Java Program to Add two Numbers

Here, you will learn how to **write a Java program to add two numbers**. We will see three programs: In the first program, the values of the two numbers are given. In the second program, user is asked to enter the two numbers and the program calculates the sum of the input numbers. In the third program, we will calculate the sum of two non-integer numbers.

Example 1: Sum of two numbers

```
public class JavaExample {
  public static void main(String[] args) {
    // two integer variables with values
    // and a variable "sum" to store the result
    int num1 = 5, num2 = 15,sum;

    //calculating the sum of num1 and num2 and
    //storing the result in the variable sum
    sum = num1+num2;

    //printing the result
    System.out.println("Sum of "+num1+" and "+num2+" is: "+sum);
  }
}
```

Example 2: Sum of two numbers using Scanner

The Scanner class provides the methods that allows us to read the user input. The values entered by user is read using Scanner class and stored in two variables num1 and num2. The program then calculates the sum of input numbers and displays it.

```
import java.util.Scanner;
public class AddTwoNumbers2 {

    public static void main(String[] args) {

        int num1, num2, sum;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter First Number: ");
        num1 = sc.nextInt();

        System.out.println("Enter Second Number: ");
        num2 = sc.nextInt();

        sc.close();
        sum = num1 + num2;
        System.out.println("Sum of these numbers: "+sum);
    }
}
```

In this program, the statement Scanner sc = new Scanner(System.in); creates an instance of Scanner class. This instance calls nextInt() method to read the number entered by user. The read values are stored in variables num1 and num2. Once the values are stored in variables. The addition is performed on these variables and result is displayed.

Example 3: Program to add two non-integer numbers

In this example, we are calculating the sum of non-integer numbers. Here you can enter float numbers such as 10.5, 16.667 etc.

```
import java.util.Scanner;
public class JavaExample {

  public static void main(String[] args) {

    double num1, num2, sum;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter First Number: ");
    num1 = sc.nextDouble();

    System.out.print("Enter Second Number: ");
    num2 = sc.nextDouble();

    sc.close();
    sum = num1 + num2;
    System.out.println("Sum of "+num1+" and "+num2+" is: "+sum);
  }
}
```

Java Program to Check Whether a Number is Even or Odd

In this article, we will write two java programs to check whether a number is even or odd. If a number is perfectly divisible by 2 (number % 2 ==0) then the number is called even number, else the number is called odd number. Perfectly divisible by 2 means that when the number is divided by 2 the remainder is zero.

E.g.1: Program to check if the number entered by user is even or odd

In this program, we have an int variable num. We are using Scanner class to get the number entered by the user. Once the entered number is stored in the variable num, we are using if..else statement to check if the number is perfectly divisible by 2 or not. Based on the outcome, it is executing if block (if condition true) or else block (if condition is false).

```
import java.util.Scanner;
public class JavaExample
{
  public static void main(String args[])
  {
    int num;
    System.out.print("Enter an Integer number: ");

    //The input provided by user is stored in num
    Scanner input = new Scanner(System.in);
    num = input.nextInt();

    // If number is divisible by 2 then it's an even number
    //else it is an odd number
    if ( num % 2 == 0 )
      System.out.println(num+" is an even number.");
    else
      System.out.println(num+" is an odd number.");
  }
}
```

Java Program to check Leap Year

Here we will write a java program to check whether the input year is a leap year or not. Before we see the program, lets see how to determine whether a year is a leap year mathematically:

To determine whether a year is a leap year, follow these steps:
1. If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.
2. If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.
3. If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.
4. The year is a leap year (it has 366 days).
5. The year is not a leap year (it has 365 days). Source of these steps.

Example: Program to check whether the input year is leap or not

Here we are using **Scanner class** to get the input from user and then we are using if-else statements to write the logic to check leap year.

```
import java.util.Scanner;
public class Demo {

    public static void main(String[] args) {

        int year;
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter any Year:");
        year = scan.nextInt();
        scan.close();
        boolean isLeap = false;

        if(year % 4 == 0)
        {
            if( year % 100 == 0)
            {
                if ( year % 400 == 0)
                    isLeap = true;
                else
                    isLeap = false;
            }
            else
                isLeap = true;
        }
        else {
            isLeap = false;
        }

        if(isLeap==true)
            System.out.println(year + " is a Leap Year.");
        else
            System.out.println(year + " is not a Leap Year.");
    }
}
```

Java Program to check Vowel or Consonant using Switch Case

The alphabets A, E, I, O and U (smallcase and uppercase) are known as Vowels and rest of the alphabets are known as consonants. Here we will write a java program that checks whether the input character is vowel or Consonant using Switch Case in Java.

Example: Program to check Vowel or Consonant using Switch Case

In this program we are not using break statement with cases intentionally, so that if user enters any vowel, the program continues to execute all the subsequent cases until `Case 'U'` is reached and that's where we are setting up the value of a boolean variable to true. This way we can identify that the alphabet entered by user is vowel or not.

```
import java.util.Scanner;
class JavaExample
{
   public static void main(String[ ] arg)
   {
        boolean isVowel=false;;
        Scanner scanner=new Scanner(System.in);
        System.out.println("Enter a character : ");
        char ch=scanner.next().charAt(0);
        scanner.close();
        switch(ch)
        {
           case 'a' :
           case 'e' :
           case 'i' :
           case 'o' :
           case 'u' :
           case 'A' :
           case 'E' :
           case 'I' :
           case 'O' :
           case 'U' : isVowel = true;
        }
        if(isVowel == true) {
           System.out.println(ch+" is  a Vowel");
        }
        else {
           if((ch>='a'&&ch<='z')||(ch>='A'&&ch<='Z'))
                System.out.println(ch+" is a Consonant");
           else
                System.out.println("Input is not an alphabet");
        }
   }
}
```

Java Program to Calculate Compound Interest

Compound Interest Formula

Compound interest is calculated using the following formula:

P (1 + R/n) $^{(nt)}$ - P

Here **P** is principal amount.
R is the annual interest rate.
t is the time the money is invested or borrowed for.
n is the number of times that interest is compounded per unit t, for example if interest is compounded monthly and t is in years then the value of n would be 12. If interest is compounded quarterly and t is in years then the value of n would be 4.
Before writing the java program let's take an example to calculate the compound interest.
Let's say an amount of $2,000 is deposited into a bank account as a fixed deposit at an annual interest rate of 8%, compounded monthly, the compound interest after 5 years would be:
P = 2000.
R = 8/100 = 0.08 (decimal).

n = 12.
t = 5.
Let’s put these values in the formula.
Compound Interest = 2000 (1 + 0.08 / 12) (12 * 5) – 2000 = $979.69
So, the compound interest after 5 years is $979.69.

Java Program to calculate Compound Interest

In this java program we are calculating the compound interest, we are taking the **same example** that we have seen above for the calculation.

```
public class JavaExample {

    public void calculate(int p, int t, double r, int n) {
        double amount = p * Math.pow(1 + (r / n), n * t);
        double cinterest = amount - p;
        System.out.println("Compound Interest after " + t + " years: "+cinterest);
        System.out.println("Amount after " + t + " years: "+amount);
    }
    public static void main(String args[]) {
        JavaExample obj = new JavaExample();
        obj.calculate(2000, 5, .08, 12);
    }
}
```

Java Program to calculate and display Student Grades

This program calculates the grade of a student based on the marks entered by user in each subject. Program prints the grade based on this logic.
If the average of marks is >= 80 then prints Grade ‘A’
If the average is <80 and >=60 then prints Grade ‘B’
If the average is <60 and >=40 then prints Grade ‘C’
else prints Grade ‘D’

Example: Program to display the grade of student

```
import java.util.Scanner;

public class JavaExample
{
    public static void main(String args[])
    {
        /* This program assumes that the student has 6 subjects,
         * thats why I have created the array of size 6. You can
         * change this as per the requirement.
         */
        int marks[] = new int[6];
        int i;
        float total=0, avg;
        Scanner scanner = new Scanner(System.in);

        for(i=0; i<6; i++) {
           System.out.print("Enter Marks of Subject"+(i+1)+":");
           marks[i] = scanner.nextInt();
           total = total + marks[i];
        }
        scanner.close();
        //Calculating average here
        avg = total/6;
        System.out.print("The student Grade is: ");
        if(avg>=80)
        {
            System.out.print("A");
        }
        else if(avg>=60 && avg<80)
        {
           System.out.print("B");
        }
        else if(avg>=40 && avg<60)
        {
            System.out.print("C");
        }
        else
        {
            System.out.print("D");
        }
    }
}
```

Java Program to Find Quotient and Remainder

In the following program we have two integer numbers num1 and num2 and we are finding the quotient and remainder when num1 is divided by num2, so we can say that num1 is the **dividend** here and num2 is the **divisor**.

```
public class JavaExample {
    public static void main(String[] args) {
        int num1 = 15, num2 = 2;
        int quotient = num1 / num2;
        int remainder = num1 % num2;
        System.out.println("Quotient is: " + quotient);
        System.out.println("Remainder is: " + remainder);
    }
}
```

Java Program to Multiply Two Numbers

When you start learning java programming, you get these type of problems in your assignment. Here we will see two Java programs, first program takes two integer numbers (entered by user) and displays the product of these numbers. The second program takes any two numbers (can be integer or floating point) and displays the result.

