

```
import java.io.*;
class M{
 void method()throws IOException{
  System.out.println("device operation performed");
 }
}
class Testthrows3{
  public static void main(String args[])throws IOException{//declare exception
    M m=new M();
    m.method();

   System.out.println("normal flow...");
  }
}
```

[Test it Now](#)

Output:

```
device operation performed
normal flow...
```

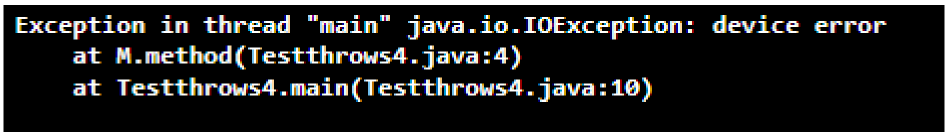
B) If exception occurs

Testthrows4.java

```
import java.io.*;
class M{
 void method()throws IOException{
  throw new IOException("device error");
 }
}
class Testthrows4{
   public static void main(String args[])throws IOException{//declare exception
     M m=new M();
     m.method();

    System.out.println("normal flow...");
  }
}
```

[Test it Now](#)

Output:



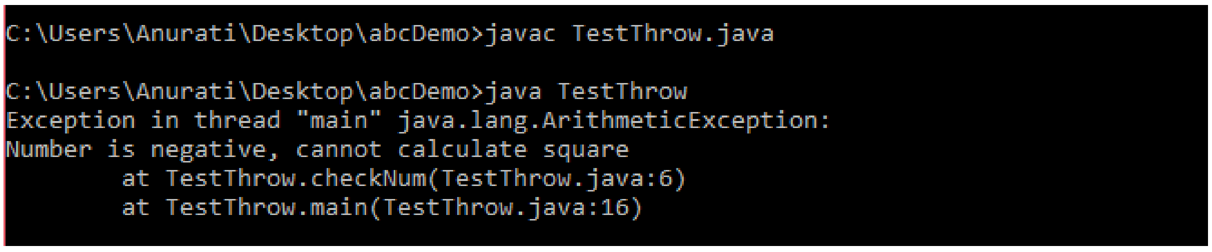
```
Exception in thread "main" java.io.IOException: device error
    at M.method(Testthrows4.java:4)
    at Testthrows4.main(Testthrows4.java:10)
```

Java throw Example

TestThrow.java

```
public class TestThrow {
   //defining a method
   public static void checkNum(int num) {
      if (num < 1) {
         throw new ArithmeticException("\nNumber is negative, cannot calculate square");
      }
      else {
         System.out.println("Square of " + num + " is " + (num*num));
      }
   }
   //main method
   public static void main(String[] args) {
         TestThrow obj = new TestThrow();
         obj.checkNum(-3);
         System.out.println("Rest of the code..");
   }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrow
Exception in thread "main" java.lang.ArithmeticException:
Number is negative, cannot calculate square
        at TestThrow.checkNum(TestThrow.java:6)
        at TestThrow.main(TestThrow.java:16)
```

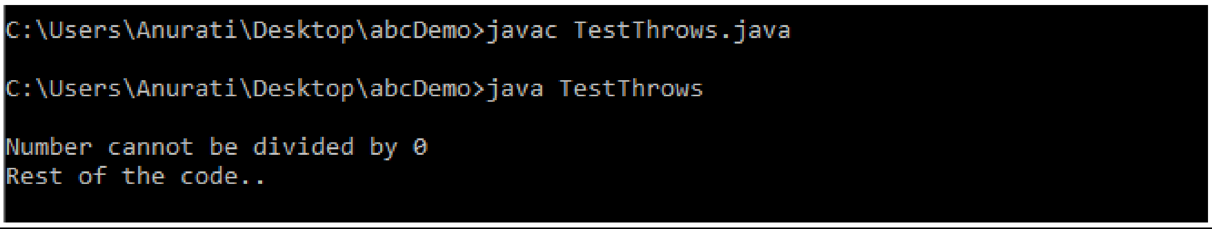
Java throws Example

TestThrows.java

```
public class TestThrows {
   //defining a method
   public static int divideNum(int m, int n) throws ArithmeticException {
      int div = m / n;
      return div;
```

```
    }
    //main method
    public static void main(String[] args) {
        TestThrows obj = new TestThrows();
        try {
            System.out.println(obj.divideNum(45, 0));
        }
        catch (ArithmeticException e){
            System.out.println("\nNumber cannot be divided by 0");
        }

        System.out.println("Rest of the code..");
    }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrows.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrows

Number cannot be divided by 0
Rest of the code..
```

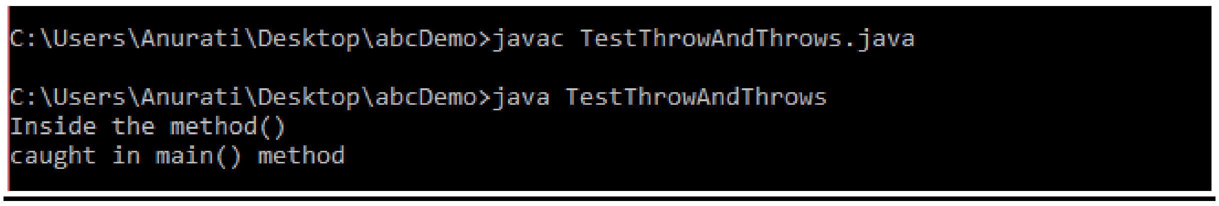
Java throw and throws Example

TestThrowAndThrows.java

```
public class TestThrowAndThrows
{
    // defining a user-defined method
    // which throws ArithmeticException
    static void method() throws ArithmeticException
    {
        System.out.println("Inside the method()");
        throw new ArithmeticException("throwing ArithmeticException");
    }
```

```
 //main method
 public static void main(String args[])
 {
   try
   {
     method();
   }
   catch(ArithmeticException e)
   {
     System.out.println("caught in main() method");
   }
 }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrowAndThrows.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrowAndThrows
Inside the method()
caught in main() method
```

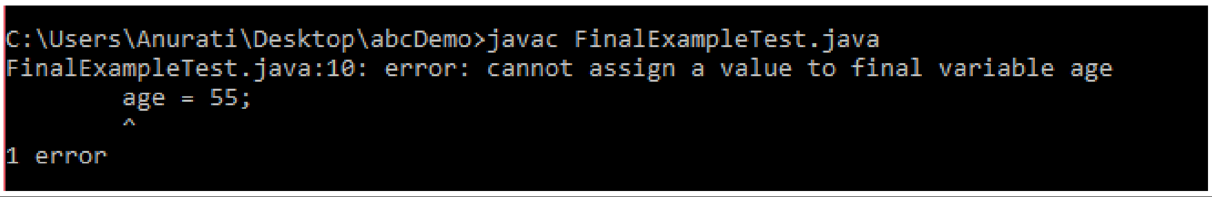
Java final Example

FinalExampleTest.java

```
public class FinalExampleTest {
  //declaring final variable
  final int age = 18;
  void display() {

  // reassigning value to age variable
  // gives compile time error
  age = 55;
  }

```

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

    FinalExampleTest obj = new FinalExampleTest();
    // gives compile time error
    obj.display();
    }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac FinalExampleTest.java
FinalExampleTest.java:10: error: cannot assign a value to final variable age
        age = 55;
        ^
1 error
```

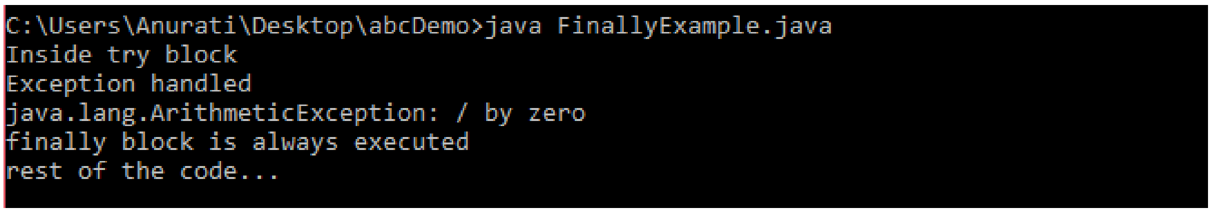
Java finally Example

FinallyExample.java

```
public class FinallyExample {
      public static void main(String args[]){
      try {
        System.out.println("Inside try block");
      // below code throws divide by zero exception
       int data=25/0;
       System.out.println(data);
      }
      // handles the Arithmetic Exception / Divide by zero exception
      catch (ArithmeticException e){
        System.out.println("Exception handled");
        System.out.println(e);
      }
      // executes regardless of exception occurred or not
      finally {
        System.out.println("finally block is always executed");
```

