18.1.1. Introduction to for-loop

04:39

A `for-loop` is used to iterate over a range of values using a `loop counter`, which is a variable taking a range of values in some orderly sequence (e.g., starting at **0** and ending at **10** in increments of **1**).

`Loop counter` changes with each iteration of the loop, providing a unique value for each individual iteration. The `loop counter` is used to decide when to terminate the loop.

```
for (initialization; condition; update) {
    statement(s);
}
```

1. The `initialization` expression initializes the loop counter; it's executed once at the beginning of the loop.
2. The loop continues to execute as long as the `condition` evaluates to `true`.
3. The `update` expression is executed after each iteration through the loop, to increment, decrement or change the loop counter.

Example:

```
for (int i = 0; i < 10; i++) {
   System.out.println(i);
}
```

1. Above **for-loop** statement declares and initializes an integer variable `i` (which is the loop counter) as part of the `initialization` expression.
2. It increments the variable `i` by 1 in the `update` expression `i++`.
3. The `condition` is `i < 10`. The `for` loop keeps on executing the code inside the loop body as long as this `condition` evaluates to `true`. And the loop terminates when the `condition` evaluates to `false`.

Explorer

ForDemo....

SUBMIT

```
1   package q10880;
2
3   public class ForDemo {
4
5       public static void main(String[] args) {
6
7
8
9           // Fill in the missing code for the for-loop that prints 1 to 10
10
11          int i=1;
12
13          while (i>0 &&i<=10) {
14
15              System.out.println(i);
16
17              i++;
18
19          }
20
21      }
22
23  }
24
```

18.1.2. Practising for loop

00:39

Write a class `APowerN` with a **public** method `powerOf` that takes two parameters `A` and `N` are of type `int`. Complete the missing code in the below program such that it should print the value of `aPowerN`.

Assumptions:

1. `A` and `N` are two positive and non-zero numbers

For example:

```
Cmd Args : 2 8
APowerN = 256
```

Hint: You can use a `for` loop to multiply `A` with itself `N` number of times.

Variable `aPowerN` can be used to store the computed value of A^N inside the `for` loop.

Note: Please don't change the package name.

Explorer

APowerN... APowerN...

SUBMIT

```java
package q10882;

public class APowerN {

    public void powerOf(int A, int N) {

        int ans = 1;

        // Fill in the missing code using the for loop calculate A to the power N

        while( N!=0) {

            ans*=A;

            --N;

        }

        System.out.println("APowerN = " + ans);

    }

}
```

18.1.3. Understanding for loop

00:25

Factorial of a non-negative integer `n` is denoted by `n!`. It is the product of all positive integers **less than or equal to** `n`.

Write a class `FactorialCalculator` with a **public** method `factorial`. The method receives one parameter `n` of type `int`. Fill in the missing code in the below program to calculate the factorial of a given number and print the output.

For example:

```
Cmd Args: 4
Factorial of 4 = 24
```

Hint: You can use the integer variable `factorial` initialized to `1`, to store the computed factorial value. You can write a **for** loop which iterates from `2` to `n` multiplying the `loop counter` in each iteration with `factorial` and storing the product again in factorial.

Note: Please don't change the package name.

Explorer

Factorial... FactorialC...

SUBMIT

```java
package ql0883;

public class FactorialCalculator {

    public void factorial(int n) {

        int factorial = 1;

        //Fill in the missing code using the for loop

        int i=1;

        while(i<=n){

            factorial*=i;

            i++;

        }



        System.out.println("Factorial of " + n + " = " + factorial);

    }

}
```

18.1.4. Write a program to produce the desired output

00:49

Write a class `PrintAlphabet` with a **main** method. Write a logic to print all the English alphabets from **A** to **Z**.

Hint: Use `for` loop to iterate over the characters from `'A'` to `'Z'`.