Example 1: Program to read two integer and print product of them

This program asks user to enter two integer numbers and displays the product. To understand how to use scanner to take user input, checkout this program:

```
import java.util.Scanner;

public class Demo {

    public static void main(String[] args) {

        /* This reads the input provided by user
         * using keyboard
         */
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter first number: ");
        // This method reads the number provided using keyboard
        int num1 = scan.nextInt();

        System.out.print("Enter second number: ");
        int num2 = scan.nextInt();

        // Closing Scanner after the use
        scan.close();

        // Calculating product of two numbers
        int product = num1*num2;

        // Displaying the multiplication result
        System.out.println("Output: "+product);
    }
}
```

Example 2: Read two integer or floating point numbers and display the multiplication

In the above program, we can only integers. What if we want to calculate the multiplication of two float numbers? This programs allows you to enter float numbers and calculates the product. Here we are using data type double for numbers so that you can enter integer as well as floating point numbers.

```
import java.util.Scanner;

public class Demo {

    public static void main(String[] args) {

        /* This reads the input provided by user
         * using keyboard
         */
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter first number: ");

        // This method reads the number provided using keyboard
        double num1 = scan.nextDouble();

        System.out.print("Enter second number: ");
        double num2 = scan.nextDouble();

        // Closing Scanner after the use
        scan.close();

        // Calculating product of two numbers
        double product = num1*num2;

        // Displaying the multiplication result
        System.out.println("Output: "+product);
    }
}
```

Java Program to calculate simple interest

In the following example we are taking the values of p, r and t from user and then we are calculating the simple interest based on entered values.

```
import java.util.Scanner;
```

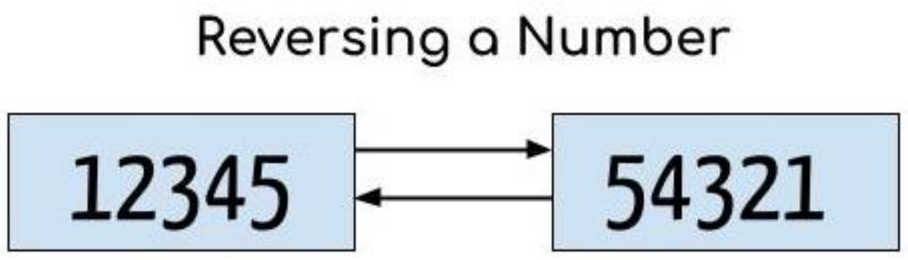
```
public class JavaExample
{
    public static void main(String args[])
    {
        float p, r, t, simple_nterest;
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter the Principal : ");
        p = scan.nextFloat();
        System.out.print("Enter the Rate of simple_nterest: ");
        r = scan.nextFloat();
        System.out.print("Enter the Time period : ");
        t = scan.nextFloat();
        scan.close();
        sinterest = (p * r * t) / 100;
        System.out.print("Simple Interest is: " +sinterest);
    }
}
```

Example: Program to find Quotient and Remainder

In the following program we have two integer numbers num1 and num2 and we are finding the quotient and remainder when num1 is divided by num2, so we can say that num1 is the **dividend** here and num2 is the **divisor**.

```
public class JavaExample {
    public static void main(String[] args) {
        int num1 = 15, num2 = 2;
        int quotient = num1 / num2;
        int remainder = num1 % num2;
        System.out.println("Quotient is: " + quotient);
        System.out.println("Remainder is: " + remainder);
    }
}
```

Java program to reverse a number using for, while and recursion



```
import java.util.Scanner;
class ReverseNumberWhile
{
   public static void main(String args[])
   {
      int num=0;
      int reversenum =0;
      System.out.println("Input your number and press enter: ");
      //This statement will capture the user input
      Scanner in = new Scanner(System.in);
      //Captured input would be stored in number num
      num = in.nextInt();
      //While Loop: Logic to find out the reverse number
      while( num != 0 )
      {
          reversenum = reversenum * 10;
          reversenum = reversenum + num%10;
          num = num/10;
      }

      System.out.println("Reverse of input number is: "+reversenum);
   }
}
```

Program 1: Reverse a number using while Loop

- In this program, user is asked to enter a number.
- This input number is read and stored in a variable **num** using Scanner class.
- The program then uses the while loop to reverse this number.
- Inside the while loop , the given number is divided by 10 using % (modulus) operator and then storing the remainder in the reversenum variable after multiplying the reversenum by 10. **When we divide the number by 10, it returns the last digit as remainder.** This remainder becomes the first digit of reversenum, we are repeating this step again and again until the given number become zero and all the digits are appended in the reversenum.
- At the end of the loop the variable reversenum contains the reverse number and the program prints the value of this variable as output.

Program 2: Reverse a number using for Loop

The logic in this program is same as above program, here we are using for loop instead of while loop. As you can see, in this program we have not used the initialization and increment/decrement section of for loop because we have already initialized the variables outside the loop and we are decreasing the value of num inside for loop by diving it by 10.

```
import java.util.Scanner;
class ForLoopReverseDemo
{
   public static void main(String args[])
   {
      int num=0;
      int reversenum =0;
      System.out.println("Input your number and press enter: ");
      //This statement will capture the user input
      Scanner in = new Scanner(System.in);
      //Captured input would be stored in number num
      num = in.nextInt();
      /* for loop: No initialization part as num is already
       * initialized and no increment/decrement part as logic
       * num = num/10 already decrements the value of num
       */
      for( ;num != 0; )
      {
          reversenum = reversenum * 10;
          reversenum = reversenum + num%10;
          num = num/10;
      }

      System.out.println("Reverse of specified number is: "+reversenum);
   }
}
```

Program 3: Reverse a number using recursion

●Here we are using **recursion** to reverse the number. A method is called recursive method, if it calls itself and this process is called recursion. We have defined a recursive method reverseMethod() and we are passing the input number to this method.

●This method then divides the number by 10, displays the **remainder** and then calls itself by passing the **quotient** as parameter. This process goes on and on until the number is in single digit and then it displays the last digit (which is the first digit of the number) and ends the recursion.

●**Note:** You do not get confused between quotient and remainder so let me explain a bit about them here, / operator return quotient and % operator returns remainder. For example: 21/5 would return 4 (quotient), while 21%5 would return 1(quotient).

```
import java.util.Scanner;
class RecursionReverseDemo
{
   //A method for reverse
   public static void reverseMethod(int number) {
       if (number < 10) {
           System.out.println(number);
           return;
       }
       else {
           System.out.print(number % 10);
           //Method is calling itself: recursion
           reverseMethod(number/10);
       }
   }
   public static void main(String args[])
   {
        int num=0;
        System.out.println("Input your number and press enter: ");
        Scanner in = new Scanner(System.in);
        num = in.nextInt();
        System.out.print("Reverse of the input number is:");
        reverseMethod(num);
        System.out.println();
   }
}
```

Java Program to Check two Strings are anagram or not

Anagram of a string is another string with the same characters but order of the characters can be different. For example, Two strings “Listen” and “Silent” are anagram strings as both contain same characters, just the order of the characters is different. Similarly Strings “Race” and “Care” are also anagrams. You will learn **how to write a java program to check two strings are anagram or not**.

Program to determine whether two strings are anagram

Steps:

●Convert the given strings in the lowercase letters using toLowerCase() method, this is to perform case insensitive comparison.
●Compare the lengths (find length using length() method) of the given strings, if lengths are not equal then strings cannot be anagram else proceed to next step.
●Copy the characters of the strings into an array using toCharArray() method.
●Sort the arrays using Arrays.sort(). This is to sort the characters in the array in a particular order as the anagram strings characters are in different order. For example, the arrays of strings “race” and “care” are {‘r’, ‘a’, ‘c’, ‘e’} & {‘c’, ‘a’, ‘r’, ‘e’} receptively. After sorting both the arrays are same : {‘a’, ‘c’, ‘e’, ‘r’}
●Compare the sorted arrays, if the arrays are equal then the strings are anagrams else they are not anagram strings.

```
import java.util.Arrays;
public class JavaExample {
  public static void main (String [] args) {
    String str1="Race";
    String str2="Care";

    //Converting the strings to lower case to
    //perform case insensitive comparison
    str1 = str1.toLowerCase();
    str2 = str2.toLowerCase();

    //Comparing the length of the two strings if the lengths
    //are not equal then strings cannot be anagrams
    if (str1.length() != str2.length()) {
      System.out.println("Given strings are not anagram.");
    }
    else {
      //copying the characters of each strings in two different arrays
      char[] arrayOfStr1 = str1.toCharArray();
      char[] arrayOfStr2 = str2.toCharArray();

      //Sorting both the arrays using Arrays.sort()
      Arrays.sort(arrayOfStr1);
      Arrays.sort(arrayOfStr2);

      //Comparing the sorted arrays, if equal anagram else not anagram
      if(Arrays.equals(arrayOfStr1, arrayOfStr2) == true) {
        System.out.println("Given Strings are anagram.");
      }
      else {
        System.out.println("Given Strings are not anagram.");
      }
    }
  }
}
```

How To Convert Char To String and a String to char in Java.

We have following two ways for char to String conversion.
Method 1: Using toString() method
Method 2: Usng valueOf() method

```
class CharToStringDemo
```

```
{
   public static void main(String args[])
   {
      // Method 1: Using toString() method
      char ch = 'a';
      String str = Character.toString(ch);
      System.out.println("String is: "+str);

      // Method 2: Using valueOf() method
      String str2 = String.valueOf(ch);
      System.out.println("String is: "+str2);
   }
}
```

Converting String to Char

We can convert a String to char using charAt() method of String class.

```
class StringToCharDemo
{
   public static void main(String args[])
   {
      // Using charAt() method
      String str = "Hello";
      for(int i=0; i<str.length();i++){
        char ch = str.charAt(i);
        System.out.println("Character at "+i+" Position: "+ch);
      }
   }
}
```

Java Program to find duplicate Characters in a String

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class Details {

  public void countDupChars(String str){

    //Create a HashMap
    Map<Character, Integer> map = new HashMap<Character, Integer>();

    //Convert the String to char array
    char[] chars = str.toCharArray();

    /* logic: char are inserted as keys and their count
     * as values. If map contains the char already then increase the value by 1
     */
    for(Character ch:chars){
      if(map.containsKey(ch)){
         map.put(ch, map.get(ch)+1);
      } else {
         map.put(ch, 1);
        }
    }

    //Obtaining set of keys
    Set<Character> keys = map.keySet();
    /* Display count of chars if it is greater than 1. All duplicate chars would be
     * having value greater than 1.
     */
    for(Character ch:keys){
      if(map.get(ch) > 1){
        System.out.println("Char "+ch+" "+map.get(ch));
      }
    }
  }

  public static void main(String a[]){
    Details obj = new Details();
    System.out.println("String: BeginnersBook.com");
    System.out.println("-------------------------");
    obj.countDupChars("BeginnersBook.com");

    System.out.println("\nString: ChaitanyaSingh");
    System.out.println("-------------------------");
    obj.countDupChars("ChaitanyaSingh");

    System.out.println("\nString: #@$@!#$%!!%@");
    System.out.println("-------------------------");
    obj.countDupChars("#@$@!#$%!!%@");
  }
}
```

Java Program to Sort Strings in an Alphabetical Order

In this program, we are asking user to enter the count of strings that he would like to enter for sorting. Once the count is captured using Scanner class, we have initialized a String array of the input count size and then are running a for loop to capture all the strings input by user. Once we have all the strings stored in the string array, we are comparing the first alphabet of each string to get them sorted in the alphabetical order.