```
    }
    System.out.println("rest of the code...");
    }
  }
```

Output:



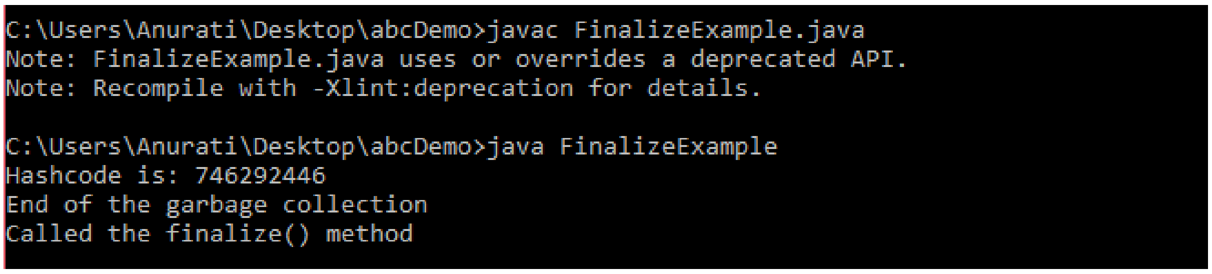
```
C:\Users\Anurati\Desktop\abcDemo>java FinallyExample.java
Inside try block
Exception handled
java.lang.ArithmeticException: / by zero
finally block is always executed
rest of the code...
```

Java finalize Example

FinalizeExample.java

```
public class FinalizeExample {
     public static void main(String[] args)
    {
        FinalizeExample obj = new FinalizeExample();
        // printing the hashcode
        System.out.println("Hashcode is: " + obj.hashCode());
        obj = null;
        // calling the garbage collector using gc()
        System.gc();
        System.out.println("End of the garbage collection");
    }
  // defining the finalize method
   protected void finalize()
    {
        System.out.println("Called the finalize() method");
    }
}
```

Output:



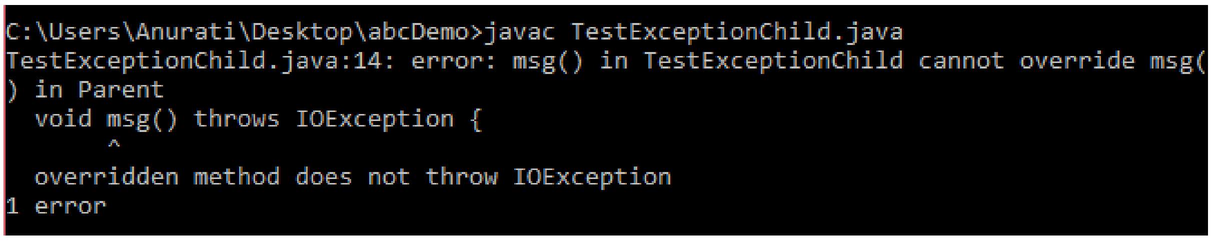
```
C:\Users\Anurati\Desktop\abcDemo>javac FinalizeExample.java
Note: FinalizeExample.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\Anurati\Desktop\abcDemo>java FinalizeExample
Hashcode is: 746292446
End of the garbage collection
Called the finalize() method
```

Exception Handling with Method Overriding in Java

TestExceptionChild.java

```
import java.io.*;
class Parent{

  // defining the method
  void msg() {
    System.out.println("parent method");
    }
}

public class TestExceptionChild extends Parent{

  // overriding the method in child class
  // gives compile time error
  void msg() throws IOException {
    System.out.println("TestExceptionChild");
  }

  public static void main(String args[]) {
   Parent p = new TestExceptionChild();
   p.msg();
  }
}
```

Output:

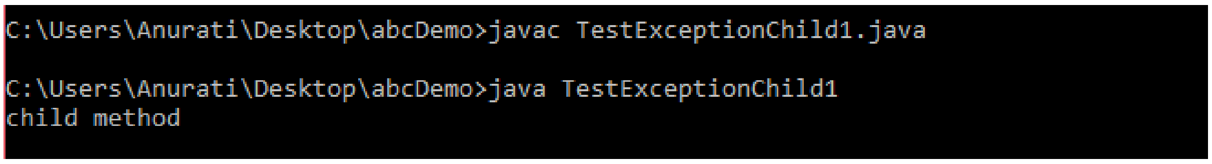

```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild.java
TestExceptionChild.java:14: error: msg() in TestExceptionChild cannot override msg(
) in Parent
  void msg() throws IOException {
       ^
  overridden method does not throw IOException
1 error
```

Rule 2: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.

TestExceptionChild1.java

```
import java.io.*;
class Parent{
  void msg() {
    System.out.println("parent method");
  }
}

class TestExceptionChild1 extends Parent{
  void msg()throws ArithmeticException {
    System.out.println("child method");
  }

  public static void main(String args[]) {
   Parent p = new TestExceptionChild1();
   p.msg();
  }
}
```

Output:


```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild1.java

C:\Users\Anurati\Desktop\abcDemo>java TestExceptionChild1
child method
```

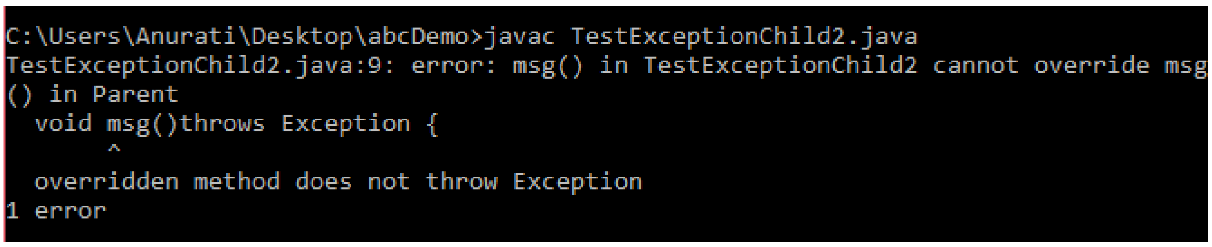
If the superclass method declares an exception

Example in case subclass overridden method declares parent exception

TestExceptionChild2.java

```
import java.io.*;
class Parent{
  void msg()throws ArithmeticException {
    System.out.println("parent method");
  }
}

public class TestExceptionChild2 extends Parent{
  void msg()throws Exception {
    System.out.println("child method");
  }

  public static void main(String args[]) {
   Parent p = new TestExceptionChild2();

   try {
   p.msg();
   }
   catch (Exception e){}

  }
}
```

Output:



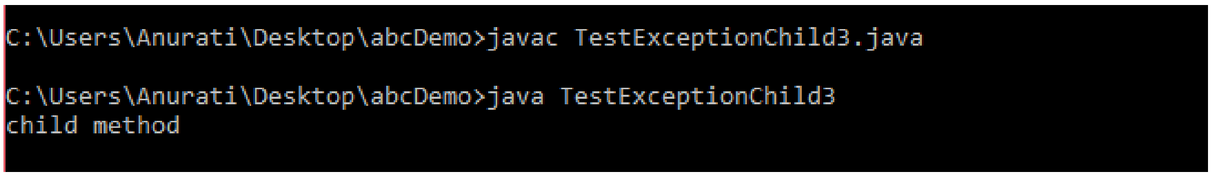
```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild2.java
TestExceptionChild2.java:9: error: msg() in TestExceptionChild2 cannot override msg
() in Parent
  void msg()throws Exception {
       ^
  overridden method does not throw Exception
1 error
```

Example in case subclass overridden method declares same exception

TestExceptionChild3.java

```
import java.io.*;
class Parent{
  void msg() throws Exception {
    System.out.println("parent method");
  }
}

public class TestExceptionChild3 extends Parent {
  void msg()throws Exception {
    System.out.println("child method");
  }

  public static void main(String args[]){
   Parent p = new TestExceptionChild3();

   try {
   p.msg();
   }
   catch(Exception e) {}
  }
}
```

Output:



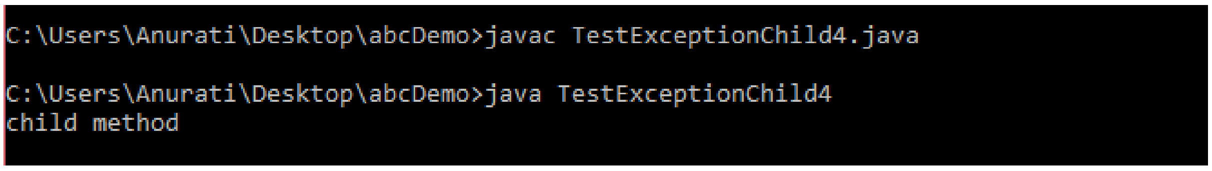
```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild3.java

C:\Users\Anurati\Desktop\abcDemo>java TestExceptionChild3
child method
```

Example in case subclass overridden method declares subclass exception

TestExceptionChild4.java