You can declare and initialize a loop counter `char i` and initialize it to 'A' (eg: `char i = 'A';`). The `condition` can similarly be `i <= 'Z';`.
And the update statement can be `i++` then print the value of `i`.

Note: Please don't change the package name.

Explorer

PrintAlph...

SUBMIT

```java
package q10884;

class PrintAlphabet

{

    public static void main(String args[])

    {

        char ch;


        for( ch = 'A' ; ch <= 'Z' ; ch++ )

            System.out.println(ch);



    }



}
```

18.1.5. Java Program to check whether the given number is Prime or not

01:54

Write a class `PrimeVerify` with a **public** method `checkPrimeOrNot` that takes one parameter `number` of type `int`. Write a code to check whether the given number is a **prime** number or not.

For example:

```
Cmd Args : 13
13 is a prime number
```

Note: Please don't change the package name.

Explorer

PrimeVeri... PrimeVerif...

SUBMIT

```java
package q10885;
public class PrimeVerify{
    boolean isPrime(int n)
    {
        // Corner cases
        if (n <= 1)
            return false;
        if (n <= 3)
            return true;
        if (n % 2 == 0 || n % 3 == 0)
            return false;
        for (int i = 5; i * i <= n; i = i + 6)
            if (n % i == 0 || n % (i + 2) == 0)
                return false;
        return true;
    }
    public void checkPrimeOrNot(int n){
        if(isPrime(n)){
            System.out.println(n+" is a prime number");
        }else{
            System.out.println(n + " is not a prime number");
        }
    }

}
```

18.1.6. Java Program to find the Factorial of a given number

00:29

Write a class `Factorial` with a **main** method. The method takes one command line argument. Write a logic to find the **factorial** of a given argument and print the output.

For example:

```
Cmd Args : 5
Factorial of 5 is 120
```

Note: Please don't change the package name.

Explorer

Factorial....

SUBMIT

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


        int i,fact=1;


        int number=Integer.parseInt(args[0]);


        for(i=1;i<=number;i++){


            fact=fact*i;


        }


        System.out.println("Factorial of "+number+" is "+fact);

    }

}
```

18.2.1. Understanding while-loop statement

00:24

A `while` statement is used to execute one or more code statements repeatedly as long as a condition is `true`.
Its syntax is

```
while (condition) {
    statement_1;
    statement_2;
    ...
}
```

The `condition` is an expression which should always evaluate to a `boolean` value (either `true` or `false`).

- If it evaluates to `true`, the body containing one or more code statements is executed.
- If the expression evaluates to `false`, the body containing code statements is not executed.

```
For example:
    int value = 4, sum = 0;
    while (value > 0) { // start of while loop body
        sum = sum + value;
        value--;
    } // end of while loop body
```

1. The `condition` in the above example is `value > 0`.
2. The code inside the `while` loop's block between the opening-brace `{` and the closing-brace `}`, will be repeatedly executed until the condition evaluates to `false`.
3. In our case, since in every iteration the `value` is decremented using `value--;`, after four iterations the `value` will be equal to `0` and the condition `value > 0` will evaluate to `false`.
4. A `while` loop's body can contain two branching statements, `break;` and `continue;`, which we will learn later.

Explorer

WhileDe... WhileDem... SUBMIT

```java
package q10888;

public class WhileDemo {

    public void printNumbers(int max) {

        int num = 1;

        // Fill in the missing code using a while loop to print from 1 to max

        while(num<=max) {

            System.out.println(num);

            num++;



        }

    }

}
```

18.2.2. Fill in the missing code

03:15

Write a class `PrintFiveTimes` with a **main** method. The program should print **Ganga** five times.

Fill the missing code in the below program so that it produces the desired output.

Note: Please don't change the package name.

Explorer

PrintFive...

SUBMIT

```
package q10889;

public class PrintFiveTimes {

    public static void main(String[] args) {

        int i = 0;

        while (i <5 ) { // complete the condition here

        System.out.println("Ganga");//write the text to be printed here

        i = i + 1;