```
import java.util.Scanner;
public class JavaExample
{
    public static void main(String[] args)
    {
        int count;
        String temp;
        Scanner scan = new Scanner(System.in);

        //User will be asked to enter the count of strings
        System.out.print("Enter number of strings you would like to enter:");
        count = scan.nextInt();

        String str[] = new String[count];
        Scanner scan2 = new Scanner(System.in);

        //User is entering the strings and they are stored in an array
        System.out.println("Enter the Strings one by one:");
        for(int i = 0; i < count; i++)
        {
            str[i] = scan2.nextLine();
        }
        scan.close();
        scan2.close();

        //Sorting the strings
        for (int i = 0; i < count; i++)
        {
            for (int j = i + 1; j < count; j++) {
                if (str[i].compareTo(str[j])>0)
                {
                    temp = str[i];
                    str[i] = str[j];
                    str[j] = temp;
                }
            }
        }

        //Displaying the strings after sorting them based on alphabetical order
        System.out.print("Strings in Sorted Order:");
        for (int i = 0; i <= count - 1; i++)
        {
            System.out.print(str[i] + ", ");
        }
    }
}
```

Java Program to reverse words in a String

This program reverses every word of a string and display the reversed string as an output. For example, if we input a string as "Reverse the word of this string" then the output of the program would be: "esrever eht drow fo siht gnirts".

To understand this program you should have the knowledge of following Java Programming topics:

1. For loop in Java
2. Java String split() method
3. Java String charAt() method

Example: Program to reverse every word in a String using methods

In this Program, we first split the given string into substrings using split() method. The substrings are stored in an String array words. The program then reverse each word of the substring using a reverse for loop.

```
public class Example
{
   public void reverseWordInMyString(String str)
   {
        /* The split() method of String class splits
         * a string in several strings based on the
         * delimiter passed as an argument to it
         */
        String[] words = str.split(" ");
        String reversedString = "";
        for (int i = 0; i < words.length; i++)
        {
           String word = words[i];
           String reverseWord = "";
           for (int j = word.length()-1; j >= 0; j--)
           {
                /* The charAt() function returns the character
                 * at the given position in a string
                 */
                reverseWord = reverseWord + word.charAt(j);
```

```
          }
          reversedString = reversedString + reverseWord + " ";
       }
       System.out.println(str);
       System.out.println(reversedString);
   }
   public static void main(String[] args)
   {
       Example obj = new Example();
       obj.reverseWordInMyString("Welcome to BeginnersBook");
       obj.reverseWordInMyString("This is an easy Java Program");
   }
}
```

Java program to find the occurrence of a character in a string

Here, we find the occurrence of each character in a String. To do this we are first creating an array of size 256 (ASCII upper range), the idea here is to store the occurrence count against the ASCII value of that character. For example, the occurrence of 'A' would be stored in counter[65] because ASCII value of A is 65, similarly occurrences of other chars are stored in against their ASCII index values.We then create an another array array to hold the characters of the given String, then we are comparing them with the characters in the String and when a match is found the count of that particular char is displayed using counter array.

```
class JavaExample {

   static void countEachChar(String str)
   {
       //ASCII values ranges upto 256
       int counter[] = new int[256];

       //String length
       int len = str.length();

       /* This array holds the occurrence of each char, For example
        * ASCII value of A is 65 so if A is found twice then
        * counter[65] would have the value 2, here 65 is the ASCII value
        * of A
        */
       for (int i = 0; i < len; i++)
               counter[str.charAt(i)]++;

       // We are creating another array with the size of String
       char array[] = new char[str.length()];
       for (int i = 0; i < len; i++) {
          array[i] = str.charAt(i);
          int flag = 0;
          for (int j = 0; j <= i; j++) {

               /* If a char is found in String then set the flag
                * so that we can print the occurrence
                */
               if (str.charAt(i) == array[j])
                       flag++;
          }
```

```
            if (flag == 1)
               System.out.println("Occurrence of char " + str.charAt(i)
                  + " in the String is:" + counter[str.charAt(i)]);
         }
    }
    public static void main(String[] args)
    {
         String str = "beginnersbook";
         countEachChar(str);
    }
}
```

Java Program to Count Vowels and Consonants in a String

Here we have two variables vcount and ccount to keep the count of vowels and consonants respectively. We have converted each char of the string to lowercase using toLowerCase() method for easy comparison. We are then comparing each char of the string to vowels 'a', 'e', 'i', 'o', 'u' using charAt() method and if..else..if statement, if a match is found then we are increasing the vowel counter vcount else we are increasing the Consonant counter ccount.

```
public class JavaExample {

    public static void main(String[] args) {
        String str = "BeginnersBook";
        int vcount = 0, ccount = 0;

        //converting all the chars to lowercase
        str = str.toLowerCase();
        for(int i = 0; i < str.length(); i++) { char ch = str.charAt(i); if(ch == 'a' || ch
== 'e' || ch == 'i' || ch == 'o' || ch == 'u') { vcount++; } else if((ch >= 'a'&& ch <=
'z')) {
                ccount++;
            }
        }
        System.out.println("Number of Vowels: " + vcount);
        System.out.println("Number of Consonants: " + ccount);
    }
}
```

Java Program to remove all the white spaces from a string

We have a string str and this string contain whitespaces. We are using replaceAll() method to replace the whitespaces in the existing string with the blanks. This will remove the whitespaces from the given string.

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

    String str="Hi, Welcome to BeginnersBook.com";
    System.out.println("Original String:" + str);

    //Using regex inside replaceAll() method to replace space with blank
    str = str.replaceAll("\\s+", "");

    System.out.println("String after white spaces are removed:" + str);
  }
}
```

```
import java.util.Scanner;
public class JavaExample {
  public static void main(String[] args) {

     String str;
     System.out.println("Enter any String: ");
     Scanner sc = new Scanner(System.in);
     str = sc.nextLine();
     System.out.println("Original String: " + str);

     //replacing white spaces space with blank
     str = str.replaceAll("\\s+", "");

     System.out.println("String after white spaces are removed: " + str);
  }
}
```

Java Program to Calculate average using Array

We will see two programs to find the average of numbers using array. First Program finds the average of specified array elements. The second programs takes the value of n (number of elements) and the numbers provided by user and finds the average of them using array.
To understand these programs you should have the knowledge of following Java Programming concepts: 1) Java Arrays 2) For loop

Example 1: Program to find the average of numbers using array

```
public class JavaExample {

    public static void main(String[] args) {
        double[] arr = {19, 12.89, 16.5, 200, 13.7};
        double total = 0;

        for(int i=0; i<arr.length; i++){
                total = total + arr[i];
        }

        /* arr.length returns the number of elements
         * present in the array
         */
        double average = total / arr.length;

        /* This is used for displaying the formatted output
         * if you give %.4f then the output would have 4 digits
         * after decimal point.
         */
        System.out.format("The average is: %.3f", average);
    }
}
```

Example 2: Calculate average of numbers entered by user

Here, we are using Scanner to get the value of n and all the numbers from user.

```
import java.util.Scanner;
```

```java
public class JavaExample {

    public static void main(String[] args) {
        System.out.println("How many numbers you want to enter?");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        /* Declaring array of n elements, the value
         * of n is provided by the user
         */
        double[] arr = new double[n];
        double total = 0;

        for(int i=0; i<arr.length; i++){
                System.out.print("Enter Element No."+(i+1)+": ");
                arr[i] = scanner.nextDouble();
        }
        scanner.close();
        for(int i=0; i<arr.length; i++){
                total = total + arr[i];
        }

        double average = total / arr.length;

        System.out.format("The average is: %.3f", average);
    }
}
```

Java program to sum the elements of an array

Program 1: No user interaction

```java
/**
 * @author: BeginnersBook.com
 * @description: Get sum of array elements
 */
class SumOfArray{
   public static void main(String args[]){
      int[] array = {10, 20, 30, 40, 50, 10};
      int sum = 0;
      //Advanced for loop
      for( int num : array) {
          sum = sum+num;
      }
      System.out.println("Sum of array elements is:"+sum);
   }
}
```

Program 2: User enters the array's elements

```java
/**
 * @author: BeginnersBook.com
 * @description: User would enter the 10 elements
 * and the program will store them into an array and
 * will display the sum of them.
 */
import java.util.Scanner;
class SumDemo{
   public static void main(String args[]){
      Scanner scanner = new Scanner(System.in);
      int[] array = new int[10];
```

```
        int sum = 0;
        System.out.println("Enter the elements:");
        for (int i=0; i<10; i++)
        {
            array[i] = scanner.nextInt();
        }
        for( int num : array) {
            sum = sum+num;
        }
        System.out.println("Sum of array elements is:"+sum);
    }
}
```

Java Program to reverse the Array

This program reverse the array. For example if user enters the array elements as 1, 2, 3, 4, 5 then the program would reverse the array and the elements of array would be 5, 4, 3, 2, 1. To understand this program, you should have the knowledge of following Java Programming topics:

1. Arrays in Java
2. Java For loop
3. Java While loop

Example: Program to reverse the array

```
import java.util.Scanner;
public class Example
{
   public static void main(String args[])
   {
        int counter, i=0, j=0, temp;
        int number[] = new int[100];
        Scanner scanner = new Scanner(System.in);
        System.out.print("How many elements you want to enter: ");
        counter = scanner.nextInt();

        /* This loop stores all the elements that we enter in an
         * the array number. First element is at number[0], second at
         * number[1] and so on
         */
        for(i=0; i<counter; i++)
        {
            System.out.print("Enter Array Element"+(i+1)+": ");
            number[i] = scanner.nextInt();
        }

        /* Here we are writing the logic to swap first element with
         * last element, second last element with second element and
         * so on. On the first iteration of while loop i is the index
         * of first element and j is the index of last. On the second
         * iteration i is the index of second and j is the index of
         * second last.
         */
        j = i - 1;
        i = 0;
        scanner.close();
        while(i<j)
        {
           temp = number[i];
           number[i] = number[j];
```

```
            number[j] = temp;
            i++;
            j--;
        }

        System.out.print("Reversed array: ");
        for(i=0; i<counter; i++)
        {
            System.out.print(number[i]+ "  ");
        }
    }
}
```

Java Program to Sort an Array in Ascending Order

```
import java.util.Scanner;
public class JavaExample
{
    public static void main(String[] args)
    {
        int count, temp;

        //User inputs the array size
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter number of elements you want in the array: ");
        count = scan.nextInt();

        int num[] = new int[count];
        System.out.println("Enter array elements:");
        for (int i = 0; i < count; i++)
        {
            num[i] = scan.nextInt();
        }
        scan.close();
        for (int i = 0; i < count; i++)
        {
            for (int j = i + 1; j < count; j++) {
                if (num[i] > num[j])
                {
                    temp = num[i];
                    num[i] = num[j];
                    num[j] = temp;
                }
            }
        }
        System.out.print("Array Elements in Ascending Order: ");
        for (int i = 0; i < count - 1; i++)
        {
            System.out.print(num[i] + ", ");
        }
        System.out.print(num[count - 1]);
    }
}
```

How to convert a char array to a string in Java?