```
import java.io.*;
class Parent{
  void msg()throws Exception {
    System.out.println("parent method");
  }
}

class TestExceptionChild4 extends Parent{
  void msg()throws ArithmeticException {
    System.out.println("child method");
  }

  public static void main(String args[]){
   Parent p = new TestExceptionChild4();

   try {
   p.msg();
   }
   catch(Exception e) {}
  }
}
```

Output:



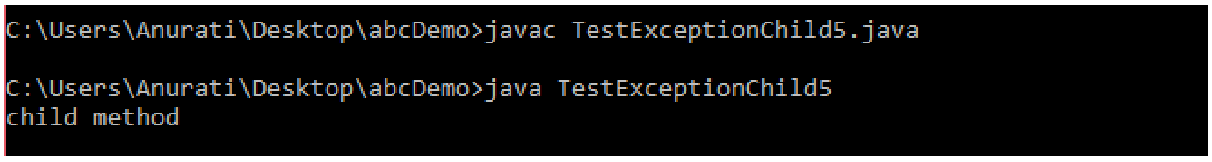
```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild4.java

C:\Users\Anurati\Desktop\abcDemo>java TestExceptionChild4
child method
```

Example in case subclass overridden method declares no exception

TestExceptionChild5.java

```
import java.io.*;
class Parent {
  void msg()throws Exception{
    System.out.println("parent method");
  }
}

class TestExceptionChild5 extends Parent{
  void msg() {
    System.out.println("child method");
  }

  public static void main(String args[]){
   Parent p = new TestExceptionChild5();

   try {
   p.msg();
   }
   catch(Exception e) {}

  }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild5.java

C:\Users\Anurati\Desktop\abcDemo>java TestExceptionChild5
child method
```

Java Custom Exception

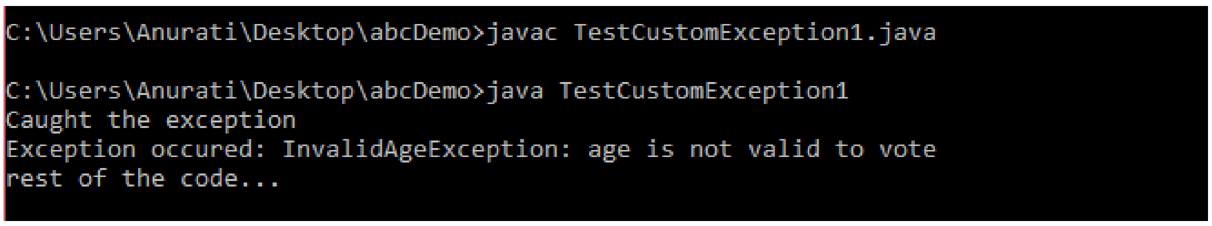
TestCustomException1.java

```
// class representing custom exception
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
```

```
      super(str);
   }
}

// class that uses custom exception InvalidAgeException
public class TestCustomException1
{

   // method to check the age
   static void validate (int age) throws InvalidAgeException{
     if(age < 18){

      // throw an object of user defined exception
      throw new InvalidAgeException("age is not valid to vote");
   }
     else {
      System.out.println("welcome to vote");
      }
    }

   // main method
   public static void main(String args[])
   {
      try
      {
         // calling the method
         validate(13);
      }
      catch (InvalidAgeException ex)
      {
         System.out.println("Caught the exception");
```

```

            // printing the message from InvalidAgeException object
            System.out.println("Exception occured: " + ex);
        }

        System.out.println("rest of the code...");
    }
}
```

Output:



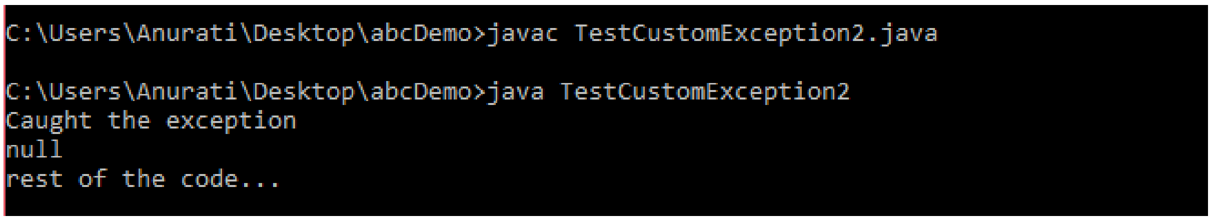
```
C:\Users\Anurati\Desktop\abcDemo>javac TestCustomException1.java

C:\Users\Anurati\Desktop\abcDemo>java TestCustomException1
Caught the exception
Exception occured: InvalidAgeException: age is not valid to vote
rest of the code...
```

Example 2:

TestCustomException2.java

```
// class representing custom exception
class MyCustomException extends Exception
{

}

// class that uses custom exception MyCustomException
public class TestCustomException2
{
    // main method
    public static void main(String args[])
    {
        try
        {
            // throw an object of user defined exception
            throw new MyCustomException();
        }
        catch (MyCustomException ex)
        {
            System.out.println("Caught the exception");
            System.out.println(ex.getMessage());
        }

        System.out.println("rest of the code...");
    }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestCustomException2.java

C:\Users\Anurati\Desktop\abcDemo>java TestCustomException2
Caught the exception
null
rest of the code...
```

Java Member Inner class

TestMemberOuter1.java

```
class TestMemberOuter1{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
  TestMemberOuter1 obj=new TestMemberOuter1();
  TestMemberOuter1.Inner in=obj.new Inner();
  in.msg();
 }
}
```

Test it Now

Output:

data is 30

Java Anonymous inner class

TestAnonymousInner.java

```
abstract class Person{
 abstract void eat();
}
class TestAnonymousInner{
 public static void main(String args[]){
  Person p=new Person(){
  void eat(){System.out.println("nice fruits");}
  };
  p.eat();
 }
}
```

[Test it Now]

Output:

nice fruits

Java anonymous inner class example using interface

```
interface Eatable{
 void eat();
}
class TestAnnonymousInner1{
 public static void main(String args[]){
 Eatable e=new Eatable(){
  public void eat(){System.out.println("nice fruits");}
 };
 e.eat();
 }
}
```

[Test it Now]

Output:nice fruits

Java Local inner class

Java local inner class example

LocalInner1.java

```
public class localInner1{
 private int data=30;//instance variable
 void display(){
  class Local{
   void msg(){System.out.println(data);}
  }
  Local l=new Local();
  l.msg();
 }
 public static void main(String args[]){
  localInner1 obj=new localInner1();
  obj.display();
 }
}
```

Test it Now

Output:

```
30
```

Example of local inner class with local variable

LocalInner2.java

```
class localInner2{
 private int data=30;//instance variable
 void display(){
  int value=50;//local variable must be final till jdk 1.7 only
  class Local{
   void msg(){System.out.println(value);}
  }
  Local l=new Local();
  l.msg();
 }
 public static void main(String args[]){
  localInner2 obj=new localInner2();
  obj.display();
 }
}
```

[Test it Now]()

Output:

```
50
```

Java static nested class

Java static nested class example with instance method

TestOuter1.java

```
class TestOuter1{
  static int data=30;
  static class Inner{
   void msg(){System.out.println("data is "+data);}
  }
  public static void main(String args[]){
  TestOuter1.Inner obj=new TestOuter1.Inner();
  obj.msg();
  }
}
```

[Test it Now]()

Output:

```
data is 30
```

TestOuter2.java

```
public class TestOuter2{
  static int data=30;
  static class Inner{
   static void msg(){System.out.println("data is "+data);}
  }
  public static void main(String args[]){
  TestOuter2.Inner.msg();//no need to create the instance of static nested class
  }
}
```

Test it Now

Output:

```
data is 30
```

Java Nested Interface

TestNestedInterface1.java

```
interface Showable{
  void show();
  interface Message{
   void msg();
  }
}
class TestNestedInterface1 implements Showable.Message{
 public void msg(){System.out.println("Hello nested interface");}

 public static void main(String args[]){
  Showable.Message message=new TestNestedInterface1();//upcasting here
  message.msg();
 }
}
```

Test it Now

Output:

```
hello nested interface
```

TestNestedInterface2.java

```
class A{
  interface Message{
   void msg();
  }
}

class TestNestedInterface2 implements A.Message{
 public void msg(){System.out.println("Hello nested interface");}

 public static void main(String args[]){
  A.Message message=new TestNestedInterface2();//upcasting here
  message.msg();
 }
}
```

Test it Now

Output:

```
hello nested interface
```

Multithreading in Java

Life cycle of a Thread (Thread States)

FileName: ThreadState.java

```
// ABC class implements the interface Runnable
class ABC implements Runnable
{
public void run()
{

