CodInClub
Powered By BridgeLabz

Programming Concept Practiced

1. **Java Strings -** Java Strings have been already used multiple times in our programs staring from the Hello World program.

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

   a. Here, "Hello World" is an example of a string. We use double quotes "" to represent strings.
   b. Strings are a collection of characters that are used to represent textual data in programming. For example, the string `"Hello World"` consists of characters `'H'`, `'e'`, `'l'`, and so on.

2. **Create Strings -** Just like other variables that are created using datatype, String variables can also be created using Java Built-In ***String*** datatype

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

        // create a string variable
        String greeting1 = "Hello";
        String greeting2 = "How are
you?";

        // print the string
        System.out.println(greeting1);
        System.out.println(greeting2);
    }
}
```

3. **Use of *"\"* Escape Sequence -**Escape sequences allow us to include special characters in strings. Similar to using ***"\n"*** for newline. SInce we wanted double quotes to be

01 Java String Notes


```
        String text = "\"What's there?\",
asked Paul";
        System.out.println(text);
    }
}
Output of the Program - "What's there?",
asked Paul
```

4. **Take String Input -** String input can be taken using the methods of the ***Scanner*** class as we have done for other datatypes.

    a. Use the ***next()*** method of the ***Scanner*** class to take string input from the user. This takes in the single word until the space is encountered as input
    b. The ***Scanner*** class also provides the ***nextLine()*** method that is used to take the whole line of text as input from the user.
    c. Using Inbuilt method ***trim()*** in ***String*** class to remove all the leading and trailing spaces from the text

```
class Main {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

      // Take input from the user for the
name and quote
      System.out.print("Enter your name and
favorite quote: ");
      String name = sc.next();
      String quote = sc.nextLine().trim();

      // Display the name and quote
      System.out.println(name + " said, \"" +
quote + "\"");
   }
}
```

5. **String Arrays -** We can create an array of strings in Java similar to arrays of any datatypes we have seen before. It can be initialized during declaration of variable or can be based on user input. The String Array can be accessed using for-each loop as well.

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


        // Create an array of strings for
roll numbers and Name

String[] rollNumbers = {"001", "002", "003", "004"};

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

```
for (int index = 0; index < names.length; ++index) {
            names[index] = input.nextLine();
        }

        // access elements of the string
array using for each statement
         int i = 0;
        for (String name : names) {
            System.out.println("Name: " +
name + "\tRoll Number: "
                                        +
rollNumbers[i++]);
        }
    }
}
```

6. **String as Method Parameters -** We can also pass String as well as String array as method parameters and also as return statements
7. **String Class Built-In Methods -** String are one of the most frequently used datatypes in programming. To make working with the String datatype easier, there are several String methods, here are some of the most common ones used in programming
   a. ***trim()*:** We have already seen this which is essentially to remove all the leading and trailing spaces from the text
   b. ***length()*:** finds the length of the string
   c. ***charAt()*:** returns a character from the string
   d. ***concat()*:** joins two strings together
   e. ***equals()*:** compares two strings
   f. ***toCharArray()*:** converts the string to a character array
   g. ***substring(int beginIndex, int endIndex)***: Returns a string that is a substring of this string. The substring begins at the specified beginIndex and ends at index endIndex - 1. Thus the length of the substring is endIndex-beginIndex.
   h. ***toLowerCase()***: Converts all of the characters in the text to lower case
   i. ***toUpperCase()***: Converts all of the characters in the text to upper case
8. **ASCII Character Codes -** The ASCII (American Standard Code for Information Interchange) codes for the uppercase letters A to Z are 65 to 90, and the codes for the lowercase letters a to z are 97 to 122

01 Java String Notes

ASCII Character Codes

Lowercase Letters		Uppercase Letters	
a = 97	n = 110	A = 65	N = 78
b = 98	o = 111	B = 66	O = 79
c = 99	p = 112	C = 67	P = 80
d = 100	q = 113	D = 68	Q = 81
e = 101	r = 114	E = 69	R = 82
f = 102	s = 115	F = 70	S = 83
g = 103	t = 116	G = 71	T = 84
h = 104	u = 117	H = 72	U = 85
i = 105	v = 118	I = 73	V = 86
j = 106	w = 119	J = 74	W = 87
k = 107	x = 120	K = 75	X = 88
l = 108	y = 121	L = 76	Y = 89
m = 109	z = 122	M = 77	Z = 90

```
// Write a Program to find the characters
and their ASCII values in a string
// and display it

class Main {
  // Method to find the characters and their
ASCII values in a string and
  // return them in a 2D array
  public static int[]
[] findASCIIValues(String text) {
      int[][] charASCIIValues = new
int[text.length()][2];

      for
(int i = 0; i < text.length(); i++) {
          charASCIIValues[i]
[0] = text.charAt(i);
          charASCIIValues[i]
[1] = text.charAt(i);
      }

      return charASCIIValues;
  }

  // Method to display the characters and
their ASCII values in a string
  public static void
displayASCIIValues(String text, int[]
[] charASCIIValues) {
      System.out.println("Characters and
their corresponding ASCII values " +
                         "for " + text + "
is:");

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

System.out.println((char)charASCIIValues[i]
[0] + "-" +

 charASCIIValues[i][1]);
      }
  }
```

```
    displayASCIIValues(text,
charASCIIValues);
  }
}
```

The Output of running the program