There are two ways to convert a char array (char[]) to String in Java:
1) Creating String object by passing array name to the constructor
2) Using valueOf() method of String class.

Example:

This example demonstrates both the above mentioned ways of converting a char array to String. Here we have a char array ch and we have created two strings str and str1 using the char array.

```
class CharArrayToString
{
   public static void main(String args[])
   {
      // Method 1: Using String object
      char[] ch = {'g', 'o', 'o', 'd', ' ', 'm', 'o', 'r', 'n', 'i', 'n', 'g'};
      String str = new String(ch);
      System.out.println(str);

      // Method 2: Using valueOf method
      String str2 = String.valueOf(ch);
      System.out.println(str2);
   }
}
```

Java Program to Add Two Matrix using Multi-dimensional Arrays

In the following example we have two matrices MatrixA and MatrixB, we have declared these matrices as multi-dimensional arrays.

Two matrices can only be added or subtracted only if they have same dimension which means they must have the **same number of rows and columns**. Here we have two MatrixA and MatrixB which have same rows and columns. The addition of these matrices will have same rows and columns.
This is how we declare a matrix as multi-dimensional array:
Matrix: This matrix has two rows and four columns.

```
| 1 1 1 1 |
| 2 3 5 2 |
```

The declaration of this matrix as 2D array:

```
int[][] MatrixA = { {1, 1, 1, 1}, {2, 3, 5, 2} };
```

We are using for loop to add the corresponding elements of both the matrices and store the addition values in sum matrix. For example: sum[0][0] = MatrixA[0][0] + MatrixB[0][0], similarly sum[0][1] = MatrixA[0][1] + MatrixB[0][1] and so on.

```
public class JavaExample {
    public static void main(String[] args) {
        int rows = 2, columns = 4;

        // Declaring the two matrices as multi-dimensional arrays
        int[][] MatrixA = { {1, 1, 1, 1}, {2, 3, 5, 2} };
        int[][] MatrixB = { {2, 3, 4, 5}, {2, 2, 4, -4} };

        /* Declaring a matrix sum, that will be the sum of MatrixA
         * and MatrixB, the sum matrix will have the same rows and
         * columns as the given matrices.
         */
        int[][] sum = new int[rows][columns];
        for(int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                sum[i][j] = MatrixA[i][j] + MatrixB[i][j];
            }
        }
        // Displaying the sum matrix
        System.out.println("Sum of the given matrices is: ");
        for(int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                System.out.print(sum[i][j] + "    ");
            }
            System.out.println();
        }
    }
}
```

Java Program to print the elements of an array present on odd position

Here, we will write a java program to **print the elements of an array present on odd position**. For example, if an array is {2, 12, 23, 7, 6, 15} then we need to display the elements 2, 23 and 6 as they are present in the array on **odd positions**.

Steps to find even position elements of an Array

1. Initialize the array.
2. Run a loop starting from 0 till the length of the given array. Use +2 in the increment part of the loop, to only traverse the odd positions in the array.
    - Print the current element of the array
3. End of the program.

Program to display array elements on even positions

Please go through the comments mentioned in the following program to understand why we are starting the loop from 0 and why are using `i=i+2` in the increment part of the loop.

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

    //Initializing the array
    int [] numbers = new int [] {1, 3, 5, 7, 9, 11, 13};

    System.out.println("Array Elements on odd Positions: ");
    /* Note we are using i = i+2 as we are only traversing odd positions
     * Important point here is that the array indices start with 0, which
     * means the odd positions such as 1st, 3rd and 5th positions are having
     * indices 0, 2, 4 and so on. That's why numbers[0] prints 1st position
     * element of the array.
     */
    for (int i = 0; i < numbers.length; i = i+2) {
      System.out.println(numbers[i]);
    }
  }
}
```

Java Program to print the elements of an array present on even position

Here, we will write a java program to print the elements of an array present on even position. For example, if an array is {1, 4, 8, 3, 9, 17} then we need to display the elements 4, 3 and 17 as they are present in the array on even positions.

Steps to find even position elements of an Array

1. Initialize the array.
2. Run a loop starting from 1 till the length of the given array. Use +2 in the increment part of the loop, to only traverse the even positions in the array.
3. Display current array element
4. End of the program.

Program to display array elements on even positions

Please go through the comments mentioned in the following program to understand why we are starting the loop from 1 and why are using i=i+2 in the increment part of the loop.

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

    //Initializing the array
    int [] numbers = new int [] {5, 12, 16, 3, 9, 7, 1, 100};

    System.out.println("Array Elements on even Positions: ");
    /* Note we are using i = i+2 as we are only traversing even positions
     * Important point here is that the array indices start with 0, which
     * means the even positions such as 2nd, 4th and 6th positions are having
     * indices 1, 3, 5 and so on. That's why numbers[1] prints 2nd position
     * element of the array.
     */
    for (int i = 1; i < numbers.length; i = i+2) {
      System.out.println(numbers[i]);
    }
  }
}
```

Java Program to copy all elements of one array into another array

Here, we will write a java program to copy all the elements of an array to another array. This can be easily done by using any loop such as for, while or do-while loop. We just need to run a loop from 0 to the array length (size of an array) and at every iteration, read the element of the given array and write the same value in another array.

```
For example:
array1 is {3, 5, 7, 9}
array2[] is {}
Then after copying all the elements of array1[] to array2[]
the elements in array2[] will be = {3, 5, 7, 9}
```

Steps to copy all elements from one array to another array

1. Initialize the first array.
2. Create another array with the same size as of the first array
3. Run a loop from 0 till the length of first array

- Read the element of first array
- Copy the element to the second array

4. Repeat the step 3 until the complete array is traversed
5. Run a loop from 0 to the length of any array
    - Read and print the elements of second array
6. Repeat the step 5 until the second array is completely traversed.
7. End of Program.

```
public class JavaExample {
  public static void main(String[] args) {
    //Initializing an array
    int [] firstArray = new int [] {3, 5, 7, 9, 11};
    /* Creating another array secondArray with same size
     * of first array using firstArray.length as it returns
     * the size of array firstArray.
     */
    int secondArray[] = new int[firstArray.length];

    //Displaying elements of first array
    System.out.println("Elements of First array: ");
    for (int i = 0; i < firstArray.length; i++) {
      System.out.print(firstArray[i] + " ");
    }

    //Copying all elements of firstArray to secondArray
    for (int i = 0; i < firstArray.length; i++) {
      secondArray[i] = firstArray[i];
    }

    //Displaying elements of secondArray
    System.out.println();
    System.out.println("Elements of Copied array: ");
    for (int i = 0; i < secondArray.length; i++) {
      System.out.print(secondArray[i] + " ");
    }
  }
}
```

Java Program to count the frequency of each element in array

```
public class JavaExample {
  public static void main(String[] args) {
    //Initializing an array
    int [] numbers = new int [] {2, 2, 3, 4, 5, 5, 5, 3, 2, 4};
    //This array will store the frequency of each element
    int [] frequency = new int [numbers.length];
    int counted = -1;
    for(int i = 0; i < numbers.length; i++){
      int count = 1;
      for(int j = i+1; j < numbers.length; j++){
        if(numbers[i] == numbers[j]){
          count++;
          //To avoid counting the frequency of same element again
          frequency[j] = counted;
        }
      }
      if(frequency[i] != counted)
        frequency[i] = count;
```

```java
    }

    //Printing the frequency of each element
    for(int i = 0; i < frequency.length; i++){
      if(frequency[i] != counted)
        System.out.println("Element: "+numbers[i] + " Frequency: " + frequency[i]);
    }
  }}
```

Java Program to print the duplicate elements of an array

```java
public class JavaExample {
  public static void main(String[] args) {
    //Initializing an int array
    int [] numbers = new int [] {2, 4, 6, 8, 4, 6, 10, 10};
    System.out.println("Duplicate elements in given array are: ");
    //Comparing each element of the array with all other elements
    for(int i = 0; i < numbers.length; i++) {
      for(int j = i + 1; j < numbers.length; j++) {
        if(numbers[i] == numbers[j]) {
          //printing duplicate elements
          System.out.println(numbers[j]);
        }
      }
    }
  }
}
```

Java Program to find largest element of an array

Here, we will write a **java program to find the largest element of an array**. In theory, a largest element can be found by, storing the first element of the array in a variable largestElement (can be given any name). Then comparing that variable with rest of the elements, if any element is found greater than the largestElement then store the value of that element into the variable. Repeat the process until all the elements are compared.

Program to print the largest element of a given array

In the following example, we have an int array arr. The first element of this given array is stored into an int variable largestElement. This is just the initialization of this variable, at the end of the program, this variable will hold the largest element of the array.
Running a loop from 0 till the size of the array. At every loop iteration, each element of the array is compared with the largestElement, if any of the element is found greater than largestElement, then the value of that element is assigned to largestElement. This process continues until the whole array is traversed. In the end, the variable largestElement holds the largest element of the given array.