// try-catch block
try
{
// moving thread t2 to the state timed waiting
Thread.sleep(100);
}
catch (InterruptedException ie)
{
ie.printStackTrace();
}


System.out.println("The state of thread t1 while it invoked the method join() on thread t2 -
"+ ThreadState.t1.getState());

// try-catch block
try
{
Thread.sleep(200);
}
catch (InterruptedException ie)
{
ie.printStackTrace();
}
}
}

// ThreadState class implements the interface Runnable
public class ThreadState implements Runnable
{
public static Thread t1;
public static ThreadState obj;

// main method
public static void main(String argvs[])
{
// creating an object of the class ThreadState
obj = new ThreadState();
t1 = new Thread(obj);

```

```
// thread t1 is spawned
// The thread t1 is currently in the NEW state.
System.out.println("The state of thread t1 after spawning it - " + t1.getState());

// invoking the start() method on
// the thread t1
t1.start();

// thread t1 is moved to the Runnable state
System.out.println("The state of thread t1 after invoking the method start() on it - " + t1.getState());
}

public void run()
{
ABC myObj = new ABC();
Thread t2 = new Thread(myObj);

// thread t2 is created and is currently in the NEW state.
System.out.println("The state of thread t2 after spawning it - "+ t2.getState());
t2.start();

// thread t2 is moved to the runnable state
System.out.println("the state of thread t2 after calling the method start() on it - " + t2.getState());

// try-catch block for the smooth flow of the  program
try
{
// moving the thread t1 to the state timed waiting
Thread.sleep(200);
}
```

```
catch (InterruptedException ie)
{
ie.printStackTrace();
}

System.out.println("The state of thread t2 after invoking the method sleep() on it - "+ t2.getState() );

// try-catch block for the smooth flow of the  program
try
{
// waiting for thread t2 to complete its execution
t2.join();
}
catch (InterruptedException ie)
{
ie.printStackTrace();
}
System.out.println("The state of thread t2 when it has completed it's execution - " + t2.getState());
}

}
```

Output:

```
The state of thread t1 after spawning it - NEW
The state of thread t1 after invoking the method start() on it - RUNNABLE
The state of thread t2 after spawning it - NEW
the state of thread t2 after calling the method start() on it - RUNNABLE
The state of thread t1 while it invoked the method join() on thread t2 -TIMED_WAITING
The state of thread t2 after invoking the method sleep() on it - TIMED_WAITING
The state of thread t2 when it has completed it's execution - TERMINATED
```

Java Threads | How to create a thread in Java

Thread Creation

1) Creating Thread by Extending Thread Class

File Name: Multi.java

```java
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

Output:

```
thread is running...
```

---

2) Java Thread Example by implementing Runnable interface

FileName: Multi3.java

```java
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

Output:

```
thread is running...
```

Using the Thread Class: Thread(String Name)

FileName: MyThread1.java

```
public class MyThread1
{
// Main method
public static void main(String argvs[])
{
// creating an object of the Thread class using the constructor Thread(String name)
Thread t= new Thread("My first thread");

// the start() method moves the thread to the active state
t.start();
// getting the thread name by invoking the getName() method
String str = t.getName();
System.out.println(str);
}
}
```

Output:

```
My first thread
```

4) Using the Thread Class: Thread(Runnable r, String name)

Observe the following program.

FileName: MyThread2.java

```
public class MyThread2 implements Runnable
{
public void run()
{
System.out.println("Now the thread is running ...");
}

// main method
public static void main(String argvs[])
{
```

```
// creating an object of the class MyThread2
Runnable r1 = new MyThread2();

// creating an object of the class Thread using Thread(Runnable r, String name)
Thread th1 = new Thread(r1, "My new thread");

// the start() method moves the thread to the active state
th1.start();

// getting the thread name by invoking the getName() method
String str = th1.getName();
System.out.println(str);
}
}
```

Output:

```
My new thread
Now the thread is running ..
```

Thread Scheduler in Java

Thread.sleep() in Java with Examples

FileName: TestSleepMethod1.java

```
class TestSleepMethod1 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
  // the thread will sleep for the 500 milli seconds
    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();


 t1.start();
 t2.start();
 }
}
```

Output:

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

Example of the sleep() Method in Java : on the main thread

FileName: TestSleepMethod2.java

```
// important import statements
import java.lang.Thread;
import java.io.*;


public class TestSleepMethod2
{
   // main method
public static void main(String argvs[])
{

try {
for (int j = 0; j < 5; j++)
{

```

```
// The main thread sleeps for the 1000 milliseconds, which is 1 sec
// whenever the loop runs
Thread.sleep(1000);

// displaying the value of the variable
System.out.println(j);
}
}
catch (Exception expn)
{
// catching the exception
System.out.println(expn);
}
}
}
```

Output:

```
0
1
2
3
4
```

Example of the sleep() Method in Java: When the sleeping time is -ive

FileName: TestSleepMethod3.java

```
// important import statements
import java.lang.Thread;
import java.io.*;

public class TestSleepMethod3
{
// main method
public static void main(String argvs[])
{
// we can also use throws keyword followed by
// exception name for throwing the exception
try
{
for (int j = 0; j < 5; j++)
{

// it throws the exception IllegalArgumentException
// as the time is -ive which is -100
Thread.sleep(-100);

// displaying the variable's value
System.out.println(j);
}
}
catch (Exception expn)
{

// the exception iscaught here
System.out.println(expn);
}
}
}
```

Output:

```
java.lang.IllegalArgumentException: timeout value is negative
```

Can we start a thread twice

```
public class TestThreadTwice1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestThreadTwice1 t1=new TestThreadTwice1();
  t1.start();
  t1.start();
 }
}
```

[Test it Now]

Output:

```
running
Exception in thread "main" java.lang.IllegalThreadStateException
```

What if we call Java run() method directly instead start() method?

FileName: TestCallRun1.java

```
class TestCallRun1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestCallRun1 t1=new TestCallRun1();
  t1.run();//fine, but does not start a separate call stack
 }
}
```

[Test it Now]

Output:

```
running...
```

Problem if you direct call run() method

FileName: TestCallRun2.java

```
class TestCallRun2 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
```

```
     System.out.println(i);
   }
  }
  public static void main(String args[]){
   TestCallRun2 t1=new TestCallRun2();
   TestCallRun2 t2=new TestCallRun2();

   t1.run();
   t2.run();
  }
 }
```

Test it Now

Output:

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

Java join() method

Example of join() Method in Java

The following program shows the usage of the join() method.

FileName: ThreadJoinExample.java

```
// A Java program for understanding
// the joining of threads

// import statement
import java.io.*;

```

```
// The ThreadJoin class is the child class of the class Thread
class ThreadJoin extends Thread
{
// overriding the run method
public void run()
{
for (int j = 0; j < 2; j++)
{
try
{
// sleeping the thread for 300 milli seconds
Thread.sleep(300);
System.out.println("The current thread name is: " + Thread.currentThread().getName());
}
// catch block for catching the raised exception
catch(Exception e)
{
System.out.println("The exception has been caught: " + e);
}
System.out.println( j );
}
}
}

public class ThreadJoinExample
{
// main method
public static void main (String argvs[])
{

// creating 3 threads
```

```
ThreadJoin th1 = new ThreadJoin();
ThreadJoin th2 = new ThreadJoin();
ThreadJoin th3 = new ThreadJoin();

// thread th1 starts
th1.start();

// starting the second thread after when
// the first thread th1 has ended or died.
try
{
System.out.println("The current thread name is: "+ Thread.currentThread().getName());

// invoking the join() method
th1.join();
}

// catch block for catching the raised exception
catch(Exception e)
{
System.out.println("The exception has been caught " + e);
}

// thread th2 starts
th2.start();

// starting the th3 thread after when the thread th2 has ended or died.
try
{
System.out.println("The current thread name is: " + Thread.currentThread().getName());
th2.join();
}

// catch block for catching the raised exception
catch(Exception e)
{
System.out.println("The exception has been caught " + e);
}

// thread th3 starts
th3.start();
}
}
```

Output:

Advertisement

```
The current thread name is: main
The current thread name is: Thread - 0
0
The current thread name is: Thread - 0
1
The current thread name is: main
The current thread name is: Thread - 1
0
The current thread name is: Thread - 1
1
The current thread name is: Thread - 2
0
The current thread name is: Thread - 2
1
```

Example of join() Method in Java

The following program shows the usage of the join() method.

FileName: ThreadJoinExample.java

FileName: ThreadJoinExample1.java

```
class ABC extends Thread
{
Thread threadToInterrupt;
// overriding the run() method
public void run()
{
// invoking the method interrupt
threadToInterrupt.interrupt();
}
}


public class ThreadJoinExample1
{
// main method
public static void main(String[] argvs)
{
try
{
// creating an object of the class ABC
ABC th1 = new ABC();

th1.threadToInterrupt = Thread.currentThread();
th1.start();