    }

}
}
```

18.2.3. Fill in the missing code

00:52

Write a class `PrintThreeTimes` with a **main** method. The program should print **Thames** three times.

Fill the missing code so that it produces the desired output.

Note: Please don't change the package name.

Explorer

PrintThre...

SUBMIT

```
package q10890;

public class PrintThreeTimes {

    public static void main(String[] args) {

        int i = 0;

        while (i<3 ) {

            System.out.println("Thames");

            i ++ ;

        }

    }

}
```

18.2.4. Write a Java program to find whether the given number is Armstrong or not 01:19

Create a class `Armstrong` with a **main** method to check the given number is an `armstrong number` or not.

[Hint: An `armstrong number` is a number that is the sum of its own digits each raised to the power of the number of digits.
For example

```
9 = 9^1 = 9
371 = 3^3 + 7^3 + 1^3 = 27 + 343 +1 = 371
8208 = 8^4 + 2^4+0^4 + 8^4 = 4096 + 16 + 0 + 4096 = 8028
```

If the input is given as command line arguments to the **main()** as **[153]** then the program should print the output as:

```
Cmd Args : 153
The given number 153 is an armstrong number
```

If the input is given as command line arguments to the **main()** as **[25]** then the program should print the output as:

```
Cmd Args : 25
The given number 25 is not an armstrong number
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

Armstron...

SUBMIT

```java
package ql0891;
public class Armstrong
	{static boolean isArmstrong(int n)
		{int temp, digits=0, last=0, sum=0;
		temp=n;while(temp>0)
		{temp = temp/10;
		digits++;
		}
		temp = n;
		while(temp>0)
		{
		last = temp % 10;
		sum += (Math.pow(last, digits));temp = temp/10;
		}
	if(n==sum)
		return true;
		else return false;
	}
		public static void main(String args[])
	{
		int num=Integer.parseInt(args[0]);
		if(isArmstrong(num))
		{System.out.println("The given number " +num+ " is an armstrong number");
		}
		else
		{System.out.println("The given number " + num + " is not an armstrong number");
		}
	}
}
```

18.2.5. Java Program to display Prime numbers between Limits

00:41

Write a class `PrimeNumbers` with a **public** method `primeInLimits` that takes two parameters `high` and `low` are of type `int`. Print the prime numbers between the given limits (including first and last values)

For example:

```
Cmd Args : 3 10
3 5 7
```

Note: Please don't change the package name.

Explorer

PrimeNu... PrimeNu...

SUBMIT

```java
package q10892;
public class PrimeNumbers {
	public void primeInLimits(int start, int end) {
		for(int i = start ; i <= end ; i++)
		{
			int count = 0;
			for(int j = 1 ; j <= i ; j++)
			{
				if(i % j == 0)
	count = count+1;
			}
			if(count == 2)
				System.out.print(i+" ");
		}
	)

}
```

18.2.6. Java Program to find Sum and Reverse of a given number

01:13

Write a class `SumAndReverseNumber` with a method `sumAndReverseANumber` that takes one parameter `number` of type `int` and find the sum of digits of a given number also find the reverse of a number and print the result.

For example, if a value of **369** is passed then the output should be:

```
Sum of digits : 18
Reverse : 963
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

SumAnd... SumAndR...

SUBMIT

```java
package q10893;
class SumAndReverseNumber {
    static int reversDigits(int num)
    {
        int rev = 0, rem;
        do {
            rem = num % 10;
            rev = rev * 10 + rem;
            num = num / 10;        }
        while (num > 0);
        return rev;
    }
    static int sumDigits(int num)
    {
        int sum = 0, rem;
        do {
            rem = num % 10;
            sum = sum + rem;
            num = num / 10;
        }
        while (num > 0);
        return sum;
    }
    public static void sumAndReverseANumber(int num)
    {
        System.out.println("Sum of digits : "+ sumDigits(num));
        System.out.println("Reverse : "+ reversDigits(num));
    }
}
```

18.2.7. Java Program to check whether the given number is Palindrome or not

01:07

Write a class `NumberPalindrome` with a **public** method `isNumberPalindrome` that takes one parameter `number` of type `int`. Write a code to check whether the given number is **palindrome** or not.

For example