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

    //Initializing an int array
    int [] arr = new int [] {11, 22, 33, 99, 88, 77};
    //This element will store the largest element of the array
    //Initializing with the first element of the array
    int largestElement = arr[0];
    //Running the loop from 1st element till last element
    for (int i = 0; i < arr.length; i++) {
      //Compare each elements of array with largestElement
      //If an element is greater, store the element into largestElement
      if(arr[i] > largestElement)
        largestElement = arr[i];
    }
    System.out.println("Largest element of given array: " + largestElement);
  }
}
```

Java program to find smallest number in an array

Here, you will learn how to write a **java program to find the smallest element in an array**. In theory, a smallest element in an array can be found by, copying the first element of the array in a variable smallestElement (you can give any name to this variable). Then comparing that variable with rest of the elements, if any element is found smaller than the smallestElement then store the value of that element into the variable. Repeat the process until all the elements are compared.

Program to find the smallest number in an array

In the following example, we have an array arr of numbers. We have copied the first element of this array into a variable smallestElement, this is to compare the other elements of the array to it. This is just the initialization of this variable, at the end of the program, this variable will hold the smallest number present in the array. Running a loop from 0 till arr.length (this represents the size of the array). At every loop iteration, each element of the array is compared with the smallestElement, if any element is smaller than smallestElement, then the value of that element is assigned to smallestElement. This process continues until the whole array is traversed.
In the end, the variable smallestElement holds the smallest element of the given array.

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

    //Initializing an int array
    int [] arr = new int [] {3, 8, 1, 12, 7, 99};
    //This element will store the smallest element of the array
    //Initializing with the first element of the array
    int smallestElement = arr[0];
    //Running the loop from first element till last element
    for (int i = 0; i < arr.length; i++) {
      //Compare each elements of array with smallestElement
      //If an element is smaller, store the element into smallestElement
      if(arr[i] < smallestElement)
        smallestElement = arr[i];
    }
    System.out.println("Smallest element of given array: " + smallestElement);
  }
}
```

Java Program to display first n or first 100 prime numbers

Program to display first n prime numbers

```
import java.util.Scanner;

class PrimeNumberDemo
{
   public static void main(String args[])
   {
      int n;
```

```
        int status = 1;
        int num = 3;
        //For capturing the value of n
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the value of n:");
        //The entered value is stored in the var n
        n = scanner.nextInt();
        if (n >= 1)
        {
           System.out.println("First "+n+" prime numbers are:");
           //2 is a known prime number
           System.out.println(2);
        }

        for ( int i = 2 ; i <=n ;   )
        {
           for ( int j = 2 ; j <= Math.sqrt(num) ; j++ )
           {
              if ( num%j == 0 )
              {
                 status = 0;
                 break;
              }
           }
           if ( status != 0 )
           {
              System.out.println(num);
              i++;
           }
           status = 1;
           num++;
        }
     }
}
```

Program to display first 100 prime numbers

To display the first 100 prime numbers, you can **either enter n value as 100 in the above program** OR write a program like this:

```
class PrimeNumberDemo
{
   public static void main(String args[])
   {
      int n;
      int status = 1;
      int num = 3;
      System.out.println("First 100 prime numbers are:");
      System.out.println(2);
      for ( int i = 2 ; i <=100 ;   )
      {
         for ( int j = 2 ; j <= Math.sqrt(num) ; j++ )
         {
            if ( num%j == 0 )
            {
               status = 0;
               break;
            }
         }
         if ( status != 0 )
         {
            System.out.println(num);
            i++;
         }
         status = 1;
         num++;
      }
   }
}
```

Java program to display prime numbers from 1-100 & 1 to n

The number which is only divisible by itself and 1 is known as prime number. E.g. 2, 3, 5, 7…are prime numbers. Here we will see two programs: 1) First program will print the prime numbers between 1 and 100 2) Second program takes the value of n (entered by user) and prints the prime numbers between 1 and n.

Program to display the prime numbers from 1 to 100

It will display the prime numbers between 1 and 100.

```
class PrimeNumbers
{
   public static void main (String[] args)
   {
       int i =0;
       int num =0;
       //Empty String
       String  primeNumbers = "";

       for (i = 1; i <= 100; i++)
       {
          int counter=0;
          for(num =i; num>=1; num--)
          {
             if(i%num==0)
             {
                counter = counter + 1;
             }
          }
          if (counter ==2)
          {
             //Appended the Prime number to the String
             primeNumbers = primeNumbers + i + " ";
          }
       }
       System.out.println("Prime numbers from 1 to 100 are :");
       System.out.println(primeNumbers);
   }
}
```

Program to display prime numbers from 1 to n

It will display all the prime numbers between 1 and n (n is the number, entered by user).

```
import java.util.Scanner;
class PrimeNumbers2
{
   public static void main (String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      int i =0;
      int num =0;
      //Empty String
      String  primeNumbers = "";
      System.out.println("Enter the value of n:");
      int n = scanner.nextInt();
      scanner.close();
      for (i = 1; i <= n; i++)
      {
         int counter=0;
         for(num =i; num>=1; num--)
         {
            if(i%num==0)
            {
                counter = counter + 1;
            }
         }
         if (counter ==2)
         {
            //Appended the Prime number to the String
            primeNumbers = primeNumbers + i + " ";
         }
      }
      System.out.println("Prime numbers from 1 to n are :");
      System.out.println(primeNumbers);
   }
}
```

Java program to check prime number

```
import java.util.Scanner;
class PrimeCheck
{
   public static void main(String args[])
   {
        int temp;
        boolean isPrime=true;
        Scanner scan= new Scanner(System.in);
        System.out.println("Enter any number:");
        //capture the input in an integer
        int num=scan.nextInt();
        scan.close();
        for(int i=2;i<=num/2;i++)
        {
           temp=num%i;
           if(temp==0)
           {
              isPrime=false;
              break;
           }
        }
        //If isPrime is true then the number is prime else not
        if(isPrime)
           System.out.println(num + " is a Prime Number");
        else
           System.out.println(num + " is not a Prime Number");
   }
}
```

Java Program to Check if given Number is Perfect Square

```
package com.beginnersbook;
import java.util.Scanner;
class JavaExample {

    static boolean checkPerfectSquare(double x)
    {

        // finding the square root of given number
        double sq = Math.sqrt(x);

        /* Math.floor() returns closest integer value, for
         * example Math.floor of 984.1 is 984, so if the value
         * of sq is non integer than the below expression would
         * be non-zero.
         */
        return ((sq - Math.floor(sq)) == 0);
    }

    public static void main(String[] args)
    {
        System.out.print("Enter any number:");
        Scanner scanner = new Scanner(System.in);
        double num = scanner.nextDouble();
        scanner.close();

        if (checkPerfectSquare(num))
                System.out.print(num+ " is a perfect square number");
        else
                System.out.print(num+ " is not a perfect square number");
    }
}
```

Java Program to Find square root of a Number without sqrt

```
package com.beginnersbook;
import java.util.Scanner;
class JavaExample {

    public static double squareRoot(int number) {
        double temp;

        double sr = number / 2;

        do {
                temp = sr;
                sr = (temp + (number / temp)) / 2;
        } while ((temp - sr) != 0);

        return sr;
    }

    public static void main(String[] args)
    {
        System.out.print("Enter any number:");
        Scanner scanner = new Scanner(System.in);
        int num = scanner.nextInt();
        scanner.close();

        System.out.println("Square root of "+ num+ " is: "+squareRoot(num));
    }
}
```

Java program to get IP address

1) Get the local host address by calling `getLocalHost()` method of InetAddress class.
2) Get the IP address by calling `getHostAddress()` method.

```
import java.net.InetAddress;

class GetMyIPAddress
{
   public static void main(String args[]) throws Exception
   {
      /* public static InetAddress getLocalHost()
       * throws UnknownHostException: Returns the address
       * of the local host. This is achieved by retrieving
       * the name of the host from the system, then resolving
       * that name into an InetAddress. Note: The resolved
       * address may be cached for a short period of time.
       */
      InetAddress myIP=InetAddress.getLocalHost();

      /* public String getHostAddress(): Returns the IP
       * address string in textual presentation.
       */
      System.out.println("My IP Address is:");
      System.out.println(myIP.getHostAddress());
  }
}
```

Java Program to get input from user

In this tutorial we are gonna see **how to accept input from user**. We are using Scanner class to get the input. In the below example we are getting input String, integer and a float number. For this we are using following methods:
1) public String nextLine(): For getting input String
2) public int nextInt(): For integer input
3) public float nextFloat(): For float input

Example:

```
import java.util.Scanner;

class GetInputData
{
  public static void main(String args[])
  {
     int num;
     float fnum;
     String str;

     Scanner in = new Scanner(System.in);
```

```
      //Get input String
      System.out.println("Enter a string: ");
      str = in.nextLine();
      System.out.println("Input String is: "+str);

      //Get input Integer
      System.out.println("Enter an integer: ");
      num = in.nextInt();
      System.out.println("Input Integer is: "+num);

      //Get input float number
      System.out.println("Enter a float number: ");
      fnum = in.nextFloat();
      System.out.println("Input Float number is: "+fnum);
   }
}
```

https://beginnersbook.com/2017/09/java-examples/

Java Program to calculate and display Student Grades

This program calculates the grade of a student based on the marks entered by user in each subject. Program prints the grade based on this logic.
If the average of marks is >= 80 then prints Grade 'A'
If the average is <80 and >=60 then prints Grade 'B'
If the average is <60 and >=40 then prints Grade 'C'
else prints Grade 'D'

To understand this Program you should have the knowledge of following concepts of Java:

- Java For Loop
- Arrays in Java
- if..else-if in Java

Example: Program to display the grade of student

```
import java.util.Scanner;

public class JavaExample
{
    public static void main(String args[])
    {
        /* This program assumes that the student has 6 subjects,
         * thats why I have created the array of size 6. You can
         * change this as per the requirement.
         */
        int marks[] = new int[6];
        int i;
        float total=0, avg;
        Scanner scanner = new Scanner(System.in);

        for(i=0; i<6; i++) {
           System.out.print("Enter Marks of Subject"+(i+1)+":");
           marks[i] = scanner.nextInt();
           total = total + marks[i];
        }
        scanner.close();
        //Calculating average here
        avg = total/6;
        System.out.print("The student Grade is: ");
        if(avg>=80)
        {
            System.out.print("A");
        }
        else if(avg>=60 && avg<80)
        {
           System.out.print("B");
        }
        else if(avg>=40 && avg<60)
        {
            System.out.print("C");
        }
        else
        {
            System.out.print("D");
        }
    }
}
```

Try Catch in Java – Exception handling

ry catch block is used for exception handling in Java. The code (or set of statements) that can throw an exception is placed inside **try block** and if the exception is raised, it is handled by the corresponding **catch block**. In this guide, we will see various examples to understand how to use try-catch for exception handling in java.

Try block in Java

As mentioned in the beginning, try block contains set of statements where an exception can occur. A try block is always followed by a catch block or finally block, if exception occurs, the rest of the statements in the try block are skipped and the flow immediately jumps to the corresponding catch block.

Note: A try block must be followed by catch blocks or finally block or both.

Syntax of try block with catch block

```
try{
   //statements that may cause an exception
}catch(Exception e){
  //statements that will execute when exception occurs
}
```

Syntax of try block with finally block

```
try{
   //statements that may cause an exception
}finally{
  //statements that execute whether the exception occurs or not
}
```

Syntax of try-catch-finally in Java

```
try{
   //statements that may cause an exception
}catch(Exception e){
  //statements that will execute if exception occurs
}finally{
  //statements that execute whether the exception occurs or not
}
```

Note: It is upto the programmer to choose which statements needs to be placed inside try block. If programmer thinks that certain statements in a program can throw a exception, such statements can be enclosed inside try block and potential exceptions can be handled in catch blocks.