// invoking the join() method leads
// to the generation of InterruptedException
th1.join();
}
catch (InterruptedException ex)
{
System.out.println("The exception has been caught. " + ex);
}
}
}
```

Output:

```
The exception has been caught. java.lang.InterruptedException
```

Filename: TestJoinMethod1.java

```
class TestJoinMethod1 extends Thread{
 public void run(){
  for(int i=1;i<=5;i++){
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
  System.out.println(i);
  }
 }
public static void main(String args[]){
 TestJoinMethod1 t1=new TestJoinMethod1();
 TestJoinMethod1 t2=new TestJoinMethod1();
 TestJoinMethod1 t3=new TestJoinMethod1();
 t1.start();
 try{
  t1.join();
 }catch(Exception e){System.out.println(e);}

 t2.start();
 t3.start();
 }
}
```

Output:

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

We can see in the above example, when t1 completes its task then t2 and t3 starts executing.

join(long miliseconds) Method Example

Filename: TestJoinMethod2.jav

```
class TestJoinMethod2 extends Thread{
 public void run(){
  for(int i=1;i<=5;i++){
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
  System.out.println(i);
  }
 }
public static void main(String args[]){
 TestJoinMethod2 t1=new TestJoinMethod2();
 TestJoinMethod2 t2=new TestJoinMethod2();
TestJoinMethod2 t3=new TestJoinMethod2();
t1.start();
try{
 t1.join(1500);
}catch(Exception e){System.out.println(e);}

t2.start();
t3.start();
}
}
```

Output:

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

Naming Thread and Current Thread

Example of naming a thread : Using setName() Method

FileName: TestMultiNaming1.java

```
class TestMultiNaming1 extends Thread{
 public void run(){
  System.out.println("running...");
 }
 public static void main(String args[]){
  TestMultiNaming1 t1=new TestMultiNaming1();
  TestMultiNaming1 t2=new TestMultiNaming1();
  System.out.println("Name of t1:"+t1.getName());
  System.out.println("Name of t2:"+t2.getName());

  t1.start();
  t2.start();

  t1.setName("Sonoo Jaiswal");
  System.out.println("After changing name of t1:"+t1.getName());
 }
}
```

Test it Now

Output:

```
Name of t1:Thread-0
Name of t2:Thread-1
After changing name of t1:Sonoo Jaiswal
running...
running...
```

Example of naming a thread : Without Using setName() Method

One can also set the name of a thread at the time of the creation of a thread, without using the setName() method. Observe the following code.

FileName: ThreadNamingExample.java

```
// A Java program that shows how one can
// set the name of a thread at the time
// of creation of the thread

// import statement
```

```
import java.io.*;

// The ThreadNameClass is the child class of the class Thread
class ThreadName extends Thread
{

// constructor of the class
ThreadName(String threadName)
{
// invoking the constructor of
// the superclass, which is Thread class.
super(threadName);
}

// overriding the method run()
public void run()
{
System.out.println(" The thread is executing....");
}
}

public class ThreadNamingExample
{
// main method
public static void main (String argvs[])
{
// creating two threads and settting their name
// using the contructor of the class
ThreadName th1 = new ThreadName("JavaTpoint1");
ThreadName th2 = new ThreadName("JavaTpoint2");

// invoking the getName() method to get the names
// of the thread created above
System.out.println("Thread - 1: " + th1.getName());
System.out.println("Thread - 2: " + th2.getName());


// invoking the start() method on both the threads
th1.start();
th2.start();
}
}
```

Output:

```
Thread - 1: JavaTpoint1
Thread - 2: JavaTpoint2
The thread is executing....
The thread is executing....
```

Current Thread

The currentThread() method returns a reference of the currently executing thread.

```
public static Thread currentThread()
```

Example of currentThread() method

FileName: TestMultiNaming2.java

```
class TestMultiNaming2 extends Thread{
 public void run(){
  System.out.println(Thread.currentThread().getName());
 }
 public static void main(String args[]){
  TestMultiNaming2 t1=new TestMultiNaming2();
  TestMultiNaming2 t2=new TestMultiNaming2();

  t1.start();
  t2.start();
 }
}
```

Test it Now

Output:

```
Thread-0
Thread-1
```

Priority of a Thread (Thread Priority)

Example of priority of a Thread:

FileName: ThreadPriorityExample.java

```
// Importing the required classes
import java.lang.*;

public class ThreadPriorityExample extends Thread
{

// Method 1
// Whenever the start() method is called by a thread
// the run() method is invoked
public void run()
{
// the print statement
System.out.println("Inside the run() method");
}

// the main method
public static void main(String argvs[])
{
// Creating threads with the help of ThreadPriorityExample class
ThreadPriorityExample th1 = new ThreadPriorityExample();
ThreadPriorityExample th2 = new ThreadPriorityExample();
ThreadPriorityExample th3 = new ThreadPriorityExample();

// We did not mention the priority of the thread.
// Therefore, the priorities of the thread is 5, the default value

// 1st Thread
// Displaying the priority of the thread
// using the getPriority() method
System.out.println("Priority of the thread th1 is : " + th1.getPriority());

// 2nd Thread
// Display the priority of the thread
System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// 3rd Thread
// // Display the priority of the thread
System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// Setting priorities of above threads by
// passing integer arguments
th1.setPriority(6);
th2.setPriority(3);
th3.setPriority(9);

// 6
System.out.println("Priority of the thread th1 is : " + th1.getPriority());

// 3
System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// 9
```

```
System.out.println("Priority of the thread th3 is : " + th3.getPriority());

// Main thread

// Displaying name of the currently executing thread
System.out.println("Currently Executing The Thread : " + Thread.currentThread().getName());

System.out.println("Priority of the main thread is : " + Thread.currentThread().getPriority());

// Priority of the main thread is 10 now
Thread.currentThread().setPriority(10);

System.out.println("Priority of the main thread is : " + Thread.currentThread().getPriority());
}
}
```

Output:

```
Priority of the thread th1 is : 5
Priority of the thread th2 is : 5
Priority of the thread th2 is : 5
Priority of the thread th1 is : 6
Priority of the thread th2 is : 3
Priority of the thread th3 is : 9
Currently Executing The Thread : main
Priority of the main thread is : 5
Priority of the main thread is : 10
```

FileName: ThreadPriorityExample1.java

```
// importing the java.lang package
import java.lang.*;

```

```
public class ThreadPriorityExample1 extends Thread
{

// Method 1
// Whenever the start() method is called by a thread
// the run() method is invoked
public void run()
{
// the print statement
System.out.println("Inside the run() method");
}


// the main method
public static void main(String argvs[])
{

// Now, priority of the main thread is set to 7
Thread.currentThread().setPriority(7);

// the current thread is retrieved
// using the currentThread() method

// displaying the main thread priority
// using the getPriority() method of the Thread class
System.out.println("Priority of the main thread is : " + Thread.currentThread().getPriority()
);

// creating a thread by creating an object of the class ThreadPriorityExample1
ThreadPriorityExample1 th1 = new ThreadPriorityExample1();

// th1 thread is the child of the main thread
// therefore, the th1 thread also gets the priority 7

// Displaying the priority of the current thread
System.out.println("Priority of the thread th1 is : " + th1.getPriority());
}
}
```

Output:

```
Priority of the main thread is : 7
Priority of the thread th1 is : 7
```

Example of IllegalArgumentException

FileName: IllegalArgumentException.java

```
// importing the java.lang package
import java.lang.*;

public class IllegalArgumentException extends Thread
{

// the main method
public static void main(String argvs[])
{

// Now, priority of the main thread is set to 17, which is greater than 10
Thread.currentThread().setPriority(17);

// The current thread is retrieved
// using the currentThread() method

// displaying the main thread priority
// using the getPriority() method of the Thread class
System.out.println("Priority of the main thread is : " + Thread.currentThread().getPriority()
);

}
}
```

When we execute the above program, we get the following exception:

```
Exception in thread "main" java.lang.IllegalArgumentException
at java.base/java.lang.Thread.setPriority(Thread.java:1141)
at IllegalArgumentException.main(IllegalArgumentException.java:12)
```

Daemon Thread in Java

Simple example of Daemon thread in java

File: MyThread.java

```
public class TestDaemonThread1 extends Thread{
 public void run(){
  if(Thread.currentThread().isDaemon()){//checking for daemon thread
   System.out.println("daemon thread work");
  }
  else{
  System.out.println("user thread work");
 }
 }
 public static void main(String[] args){
  TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
  TestDaemonThread1 t2=new TestDaemonThread1();
  TestDaemonThread1 t3=new TestDaemonThread1();

  t1.setDaemon(true);//now t1 is daemon thread

  t1.start();//starting threads
  t2.start();
  t3.start();
 }
}
```

[Test it Now]()

Output:

```
daemon thread work
user thread work
user thread work
```

File: MyThread.java

```
class TestDaemonThread2 extends Thread{
 public void run(){
  System.out.println("Name: "+Thread.currentThread().getName());
  System.out.println("Daemon: "+Thread.currentThread().isDaemon());
 }

 public static void main(String[] args){
  TestDaemonThread2 t1=new TestDaemonThread2();
  TestDaemonThread2 t2=new TestDaemonThread2();
  t1.start();
  t1.setDaemon(true);//will throw exception here
  t2.start();
 }
}
```

[Test it Now]()

Output:

```
exception in thread main: java.lang.IllegalThreadStateException
```

Java Thread Pool

File: TestThreadPool.java

```
public class TestThreadPool {
     public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(5);//creating a pool of 5 threads
        for (int i = 0; i < 10; i++) {
            Runnable worker = new WorkerThread("" + i);
            executor.execute(worker);//calling execute method of ExecutorService
          }
        executor.shutdown();
        while (!executor.isTerminated()) {   }

        System.out.println("Finished all threads");
    }
}
```

Output:

```
pool-1-thread-1 (Start) message = 0
pool-1-thread-2 (Start) message = 1
pool-1-thread-3 (Start) message = 2
pool-1-thread-5 (Start) message = 4
pool-1-thread-4 (Start) message = 3
pool-1-thread-2 (End)
pool-1-thread-2 (Start) message = 5
pool-1-thread-1 (End)
pool-1-thread-1 (Start) message = 6
pool-1-thread-3 (End)
pool-1-thread-3 (Start) message = 7
pool-1-thread-4 (End)
pool-1-thread-4 (Start) message = 8
pool-1-thread-5 (End)
pool-1-thread-5 (Start) message = 9
pool-1-thread-2 (End)
pool-1-thread-1 (End)
pool-1-thread-4 (End)
pool-1-thread-3 (End)
pool-1-thread-5 (End)
```

Finished all threads

download this example

Thread Pool Example: 2

Let's see another example of the thread pool.

FileName: ThreadPoolExample.java

```
// important import statements
import java.util.Date;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.text.SimpleDateFormat;


class Tasks implements Runnable
{
private String taskName;

// constructor of the class Tasks
public Tasks(String str)
{
// initializing the field taskName
taskName = str;
}

// Printing the task name and then sleeps for 1 sec
// The complete process is getting repeated five times
public void run()
{
try
{
for (int j = 0; j <= 5; j++)
```

```
{
if (j == 0)
{
Date dt = new Date();
SimpleDateFormat sdf = new SimpleDateFormat("hh : mm : ss");

//prints the initialization time for every task
System.out.println("Initialization time for the task name: "+ taskName + " = " + sdf.format(dt));

}
else
{
Date dt = new Date();
SimpleDateFormat sdf = new SimpleDateFormat("hh : mm : ss");

// prints the execution time for every task
System.out.println("Time of execution for the task name: " + taskName + " = " +sdf.format(dt));

}

// 1000ms = 1 sec
Thread.sleep(1000);
}

System.out.println(taskName + " is complete.");
}

catch(InterruptedException ie)
{
ie.printStackTrace();
```

```
}
}
}

public class ThreadPoolExample
{
// Maximum number of threads in the thread pool
static final int MAX_TH = 3;

// main method
public static void main(String argvs[])
{
// Creating five new tasks
Runnable rb1 = new Tasks("task 1");
Runnable rb2 = new Tasks("task 2");
Runnable rb3 = new Tasks("task 3");
Runnable rb4 = new Tasks("task 4");
Runnable rb5 = new Tasks("task 5");

// creating a thread pool with MAX_TH number of
// threads size the pool size is fixed
ExecutorService pl = Executors.newFixedThreadPool(MAX_TH);

// passes the Task objects to the pool to execute (Step 3)
pl.execute(rb1);
pl.execute(rb2);
pl.execute(rb3);
pl.execute(rb4);
pl.execute(rb5);

// pool is shutdown
```

```java
pl.shutdown();
}
}
```

Output:

```
Initialization time for the task name: task 1 = 06 : 13 : 02
Initialization time for the task name: task 2 = 06 : 13 : 02
Initialization time for the task name: task 3 = 06 : 13 : 02
Time of execution for the task name: task 1 = 06 : 13 : 04
Time of execution for the task name: task 2 = 06 : 13 : 04
Time of execution for the task name: task 3 = 06 : 13 : 04
Time of execution for the task name: task 1 = 06 : 13 : 05
Time of execution for the task name: task 2 = 06 : 13 : 05
Time of execution for the task name: task 3 = 06 : 13 : 05
Time of execution for the task name: task 1 = 06 : 13 : 06
Time of execution for the task name: task 2 = 06 : 13 : 06
Time of execution for the task name: task 3 = 06 : 13 : 06
Time of execution for the task name: task 1 = 06 : 13 : 07
Time of execution for the task name: task 2 = 06 : 13 : 07
Time of execution for the task name: task 3 = 06 : 13 : 07
Time of execution for the task name: task 1 = 06 : 13 : 08
Time of execution for the task name: task 2 = 06 : 13 : 08
Time of execution for the task name: task 3 = 06 : 13 : 08
task 2 is complete.
Initialization time for the task name: task 4 = 06 : 13 : 09
task 1 is complete.
Initialization time for the task name: task 5 = 06 : 13 : 09
task 3 is complete.
Time of execution for the task name: task 4 = 06 : 13 : 10
Time of execution for the task name: task 5 = 06 : 13 : 10
Time of execution for the task name: task 4 = 06 : 13 : 11
Time of execution for the task name: task 5 = 06 : 13 : 11
Time of execution for the task name: task 4 = 06 : 13 : 12
Time of execution for the task name: task 5 = 06 : 13 : 12
Time of execution for the task name: task 4 = 06 : 13 : 13
Time of execution for the task name: task 5 = 06 : 13 : 13
Time of execution for the task name: task 4 = 06 : 13 : 14
Time of execution for the task name: task 5 = 06 : 13 : 14
task 4 is complete.
task 5 is complete.
```

ThreadGroup in Java

ThreadGroup Example

File: ThreadGroupDemo.java

```
public class ThreadGroupDemo implements Runnable{
    public void run() {
          System.out.println(Thread.currentThread().getName());
    }
   public static void main(String[] args) {
      ThreadGroupDemo runnable = new ThreadGroupDemo();
          ThreadGroup tg1 = new ThreadGroup("Parent ThreadGroup");

          Thread t1 = new Thread(tg1, runnable,"one");
          t1.start();
          Thread t2 = new Thread(tg1, runnable,"two");
          t2.start();
          Thread t3 = new Thread(tg1, runnable,"three");
          t3.start();

          System.out.println("Thread Group Name: "+tg1.getName());
         tg1.list();

    }
   }
```

```
Output:
one
two
three
Thread Group Name: Parent ThreadGroup
java.lang.ThreadGroup[name=Parent ThreadGroup,maxpri=10]
```

Thread Pool Methods Example: int activeCount()

Let's see how one can use the method activeCount().

FileName: ActiveCountExample.java

```
// code that illustrates the activeCount() method

// import statement
import java.lang.*;


class ThreadNew extends Thread
{
// constructor of the  class
ThreadNew(String tName, ThreadGroup tgrp)
{
super(tgrp, tName);
start();
}

// overriding the run method
public void run()
{

for (int j = 0; j < 1000; j++)
{
```

```
try
{
Thread.sleep(5);
}
catch (InterruptedException e)
{
System.out.println("The exception has been encountered " + e);
}
}
}
}

public class ActiveCountExample
{
// main method
public static void main(String argvs[])
{
// creating the thread group
ThreadGroup tg = new ThreadGroup("The parent group of threads");

ThreadNew th1 = new ThreadNew("first", tg);
System.out.println("Starting the first");

ThreadNew th2 = new ThreadNew("second", tg);
System.out.println("Starting the second");

// checking the number of active thread by invoking the activeCount() method
System.out.println("The total number of active threads are: " + tg.activeCount());
}
}
```

Output:

```
Starting the first

Starting the second

The total number of active threads are: 2
```

Thread Pool Methods Example: int activeGroupCount()

Now, we will learn how one can use the activeGroupCount() method in the code.

FileName: ActiveGroupCountExample.java

```
// Java code illustrating the activeGroupCount() method

// import statement
import java.lang.*;


class ThreadNew extends Thread
{
// constructor of the  class
ThreadNew(String tName, ThreadGroup tgrp)
{
super(tgrp, tName);
start();
}

// overriding the run() method
public void run()
{

for (int j = 0; j < 100; j++)
{
try
{
Thread.sleep(5);
}
catch (InterruptedException e)
{
System.out.println("The exception has been encountered " + e);
}

}

System.out.println(Thread.currentThread().getName() + " thread has finished executing");
}
}

public class ActiveGroupCountExample
{
// main method
public static void main(String argvs[])
{
// creating the thread group
ThreadGroup tg = new ThreadGroup("The parent group of threads");

ThreadGroup tg1 = new ThreadGroup(tg, "the child group");

ThreadNew th1 = new ThreadNew("the first", tg);
System.out.println("Starting the first");

ThreadNew th2 = new ThreadNew("the second", tg);
System.out.println("Starting the second");

// checking the number of active thread by invoking the activeGroupCount() method
System.out.println("The total number of active thread groups are: " + tg.activeGroupCount
());
```

```
}
}
```

Output:

```
Starting the first
Starting the second
The total number of active thread groups are: 1
the second thread has finished executing
the first thread has finished executing
```

Thread Pool Methods Example: void destroy()

Now, we will learn how one can use the destroy() method in the code.