```
Cmd Args : 333
333 is a palindrome
```

Note: Please don't change the package name.

Explorer

NumberP... NumberP...

SUBMIT

```java
package q10894;
public class NumberPalindrome {
    public void isNumberPalindrome(int num) {
        int reversedNum = 0, remainder;
        int originalNum = num;

        while (num != 0) {
            remainder = num % 10;
            reversedNum = reversedNum * 10 + remainder;
            num /= 10;
        }
        if (originalNum == reversedNum) {
            System.out.println(originalNum + " is a palindrome");
        }
        else {
            System.out.println(originalNum + " is not a palindrome");
        }
    }
}
```

18.3.1. Understanding do-while loop statement

01:15

A `do-while` loop statement is used to execute one or more code statements once and then repeatedly execute the same code statements as long as the condition is `true`.
Its syntax is similar to the reverse of `while` loop, and note the extra `;` which we have to provide at the end of `while(condition)`

```
do {
    statement1;
    statement2;
    ....
} while (condition);
```

Note: The statements inside the `do-while` block will be **executed once** before the condition in the `while` is evaluated.
The `condition` is an expression which should always evaluate to a `boolean` value.

- If it evaluates to `true`, the body containing one or more code statements is executed.
- If the expression evaluates to `false`, the body containing code statements is not executed.

```
For example:
    int value = 4, sum = 0;
    do { // start of do-while loop body
        sum = sum + value;
        value--;
    } while (value > 0);// end of do-while loop body
```

1. The `condition` in the above example is `value > 0`.
2. The code inside the `do-while` loop's block between the opening-brace `{` and the closing-brace `}`, will be repeatedly executed until the condition evaluates to `false`.
3. In our case, since in every iteration the `value` is decremented using `value--;`, after four iterations the `value` will be equal to `0` and the condition `value > 0` will become `false`.

Explorer

DoWhile... | DoWhileD...

SUBMIT

```
1   package q10895;
2
3   public class DoWhileDemo {
4
5       public void printNumbers(int max) {
6
7           int num = 1;
8
9
10          do {
11
12              System.out.println(num);
13
14              num++;
15
16          }while (num<=max) ;
17
18      }
19
20  }
21
```

18.3.2. Java Program to Display the Fibonacci Series

01:03

Write a class **FibonacciSeries** with a **main** method. The method receives one command line argument. Write a program to display **fibonacci series** i.e. 0 1 1 2 3 5 8 13 21.....

For example:

```
Cmd Args : 80
0 1 1 2 3 5 8 13 21 34 55
```

Note: Please don't change the package name.

Explorer

Fibonacci...

SUBMIT

```java
package q10896;
class FibonacciSeries{
    static void Fibonacci(int N)
    {
        int num1 = 0, num2 = 1;
        int counter = 0;
        while (counter < N) {
            System.out.print(num1 + " ");
            int num3 = num2 + num1;
            num1 = num2;
            num2 = num3;
            counter = counter + 1;
        }
    }public static void main(String[] args){
        int n=Integer.parseInt(args[0]);
        int a=0,b=1,c;
        int k=0;
        do{if(k==0){
System.out.print(a);k=1;
        }else{
System.out.print(" "+a);}
            c=a+b;
            a=b;
            b=c;
        }while(a<=n);
    }}
```

18.4.1. Understanding the usage of break statement

01:47

A `break;` statement is used to transfer control out of the enclosing `switch`, `for`, `while` and `do-while` statements.
A break statement can be written simply as `break;` or with a label as `break targetLabelName;`