Catch block in Java

A catch block is where you handle the exceptions, this block must immediately placed after a try block. **A single try block can have several catch blocks associated with it**. You can catch different exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes. For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.

Syntax of try catch in java

```
try
{
     //statements that may cause an exception
}
catch (exception(type) e(object))
{
     //error handling code
}
```

Example: try catch in Java

If an exception occurs in try block then the control of execution is passed to the corresponding catch block. As discussed earlier, a single try block can have multiple catch blocks associated with it, you should place the catch blocks in such a way that the generic exception handler catch block is at the last(see in the example below).

The **generic exception handler** can handle all the exceptions but you should place is at the end, if you place it at the before all the catch blocks then it will display the generic message. You always want to give the user a meaningful message for each type of exception rather then a generic message.

```
class Example1 {
   public static void main(String args[]) {
      int num1, num2;
      try {
         /* We suspect that this block of statement can throw
          * exception so we handled it by placing these statements
          * inside try and handled the exception in catch block
          */
         num1 = 0;
         num2 = 62 / num1;
         System.out.println(num2);
         System.out.println("Hey I'm at the end of try block");
      }
      catch (ArithmeticException e) {
         /* This block will only execute if any Arithmetic exception
          * occurs in try block
          */
         System.out.println("You should not divide a number by zero");
      }
      catch (Exception e) {
         /* This is a generic Exception handler which means it can handle
          * all the exceptions. This will execute if the exception is not
          * handled by previous catch blocks.
          */
         System.out.println("Exception occurred");
      }
      System.out.println("I'm out of try-catch block in Java.");
   }
}
```

Output:

```
You should not divide a number by zero
I'm out of try-catch block in Java.
```

Multiple catch blocks in Java

The example we seen above is having multiple catch blocks, let’s see few rules about multiple catch blocks with the help of examples. To read this in detail, see catching multiple exceptions in java.

1. As I mentioned above, a single try block can have any number of catch blocks.

2. A generic catch block can handle all the exceptions. Whether it is ArrayIndexOutOfBoundsException or ArithmeticException or NullPointerException or any other type of exception, this handles all of them. To see the examples of NullPointerException and ArrayIndexOutOfBoundsException, refer this article: Exception Handling example programs.

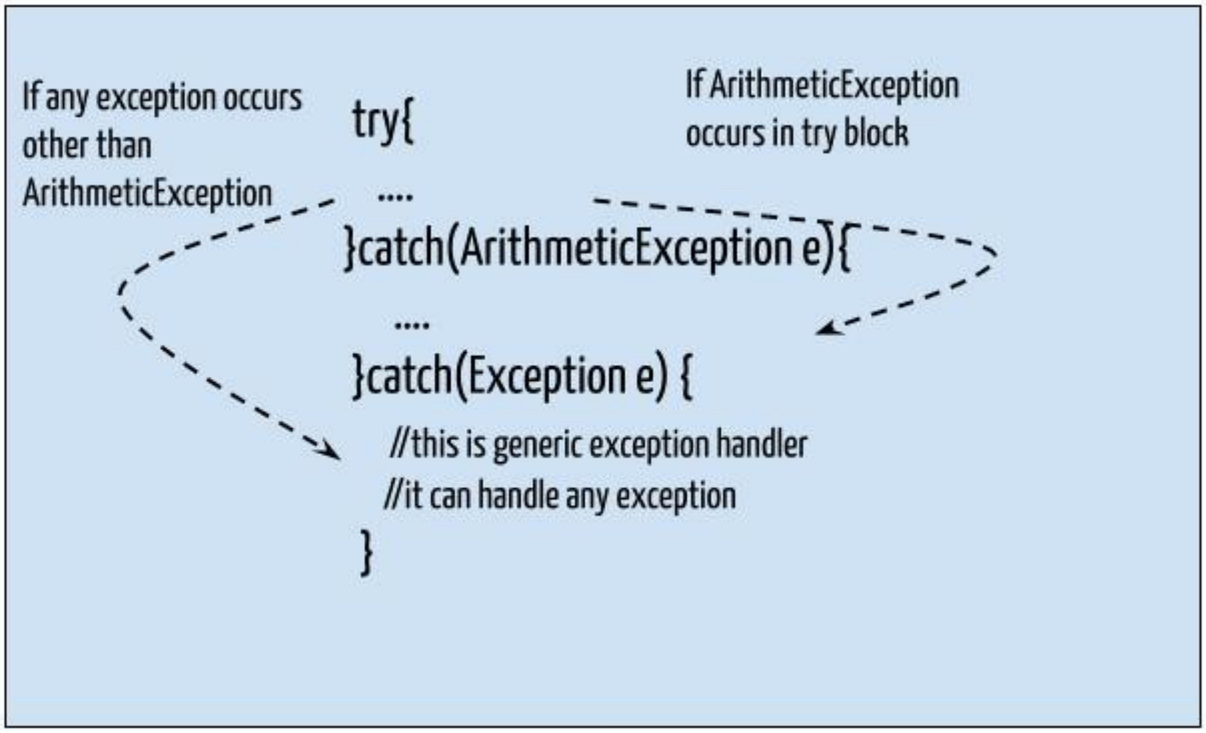
```
catch(Exception e){
  //This catch block catches all the exceptions
}
```

If you are wondering why we need other catch handlers when we have a generic that can handle all. This is because in generic exception handler you can display a message but you are not sure for which type of exception it may trigger so it will display the same message for all the exceptions and user may not be able to understand which exception occurred. Thats the reason you should place is at the end of all the specific exception catch blocks

3. If **no exception** occurs in try block then the **catch blocks are completely ignored**.

4. Corresponding catch blocks execute for that specific type of exception:
catch(ArithmeticException e) is a catch block that can handle ArithmeticException
catch(NullPointerException e) is a catch block that can handle NullPointerException

5. You can also throw exception, which is an advanced topic and I have covered it in separate tutorials: user defined exception, throws keyword, throw vs throws.


If any exception occurs other than ArithmeticException
try{
....
}catch(ArithmeticException e){
If ArithmeticException occurs in try block
....
}catch(Exception e) {
//this is generic exception handler
//it can handle any exception
}

Example of Multiple catch blocks

```
class Example2{
   public static void main(String args[]){
     try{
         int a[]=new int[7];
         a[4]=30/0;
         System.out.println("First print statement in try block");
     }
     catch(ArithmeticException e){
        System.out.println("Warning: ArithmeticException");
     }
     catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Warning: ArrayIndexOutOfBoundsException");
```

```
      }
      catch(Exception e){
         System.out.println("Warning: Some Other exception");
      }
    System.out.println("Out of try-catch block...");
  }
}
```
Output:

```
Warning: ArithmeticException
Out of try-catch block...
```
In the above example there are multiple catch blocks and these catch blocks executes sequentially when an exception occurs in try block. Which means if you put the last catch block (catch(Exception e)) at the first place, just after try block then in case of any exception this block will execute as it can handle all exceptions. This catch block should be placed at the last to avoid such situations.

Exception handling in Java with examples

Exception handling is one of the most important feature of java programming that allows us to handle the **runtime errors** caused by exceptions. In this guide, you will learn what is an exception, types of it, exception classes and how to handle exceptions in java with examples.

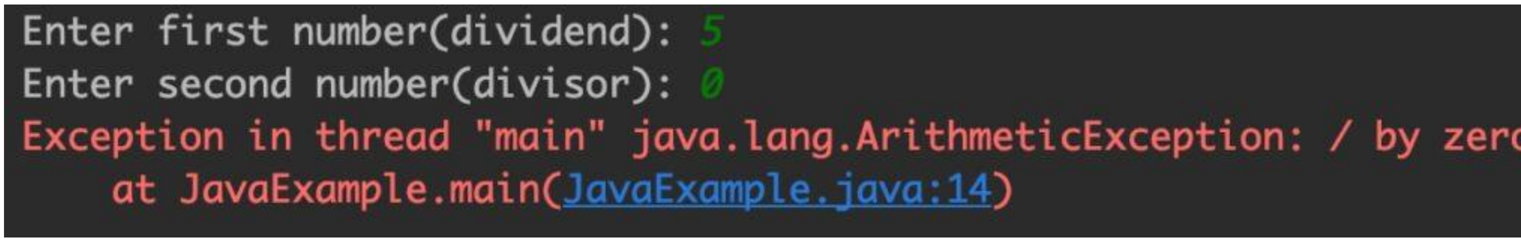
What is an exception?

An Exception is an unwanted event that **interrupts the normal flow of the program**. When an exception occurs program execution gets terminated. In such cases we get a system generated error message.

The good thing about exceptions is that java developer can handle these exception in such a way so that the program doesn't get terminated abruptly and the user get a meaningful error message.

For example: You are writing a program for division and both the numbers are entered by user. In the following example, user can enter any number, if user enters the second number (divisor) as 0 then the program will terminate and throw an exception because dividing a number by zero gives undefined result. To get the user input, we are using Scanner class. Notice the output of the program.

```
import java.util.Scanner;
public class JavaExample {

  public static void main(String[] args) {

    int num1, num2;
    Scanner scan = new Scanner(System.in);
    System.out.print("Enter first number(dividend): ");
    num1 = scan.nextInt();
    System.out.print("Enter second number(divisor): ");
    num2 = scan.nextInt();

    int div = num1/num2;
    System.out.println("Quotient: "+div);
  }
}
```
Output:



```
Enter first number(dividend): 5
Enter second number(divisor): 0
Exception in thread "main" java.lang.ArithmeticException: / by zer
	at JavaExample.main(JavaExample.java:14)
```

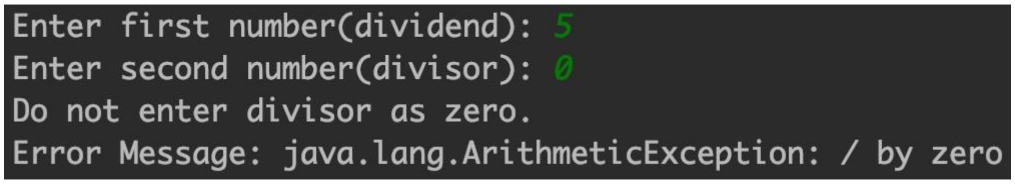
As you can see, the user input caused the program to throw Arithmetic exception, however this is not a good programming practice to leave such exceptions unhandled. Let’s handle this exception.

Exception Handling in Java

Here, we are trying to handle the exception that is raised in the above program. You can see that the program ran fine and gave a meaningful error message which can be understood by the user.