```
Starting the first
Starting the second
the first thread has finished executing
the second thread has finished executing
the child group is destroyed.
the parent group is destroyed.
```

Thread Pool Methods Example: int enumerate()

Now, we will learn how one can use the enumerate() method in the code.

FileName: EnumerateExample.java

```
// Code illustrating the enumerate() method

// import statement
import java.lang.*;


class ThreadNew extends Thread
{
// constructor of the class
ThreadNew(String tName, ThreadGroup tgrp)
{
super(tgrp, tName);
```

```
start();
}

// overriding the run() method
public void run()
{

for (int j = 0; j < 100; j++)
{
try
{
Thread.sleep(5);
}
catch (InterruptedException e)
{
System.out.println("The exception has been encountered " + e);
}

}

System.out.println(Thread.currentThread().getName() + " thread has finished executing");
}
}

public class EnumerateExample
{
// main method
public static void main(String argvs[]) throws SecurityException, InterruptedException
{
// creating the thread group
ThreadGroup tg = new ThreadGroup("the parent group");

ThreadGroup tg1 = new ThreadGroup(tg, "the child group");

ThreadNew th1 = new ThreadNew("the first", tg);
System.out.println("Starting the first");

ThreadNew th2 = new ThreadNew("the second", tg);
System.out.println("Starting the second");

// returning the number of threads kept in this array
Thread[] grp = new Thread[tg.activeCount()];
int cnt = tg.enumerate(grp);
for (int j = 0; j < cnt; j++)
{
System.out.println("Thread " + grp[j].getName() + " is found.");
}
}
}
```

Advertisement

Output:

```
Starting the first
Starting the second
Thread the first is found.
Thread the second is found.
the first thread has finished executing
the second thread has finished executing
```

Thread Pool Methods Example: int getMaxPriority()

The following code shows the working of the getMaxPriority() method.

FileName: GetMaxPriorityExample.java

```
// Code illustrating the getMaxPriority() method

// import statement
import java.lang.*;


class ThreadNew extends Thread
{
// constructor of the class
ThreadNew(String tName, ThreadGroup tgrp)
{
super(tgrp, tName);
start();
}

// overriding the run() method
public void run()
{

for (int j = 0; j < 100; j++)
{
try
{
Thread.sleep(5);
}
catch (InterruptedException e)
{
System.out.println("The exception has been encountered " + e);
}

}

System.out.println(Thread.currentThread().getName() + " thread has finished executing");
}
}

public class GetMaxPriorityExample
{
// main method
public static void main(String argvs[]) throws SecurityException, InterruptedException
{
// creating the thread group
ThreadGroup tg = new ThreadGroup("the parent group");

ThreadGroup tg1 = new ThreadGroup(tg, "the child group");

ThreadNew th1 = new ThreadNew("the first", tg);
System.out.println("Starting the first");

ThreadNew th2 = new ThreadNew("the second", tg);
System.out.println("Starting the second");

int priority = tg.getMaxPriority();

System.out.println("The maximum priority of the parent ThreadGroup: " + priority);


}
}
```

Output:

```
Starting the first
Starting the second
The maximum priority of the parent ThreadGroup: 10
```

```
the first thread has finished executing
the second thread has finished executing
```

Thread Pool Methods Example: ThreadGroup getParent()

Now, we will learn how one can use the getParent() method in the code.

FileName: GetParentExample.java

```java
// Code illustrating the getParent() method

// import statement
import java.lang.*;


class ThreadNew extends Thread
{
// constructor of the class
ThreadNew(String tName, ThreadGroup tgrp)
{
super(tgrp, tName);
start();
}

// overriding the run() method
public void run()
{

for (int j = 0; j < 100; j++)
{
try
{
Thread.sleep(5);
}
catch (InterruptedException e)
{
System.out.println("The exception has been encountered" + e);
}

}

System.out.println(Thread.currentThread().getName() + " thread has finished executing");
}
}

public class GetMaxPriorityExample
{
// main method
public static void main(String argvs[]) throws SecurityException, InterruptedException
{
// creating the thread group
ThreadGroup tg = new ThreadGroup("the parent group");

ThreadGroup tg1 = new ThreadGroup(tg, "the child group");

ThreadNew th1 = new ThreadNew("the first", tg);
System.out.println("Starting the first");

ThreadNew th2 = new ThreadNew("the second", tg);
System.out.println("Starting the second");

// printing the parent ThreadGroup
// of both child and parent threads
System.out.println("The ParentThreadGroup for " + tg.getName() + " is " + tg.getParent().getName());
System.out.println("The ParentThreadGroup for " + tg1.getName() + " is " + tg1.getParent()
.getName());


}
}
```

Advertisement

Output:

```
Starting the first
Starting the second
The ParentThreadGroup for the parent group is main
The ParentThreadGroup for the child group is the parent group
the first thread has finished executing
the second thread has finished executing
```

Thread Pool Methods Example: void interrupt()

The following program illustrates how one can use the interrupt() method.

FileName: InterruptExample.java

```
// Code illustrating the interrupt() method

// import statement
import java.lang.*;


class ThreadNew extends Thread
{
// constructor of the class
ThreadNew(String tName, ThreadGroup tgrp)
{
super(tgrp, tName);
start();
```

```
}

// overriding the run() method
public void run()
{

for (int j = 0; j < 100; j++)
{
try
{
Thread.sleep(5);
}
catch (InterruptedException e)
{
System.out.println("The exception has been encountered " + e);
}

}

System.out.println(Thread.currentThread().getName() + " thread has finished executing");
}
}

public class InterruptExample
{
// main method
public static void main(String argvs[]) throws SecurityException, InterruptedException
{
// creating the thread group
ThreadGroup tg = new ThreadGroup("the parent group");

ThreadGroup tg1 = new ThreadGroup(tg, "the child group");

ThreadNew th1 = new ThreadNew("the first", tg);
System.out.println("Starting the first");

ThreadNew th2 = new ThreadNew("the second", tg);
System.out.println("Starting the second");

// invoking the interrupt method
tg.interrupt();

}
}
```

Output:

```
Starting the first
Starting the second
The exception has been encountered java.lang.InterruptedException: sleep interrupted
The exception has been encountered java.lang.InterruptedException: sleep interrupted
the second thread has finished executing
the first thread has finished executing
```

Thread Pool Methods Example: boolean isDaemon()

The following program illustrates how one can use the isDaemon() method.

FileName: IsDaemonExample.java

```
// Code illustrating the isDaemon() method

// import statement
import java.lang.*;


class ThreadNew extends Thread
```

```
{
// constructor of the class
ThreadNew(String tName, ThreadGroup tgrp)
{
super(tgrp, tName);
start();
}

// overriding the run() method
public void run()
{

for (int j = 0; j < 100; j++)
{
try
{
Thread.sleep(5);
}
catch (InterruptedException e)
{
System.out.println("The exception has been encountered" + e);
}

}

System.out.println(Thread.currentThread().getName() + " thread has finished executing");
}
}

public class IsDaemonExample
{
```

```
// main method
public static void main(String argvs[]) throws SecurityException, InterruptedException
{
// creating the thread group
ThreadGroup tg = new ThreadGroup("the parent group");

ThreadGroup tg1 = new ThreadGroup(tg, "the child group");

ThreadNew th1 = new ThreadNew("the first", tg);
System.out.println("Starting the first");

ThreadNew th2 = new ThreadNew("the second", tg);
System.out.println("Starting the second");

if (tg.isDaemon() == true)
{
System.out.println("The group is a daemon group.");
}
else
{
System.out.println("The group is not a daemon group.");
}

}
}
```

Output:

```
Starting the first
Starting the second
The group is not a daemon group.
the second thread has finished executing
the first thread has finished executing
```

Thread Pool Methods Example: boolean isDestroyed()

The following program illustrates how one can use the isDestroyed() method.

FileName: IsDestroyedExample.java