```
for (int i = 1; i < 10; i++) {
    if (i % 5 == 0) {
        break; // breaks from for-loop, if i is divisible by 5.
    }
    System.out.println(i);
}
```

Click on LIVE DEMO to understand the working of `break;` statement.

Write a class `BreakDemo` with a **public** method `calculateSum` that takes one parameter `arr` of type `int` and returns the sum of all elements, untill it encounters a **negative number**. The return type of `calculateSum` should be `int`.

During the iteration if the code encounters a **negative** integer, the code should **break** (stop) from the iteration and return the **sum** of integers it encountered till then.

For example:

```
Cmd Args : 1 2 3 -4 5
6
```

Fill the missing code in the below program.

Hint: You can use `for-each` statement to iterate over the elements of the array.

Sample Test Cases +

Explorer

BreakDe... BreakDe...

SUBMIT

```
package q10897;

public class BreakDemo {

    public int calculateSum(int[] arr) {

        int sum =0 ;

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

            if(arr[i]<0){

                break;
            }else{

                sum+=arr[i];
            }
        }
        return sum;

    }

}
```

18.4.2. Understanding the usage of break target statement

05:35

A `break targetLabelName;` statement is called a **break target** statement.

It is used to transfer control out of the enclosing `switch`, `for`, `while` and `do-while` statements to the next statement below the statement labelled `targetLabelName`.

A break target need not always be a `switch`, `while`, `do-while`, or `for` statement, it could also be a simple block of statements enclosed in opening and closing braces `{ }`, as shown in below example:

```
for (int i = 1; i < 10; i++) {
    System.out.println("Step : 1");
    SkipMe : {
        if (i % 5 == 0) {
            break SkipMe; // breaks to the SkipMe : label
        }
        System.out.println(i);
    }
    System.out.println("Step : 2");
}
```

The `SkipMe :` is called a label. A label is used to name a block of code, so that it can be targeted using a `break` and `continue` statements. Generally labels are used to name loop statements in nested loops, so that one can directly transfer control out of the **labelled** loop block. For example:

```
OuterLoop :
        while (condition1) {
            //do something...
            /do something...
```

Sample Test Cases +

Explorer

BreakTar...

SUBMIT

```
package q10898;
public class BreakTargetDemo {

    public static void main(String[] args) {

        OuterLoop:

        for (int i = 20; i < 25; i++) {

System.out.println("i = " + i);

            InnerLoop :
for (int j = 1; j < 10; j++) {
            if(i%11 ==0){
                break OuterLoop;
            }
            if(j%5==0){
                break InnerLoop;
            }

                System.out.println("j = " + j);

            }
        }
    }
}
```

18.4.3. Understanding the usage of continue statement

00:39

While a `break;` transfers control **out (meaning terminates)**, a `continue;` statement transfers control **to (meaning continues with next iteration)** the innermost enclosing `for`, `while` and `do` statements.

Similar to a `break` statement, a `continue` statement can be written simply as `continue;` or with a label as `continue targetLabelName;`

```
for (int i = 1; i < 10; i++) {
    if (i % 2 == 0) {
        continue; // transfers control to for-loop, if i is divisible by 2.
    }
    System.out.println(i);
}
```

The above code skips printing all even numbers.

Click on LIVE DEMO to understand the working of `continue` in action.

In the below program the method `skipEven(int[] args)` receives an array of integers.

Fill in the missing code inside the `skipEven(...)` such that it uses a `for-each` loop to iterate over the array `arr` elements and uses the `continue;` statement to skip even numbers.

Note: Please don't change the package name.

Explorer

Continue... | Continue...

SUBMIT

```
package ql0899;

public class ContinueDemo {

    public void skipEven(int[] arr) {

        // Fill in the missing code

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

            if(arr[i]%2==0){

                continue;

            }



            System.out.println(arr[i]);