Note: Do not worry about the try and catch blocks as we have covered these topics in detail in separate tutorials. For now just remember that the code that can throw exception needs to be inside try block and the catch block follows the try block, where the exception error message is set.

```
import java.util.Scanner;
public class JavaExample {

  public static void main(String[] args) {

    int num1, num2;
    Scanner scan = new Scanner(System.in);
    System.out.print("Enter first number(dividend): ");
    num1 = scan.nextInt();

    System.out.print("Enter second number(divisor): ");
    num2 = scan.nextInt();
    try {
      int div = num1 / num2;
      System.out.println("Quotient: "+div);
    }catch(ArithmeticException e){
      System.out.println("Do not enter divisor as zero.");
      System.out.println("Error Message: "+e);
    }

  }
}
```
Output:



```
Enter first number(dividend): 5
Enter second number(divisor): 0
Do not enter divisor as zero.
Error Message: java.lang.ArithmeticException: / by zero
```

If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user.

These system generated messages are **not user friendly** so a user will not be able to understand what went wrong. In order to let them know the reason in simple language, we handle exceptions. We handle such exceptions and then prints a user friendly warning message to user, which lets them correct the error as most of the time **exception occurs due to bad data provided by user**.

Why we handle the exceptions?

Exception handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements, that occur after the statement that caused the exception will not execute and the program will terminate abruptly. By handling we make sure that all the statements execute and the flow of execution of program doesn't break.

Why an exception occurs?

There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc. Let's see few scenarios:
1. ArithmeticException:
We have already seen this exception in our example above. This exception occurs when we divide a number by zero. If we divide any number by zero.

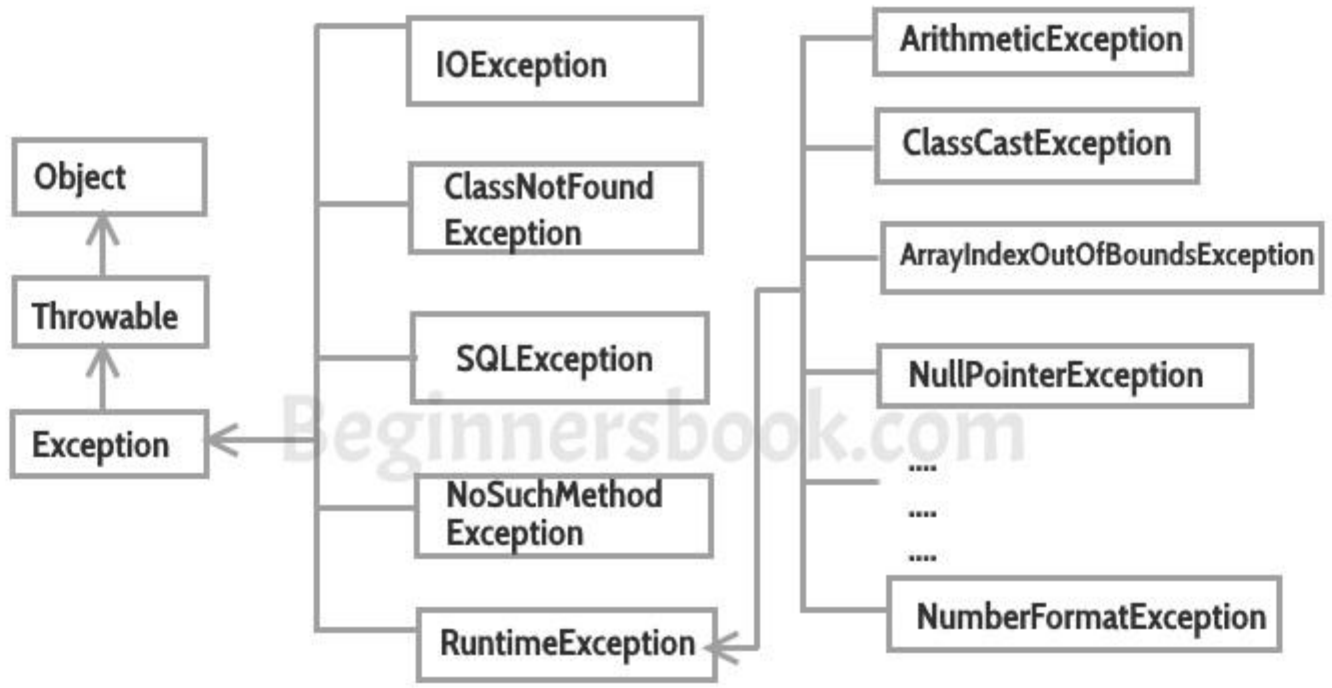
```
int num = 25/0;//ArithmeticException
```
2. NullPointerException:
When a variable contains null value and you are performing an operation on the variable. For example, if a string variable contains null and you are comparing with another string. Another example is when you are trying to print the length of the string that contains null.

```
String str = null;

//NullPointerException
System.out.println(str.length());
```
3. NumberFormatException:
This exception occurs where there is a type mismatch. Let's say you are trying to perform an arithmetic operator on a string variable.

```
String str = "beginnersbook.com";

//NumberFormatException
int num=Integer.parseInt(str);
```

4. ArrayIndexOutOfBoundsException:
When you are trying to access the array index which is beyond the size of array. Here, we are trying to access the index 8 (9th element) but the size of the array is only 3. This exception occurs when you are accessing index which doesn't exist.

```
int arr[]=new int[3];

//ArrayIndexOutOfBoundsException
arr[8]=100;
```

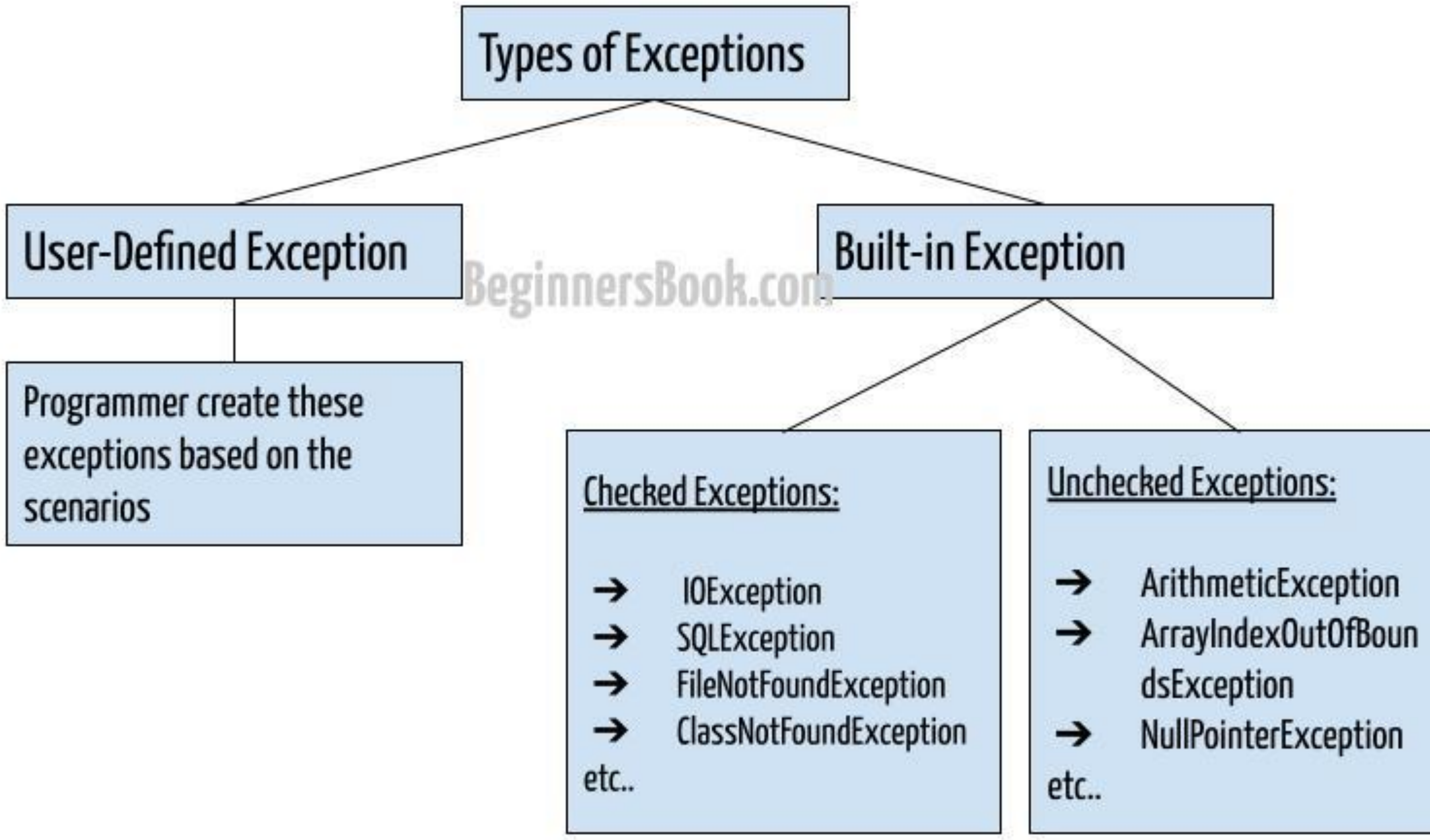


Difference between error and exception

Errors indicate that something went wrong which is not in the scope of a programmer to handle. You cannot handle an error. Also, the error doesn't occur due to bad data entered by user rather it indicates a system failure, disk crash or resource unavailability.

Exceptions are events that occurs during runtime due to bad data entered by user or an error in programming logic. A programmer can handle such conditions and take necessary corrective actions. Few examples:
NullPointerException – When you try to use a reference that points to null.
ArithmeticException – When bad data is provided by user, for example, when you try to divide a number by zero this exception occurs because dividing a number by zero is undefined.
ArrayIndexOutOfBoundsException – When you try to access the elements of an array out of its bounds, for example array size is 5 (which means it has five elements) and you are trying to access the 10th element.

Types of exceptions



There are two types of exceptions in Java:
1) Checked exceptions
2) Unchecked exceptions

I have covered these topics in detail in a separate tutorial: Checked and Unchecked exceptions in Java.

1) Checked exceptions

All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error. For example, SQLException, IOException, ClassNotFoundException etc.

2) Unchecked Exceptions

Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit.

For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. The examples that we seen above were unchecked exceptions.

Note: Compiler doesn’t enforce you to catch such exceptions or ask you to declare it in the method using throws keyword.