```
// Code illustrating the isDestroyed() method

// import statement
import java.lang.*;


class ThreadNew extends Thread
{
// constructor of the class
ThreadNew(String tName, ThreadGroup tgrp)
{
super(tgrp, tName);
start();
}

// overriding the run() method
public void run()
{

for (int j = 0; j < 100; j++)
{
try
{
Thread.sleep(5);
}
catch (InterruptedException e)
{
```

```
System.out.println("The exception has been encountered" + e);
}

}

System.out.println(Thread.currentThread().getName() + " thread has finished executing");
}
}

public class IsDestroyedExample
{
// main method
public static void main(String argvs[]) throws SecurityException, InterruptedException
{
// creating the thread group
ThreadGroup tg = new ThreadGroup("the parent group");

ThreadGroup tg1 = new ThreadGroup(tg, "the child group");

ThreadNew th1 = new ThreadNew("the first", tg);
System.out.println("Starting the first");

ThreadNew th2 = new ThreadNew("the second", tg);
System.out.println("Starting the second");

if (tg.isDestroyed() == true)
{
System.out.println("The group has been destroyed.");
}
else
{
```

```
System.out.println("The group has not been destroyed.");
}

}
}
```

Output:

```
Starting the first
Starting the second
The group has not been destroyed.
the first thread has finished executing
the second thread has finished executing
```

Java Shutdown Hook

Simple example of Shutdown Hook

FileName: MyThread.java

```
class MyThread extends Thread{
   public void run(){
      System.out.println("shut down hook task completed..");
   }
}

public class TestShutdown1{
public static void main(String[] args)throws Exception {

Runtime r=Runtime.getRuntime();
r.addShutdownHook(new MyThread());

System.out.println("Now main sleeping... press ctrl+c to exit");
try{Thread.sleep(3000);}catch (Exception e) {}
}
}
```

Output:

```
Now main sleeping... press ctrl+c to exit
shut down hook task completed.
```

Same example of Shutdown Hook by anonymous class:

FileName: TestShutdown2.java

```java
public class TestShutdown2{
public static void main(String[] args)throws Exception {

Runtime r=Runtime.getRuntime();

r.addShutdownHook(new Thread(){
public void run(){
    System.out.println("shut down hook task completed..");
    }
}
);

System.out.println("Now main sleeping... press ctrl+c to exit");
try{Thread.sleep(3000);}catch (Exception e) {}
}
}
```

Output:

```
Now main sleeping... press ctrl+c to exit
shut down hook task completed.
```

```java
FileName: RemoveHookExample.javapublic class RemoveHookExample
{

// the Msg class is derived from the Thread class
static class Msg extends Thread
{

```

```
public void run()
{
System.out.println("Bye ...");
}
}

// main method
public static void main(String[] argvs)
{
try
{
// creating an object of the class Msg
Msg ms = new Msg();

// registering the Msg object as the shutdown hook
Runtime.getRuntime().addShutdownHook(ms);

// printing the current state of program
System.out.println("The program is beginning ...");

// causing the thread to sleep for 2 seconds
System.out.println("Waiting for 2 seconds ...");
Thread.sleep(2000);

// removing the hook
Runtime.getRuntime().removeShutdownHook(ms);

// printing the message program is terminating
System.out.println("The program is terminating ...");
}
catch (Exception ex)
```

```
{
ex.printStackTrace();
}
}
}
```

Output:

```
The program is beginning ...
Waiting for 2 seconds ...
The program is terminating ..
```

How to perform single task by multiple threads in Java?

FileName: TestMultitasking1.java

```
class TestMultitasking1 extends Thread{
 public void run(){
   System.out.println("task one");
 }
 public static void main(String args[]){
  TestMultitasking1 t1=new TestMultitasking1();
  TestMultitasking1 t2=new TestMultitasking1();
  TestMultitasking1 t3=new TestMultitasking1();

  t1.start();
  t2.start();
  t3.start();
 }
}
```

Test it Now

Output:

```
task one
task one
task one
```

Program of performing single task by multiple threads

```
class TestMultitasking2 implements Runnable{
public void run(){
System.out.println("task one");
}

public static void main(String args[]){
Thread t1 =new Thread(new TestMultitasking2());//passing annonymous object of TestMultitasking2 class
Thread t2 =new Thread(new TestMultitasking2());

t1.start();
t2.start();

 }
}
```

Test it Now

Output:

```
task one
task one
```

Program of performing two tasks by two threads

FileName: TestMultitasking3.java

```
class Simple1 extends Thread{
 public void run(){
   System.out.println("task one");
 }
}

class Simple2 extends Thread{
 public void run(){
   System.out.println("task two");
 }
}

class TestMultitasking3{
 public static void main(String args[]){
  Simple1 t1=new Simple1();
  Simple2 t2=new Simple2();

  t1.start();
  t2.start();
 }
}
```

Test it Now

Output:

```
task one
task two
```

FileName: TestMultitasking4.java

```
class TestMultitasking4{
 public static void main(String args[]){
  Thread t1=new Thread(){
    public void run(){
      System.out.println("task one");
    }
  };
  Thread t2=new Thread(){
    public void run(){
      System.out.println("task two");
    }
  };


  t1.start();
t2.start();
 }
}
```

Test it Now

Output:

```
task one
task two
```

Same example as above by anonymous class that implements Runnable interface:

Program of performing two tasks by two threads

FileName: TestMultitasking5.java

```
class TestMultitasking5{
 public static void main(String args[]){
  Runnable r1=new Runnable(){
    public void run(){
      System.out.println("task one");
    }
  };

  Runnable r2=new Runnable(){
    public void run(){
      System.out.println("task two");
    }
  };

  Thread t1=new Thread(r1);
  Thread t2=new Thread(r2);

  t1.start();
  t2.start();
 }
}
```

Test it Now

Output:

```
task one
task two
```

FileName: OddEvenExample.java

```
// Java program that prints the odd and even numbers using two threads.
// the time complexity of the program is O(N), where N is the number up to which we
// are displaying the numbers
public class OddEvenExample
{
// Starting the counter
int contr = 1;
static int NUM;
// Method for printing the odd numbers
public void displayOddNumber()
{
// note that synchronized blocks are necessary for the code for getting the desired
// output. If we remove the synchronized blocks, we will get an exception.
synchronized (this)
{
// Printing the numbers till NUM
while (contr < NUM)
{
// If the contr is even then display
while (contr % 2 == 0)
{
// handling the exception handle
try
{
wait();
}
catch (InterruptedException ex)
{
ex.printStackTrace();
}
}
// Printing the number
System.out.print(contr + " ");
// Incrementing the contr
contr = contr + 1;
// notifying the thread which is waiting for this lock
notify();
}
}
}
// Method for printing the even numbers
public void displayEvenNumber()
{
synchronized (this)
{
// Printing the number till NUM
while (contr < NUM)
{
// If the count is odd then display
while (contr % 2 == 1)
{
// handling the exception
try
{
wait();
}
```

```
catch (InterruptedException ex)
{
ex.printStackTrace();
}
}
// Printing the number
System.out.print(contr + " ");
// Incrementing the contr
contr = contr +1;
// Notifying to the 2nd thread
notify();
}
}
}
// main method
public static void main(String[] argvs)
{
// The NUM is given
NUM = 20;
// creating an object of the class OddEvenExample
OddEvenExample oe = new OddEvenExample();
// creating a thread th1
Thread th1 = new Thread(new Runnable()
{
public void run()
{
// invoking the method displayEvenNumber() using the thread th1
oe.displayEvenNumber();
}
});
// creating a thread th2
```

```
Thread th2 = new Thread(new Runnable()
{
public void run()
{
// invoking the method displayOddNumber() using the thread th2
oe.displayOddNumber();
}
});
// starting both of the threads
th1.start();
th2.start();
}
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Java Garbage Collection

Simple Example of garbage collection in java

```
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 }
}
```

[Test it Now]

```
object is garbage collected
object is garbage collected
```

Java Runtime class

Java Runtime freeMemory() and totalMemory() method

```
public class MemoryTest{
 public static void main(String args[])throws Exception{
  Runtime r=Runtime.getRuntime();
  System.out.println("Total Memory: "+r.totalMemory());
  System.out.println("Free Memory: "+r.freeMemory());

  for(int i=0;i<10000;i++){
   new MemoryTest();
  }
  System.out.println("After creating 10000 instance, Free Memory: "+r.freeMemory());
  System.out.println("After gc(), Free Memory: "+r.freeMemory());
 }
}
```

```
Total Memory: 100139008
Free Memory: 99474824
After creating 10000 instance, Free Memory: 99310552
After gc(), Free Memory: 100182832
```