```
Characters and their corresponding ASCII
values for AaBb is:
A-65
a-97
B-66
b-98
```

9. **Java Exceptions -** An exception is an unexpected event that occurs during program execution. It affects the flow of the program instructions which can cause the program to terminate abnormally. An exception can occur for many reasons. Some of them are:
    a. Invalid user input
    b. Device failure
    c. Loss of network connection
    d. Physical limitations (out of disk memory)
    e. Code errors
    f. Accessing unavailable Memory
    g. Opening an unavailable file
10. **Java Exception Hierarchy -** Here is a simplified diagram of the exception hierarchy in Java. Java ***Throwable*** class is the root class in the hierarchy. It Splits into ***Error*** and ***Exception***
    a. ***java.lang.Error*** **-** ***Error*** represents irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc.
    b. ***java.lang.Exception*** **-** Exceptions can be caught and handled by the program.
        i. When an exception occurs, it creates an object. This object is called the exception object.
        ii. It contains information about the exception such as the name and description of the exception and the state of the program when the exception occurred.

11. **Java Exception Types -** The Exception class is further divided into ***RuntimeException*** which are Unchecked Exceptions and Checked Exceptions like ***IOException***.
    a. ***java.lang.RuntimeException***: These exceptions are raised during runtime due to programming errors. They are not checked at compile time and hence are also called **unchecked exceptions**. Some examples are
        i. Improper use of an API - ***IllegalArgumentException***
        ii. Null pointer access (missing the initialization of a variable) - ***NullPointerException***
        iii. Out-of-bounds array access - ***ArrayIndexOutOfBoundsException***
        iv. Dividing a number by 0 - ***ArithmeticException***

    Example for unchecked exceptions. It is not compulsory to catch unchecked exceptions while writing code. Hence the following code will compile. However, when you run the code you will encounter runtime exceptions.

```
class Main {
   public static void main(String[] args) {
      // Following code generate Runtime
exception but not at compile time
      int divideByZero = 5 / 0;
   }
}
```

    Refactor the code to handle the RuntimeException using the try-catch block. It is a good programming practice to write the code with a try-catch. You can catch specific ***ArithmeticException*** or generic ***Exception*** or both. All will work.

```
// Program demonstrates handling of
Exceptions.

class Main {
   public static void main(String[] args) {
     // Following code generate Runtime
exception which is ArithmeticException
     try {
        int divideByZero = 5 / 0;
     } catch (ArithmeticException e) {

System.out.println("ArithmeticException =>
" + e.getMessage());
     } catch (Exception e) {
```

Compiler during compile-time and need to be handled by the programmer using the try...catch block. ***java.io.IOException*** is regularly used during the handling of File Operations. Also All User-Defined Exceptions are Checked Exceptions.

The following example is about reading the contents from a File using the File and the Scanner class. In this case, it is mandatory to handle FileNotFoundException by catching it with ***java.io.FileNotFoundException*** or with ***java.io.IOException***. Please note both FileNotFoundException and IOException cannot be used together in the catch block as anyone has already handled the file not found exception.

It's a good programming practice to catch the Checked Exception as well as generic ***Exception*** to handle other runtime exceptions.

```
// Following Program demonstrates handling
FileNotFoundException

class Main {
   public static void main(String[] args) {
      try {

 java.io.File file = new java.io.File("file.txt");

 java.util.Scanner sc = new java.util.Scanner(file);

} catch (java.io.FileNotFoundException e) {

 System.out.println("FileNotFoundException
occurred");
      }  catch (Exception e) {
         System.out.println("Exception
occurred");
      }
   }
}

// Following Program demonstrates handling
IOException. Exception Handling is
// achieved using FileNotFoundException or
IOException. Other runtime
// exceptions is handled with Exception

class Main {
   public static void main(String[] args) {
      try {

 java.io.File file = new java.io.File("file.txt");

 java.util.Scanner sc = new java.util.Scanner(file);
      } catch (java.io.IOException e) {
```

```
}
```

Best Programming Practice

1. Use Variables including for Fixed, User Inputs, and Results
2. Use Methods instead of writing code in the main() function
3. Proper naming conventions for all variables and methods
4. Proper Program Name and Class Name
5. Handle Checked and Unchecked Exceptions wherever possible
6. Proper Method Name which indicates action taking inputs and providing result

Sample Program 1: Create a program to find all the occurrences of a character in a string using charAt() method

   a. Take user input for the String and occurrences of the Character to find
   b. Write a method to find all the occurrences of the characters.
      i. The logic used is to first find the number of occurrences of the character and
      ii. then create an array to store the indexes of the character
   c. Call the method in the main and display the result

```
// Program to find all the occurrences of a
character in a string
import java.util.Scanner;
class StringAnalyzer {
  // Method to find all the index of a
character in a string using charAt()
  // method and return them in an array

public static int[] findAllIndexes(String text, char ch) {
    // The count is used to find the number
of occurrences of the character
    int count = 0;
    for (int i = 0; i < text.length(); i++) {
        if (text.charAt(i) == ch) {
            count++;
        }
```

01 Java String Notes


```
            if (text.charAt(i) == ch) {
                indexes[j] = i;
                j++;
            }
        }
        return indexes;
    }
    public static void main(String[] args) {
        // Take user input for Text and Character
to check Occurrences
        Scanner sc = new Scanner(System.in);

        System.out.print(Enter a text: ");
        String text = sc.nextLine();

System.out.print("Enter a character to find the occurrences: ");
        char ch = sc.next().charAt(0);

        // Find the occurrences of the character
        int[] indexes = findAllIndexes(text, ch);

        // Display the occurrences of the
character

System.out.println("Indexes of the character '"
+ ch + "': ");
        for (int i = 0; i < indexes.length; i++)
{
            System.out.print(indexes[i] + " ");
        }
    }
}
```