Frequently used terms in Exception handling

try: The code that can cause the exception, is placed inside try block. The try block detects whether the exception occurs or not, if exception occurs, it transfer the flow of program to the corresponding catch block or finally block. A try block is always followed by either a catch block or finally block.

catch: The catch block is where we write the logic to handle the exception, if it occurs. A catch block only executes if an exception is caught by the try block. A catch block is always accompanied by a try block.

finally: This block always executes whether an exception is occurred or not.

throw: It is used to explicitly throw an exception. It can be used to throw a checked or unchecked exception.

throws: It is used in method signature. It indicates that this method might throw one of the declared exceptions. While calling such methods, we need to handle the exceptions using try-catch block.

What topics are covered in the next tutorials

You can read these topics to understand the exception handling concept in detail. You can also practice various programs covered in the following tutorials.

1. Try-catch in Java
2. Nested Try Catch
3. Checked and unchecked exceptions
4. Finally block in Java
5. try-catch-finally
6. finally block & return statement
7. Throw exception in Java
8. Example of throw keyword
9. Example of throws clause
10. Throws in Java
11. throw vs throws
12. Exception handling examples

Java Scanner class with examples

you will learn **Java Scanner class** and how to use it in java programs to get the user input. This is one of the important classes as it provides you various methods to capture different types of user entered data. In this guide, we will discuss java **Scanner class methods** as well as examples of some of the important methods of this class.

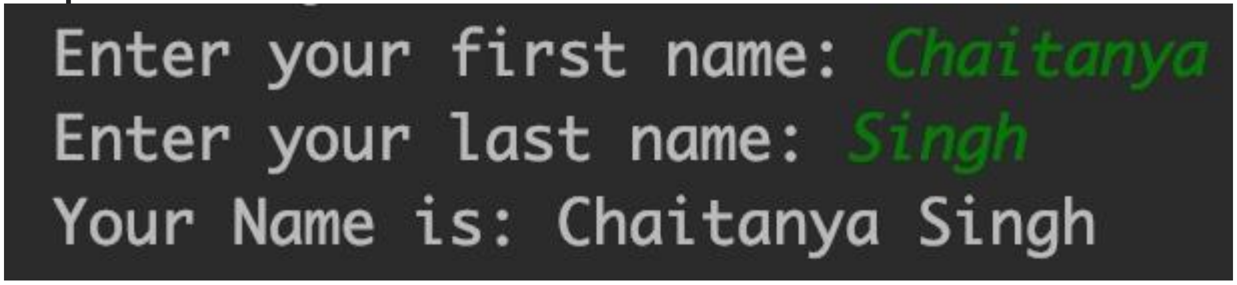
The `Scanner` class is present in the `java.util` package so be sure import this package when you are using this class.

Example 1: Read user input text using Scanner

This is the first example of Scanner class, let's discuss everything in detail.
1. The first statement `import java.util.Scanner;` is mandatory when using Scanner class. Alternatively, You can also import Scanner like this: `java.util.*`, this will import all the classes present in the `java.util` package.
2. While creating an object of Scanner class, we passed the `System.in` in the Scanner class constructor. This is to read the data from standard input.
3. We have used the `nextLine()` method of Scanner as this method is used to read the **line of text** entered by the user.

```
import java.util.Scanner;
public class JavaExample {
  public static void main(String[] args) {

    // creating a scanner
    Scanner scan = new Scanner(System.in);

    System.out.print("Enter your first name: ");

    // read user input and store it in a string variable
    String firstName = scan.nextLine();

    System.out.print("Enter your last name: ");

    // read second input
    String lastName = scan.nextLine();

    // prints the user entered data
    System.out.println("Your Name is: "+firstName+" "+lastName);

    // close scanner
    scan.close();
  }
}
```

Output:



```
Enter your first name: Chaitanya
Enter your last name: Singh
Your Name is: Chaitanya Singh
```

Java Scanner class methods

In the above example, we have seen the use of `nextLine()` method. There are several methods available in this class. Let’s list down some of the frequently used **methods of Scanner class**.

METHOD	**DESCRIPTION**
`nextInt()`	It reads an `int` value entered by the user
`nextFloat()`	It reads a `float` value entered by the user
`nextBoolean()`	It reads a `boolean` value entered by the user
`nextLine()`	It reads a line of text entered by the user
`next()`	This method reads a word entered by the user
`hasNextLine()`	Checks if there is another line of text entered by the user
`hasNextInt()`	Checks if there is another int value input by the user

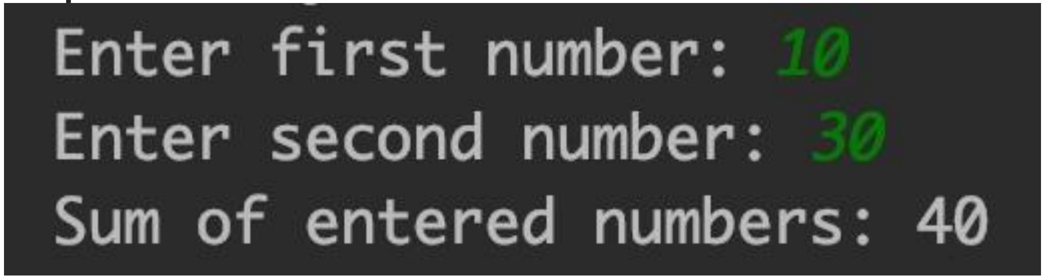
`reset()`	It resets the scanner
`toString()`	It is used to get string representation of user input
`nextByte()`	This method reads a `byte` value entered by the user
`nextDouble()`	This method reads a `double` value entered by the user
`nextShort()`	It reads a `short` value entered by the user
`nextLong()`	It reads a `long` value entered by the user

Example 2: Java Scanner nextInt() method

Here, we are demonstrating the use of `nextInt()` method. This method reads the integer value entered by the user. We are using the nextInt() method twice to get two numbers from user and then we are printing the sum of entered numbers.

```java
import java.util.Scanner;
public class JavaExample {
  public static void main(String[] args) {

    // creating a scanner
    Scanner scan = new Scanner(System.in);

    System.out.print("Enter first number: ");

    // read user input and store it in an int variable
    int num1 = scan.nextInt();

    System.out.print("Enter second number: ");

    // read second input
    int num2 = scan.nextInt();

    // prints the sum of entered numbers
    System.out.print("Sum of entered numbers: "+(num1+num2));

    // close scanner
    scan.close();
  }
}
```

Output:



```
Enter first number: 10
Enter second number: 30
Sum of entered numbers: 40
```

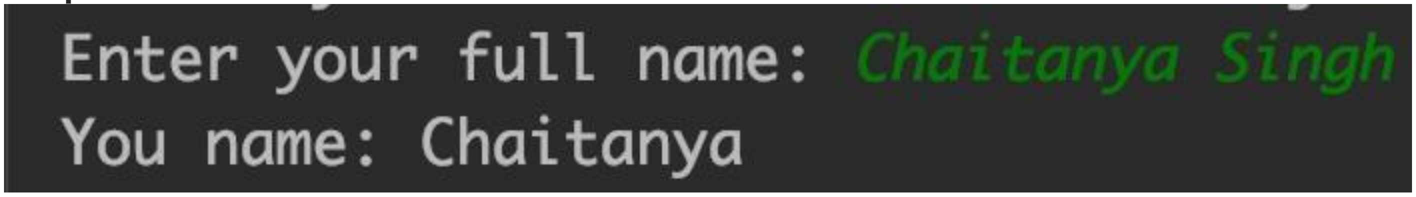
Example 3: Java Scanner next() method

The `next()` method is different from the `nextLine()` method. Where the `nextLine()` method is used to read the line of text, the `next()` method reads the word entered by the user. The `next()` method reads the input until a whitespace is encountered. It doesn't read the user input after whitespace.

In the following example, user is asked to enter the full name, however the `next()` method captured only the first name as it stopped reading input as soon as it found a whitespace. We will revisit the same example next with `nextLine()` method to get the desired output.

```
import java.util.Scanner;
public class JavaExample {
  public static void main(String[] args) {

    // creating a scanner
    Scanner scan = new Scanner(System.in);

    System.out.print("Enter your full name: ");

    // read the user input using next() method
    // This method only reads a word, which means
    // if there is a whitespace between words, the
    // first word is scanned and second skipped
    String name = scan.next();

    System.out.print("You name: "+name);

    // close scanner
    scan.close();
  }
}
```

Output:



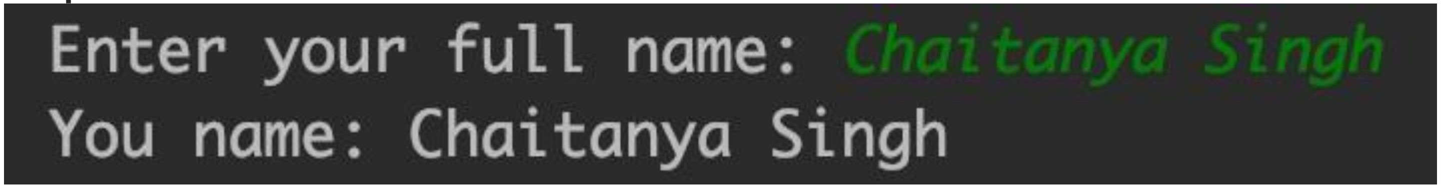
```
Enter your full name: Chaitanya Singh
You name: Chaitanya
```

Example 4: Java Scanner nextLine() method

Let’s revisit the same example. As you can see, using nextLine(), we can read the complete user input. This is because this method reads a complete line.

```
import java.util.Scanner;
public class JavaExample {
  public static void main(String[] args) {

    // creating a scanner
    Scanner scan = new Scanner(System.in);

    System.out.print("Enter your full name: ");

    // The nextLine() method works different from
    // the next() method, unlike next(), it reads the
    // complete line
    String name = scan.nextLine();

    System.out.print("You name: "+name);

    // close scanner
    scan.close();
  }
}
```

Output:



```
Enter your full name: Chaitanya Singh
You name: Chaitanya Singh
```

Example 5: Java Scanner useDelimiter() Method

The `userDelimiter()` method is used to specify a delimiter character in the input. In the following example, we have specified ‘/’ as a delimiter. We are also using hasNext() method of Scanner class to read the input separated by the delimiter.

```
import java.util.Scanner;
public class JavaExample {
  public static void main(String args[]){
    // Initializing a Scanner object
    Scanner scan = new Scanner("BeginnersBook/Chaitanya/Website");

    //Initialize the delimiter in useDelimiter() method
    scan.useDelimiter("/");

    //printing the strings separated by delimiter
    while(scan.hasNext()){
      System.out.println(scan.next());
    }
    scan.close();
  }
}
```