        }

    }

}
```

18.4.4. Understanding the usage of continue target statement

01:13

Similar to a `break targetLabelName;` statement, `continue targetLabelName;` statement is called a `continue target` statement.

It is used to transfer control to enclosing `for`, `while` or `do` statements that are tagged by the label `targetLabelName`.

The **targetLabelName** can be used to tag only a `while`, `do`, or a `for` statement.

Unlike in `break;` a **continue's target** cannot be a **labeled block** that is not associated to one of the above mentioned loops.

As mentioned above the target labels for a `continue`, are generally used to name loop statements in nested loops.

This allows one to directly transfer control (using continue) to a labelled loop block. For example in the below code:

```
OuterLoop :
while (condition1) {
    //do something...
    //do something...

    InnerLoop :
    while (condition2) {
        //do something...
        //do something...
        if (condition3) {
            continue OuterLoop;
        }
```

Explorer

Continue...

SUBMIT

```
package q10900;

public class ContinueTargetDemo {

    public static void main(String[] args) {
        OuterLoop:
        for (int i = 20; i < 25; i++) {
System.out.println("i = " + i);
            InnerLoop :
        for (int j = 1; j < 10; j++) {
            if(i%2==0) continue OuterLoop;
            if(j%3==0) continue InnerLoop;

            System.out.println("j = " +
j);
        }

        }

    }

}
```

18.5.1. When to use which loop

00:23

Any of the three loops (for, while and do-while) can be used for achieving the same goal in a program. However, appropriate loop construct should be selected to improve readability and clarity of expression.

Here are some general guidelines on when to use which loop.

The `for-each` loop is used when we want to iterate through each and every element of a collection of elements.

The `for` loop is generally used when a piece of code has to be repeated n number of times i.e. when we know beforehand the number of iterations the loop should run. Unlike `for-each` loop, a `for` loop has the flexibility to iterate over a range of elements or values determined by the loop counter.

The `while` loop is generally used when the loop's terminating condition happens at some yet-to-be determined time i.e. we do not know beforehand the number of iterations the loop should run, or when the termination condition arrives.

The `do while` loop is the same as `while` loop, except that it will always execute the body of the loop once before the condition is evaluated.

Select all the correct statements from below:

- [x] Whatever can be done with a `for` loop can also be done with a `while` loop.
- [x] Whatever can be done with a `while` loop can also be done with a `for` loop.
- [ ] Whatever can be done with a `for` loop can also be done with a `for-each` loop.

19.1.1. Introduction to Arrays

06:47

In a programming language the data that has to be processed is loaded in memory and held in variables. When we want to process an integer value, we declare an `int` variable and assign the value to it.

Suppose we want to write a program that prints the total marks scored by students in a class (say, for a total of 30 students).

One way of doing it is to declare 30 `int` variables, which is unwieldy when we think of multiple classes or a school.

In such cases, when we want to store multiple values of the same data type, we use an array data structure.

A `data structure` is a particular way of organizing data in a computer so that data can be accessed and modified efficiently.

An `array` is a kind of data structure that holds a fixed number of values of a single type, each identified by an `array index`.

For example, an array of size 10 (`int[] marksArr = new int[10];`) can be visualized as shown below.

0	1	2	3	4	5	6	7	8	9

Essentially an array can be thought of as a sequence of buckets. The first bucket is identified with number 0, the second bucket with 1 and so on. This number is called the `index`.

An element is stored in a bucket using the bucket's index. For example, if we want to store a value of 341 at the first index, the code is `marksArr[0] = 341;`.

Similarly, if we want to store a value of 425 in the second bucket, the code will be

Sample Test Cases +

Explorer ArrayDe... Submit Plots

```java
package q10935;
import java.util.*;
public class ArrayDemo {
    public static void main(String[] args) {
        int[] ramanujanNumbers = { 1729, 4104, 13832, 20683, 33832 };
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        //Write your code here
        System.out.println("Printing Ramanujan Numbers: ");
        for(int i=0;i<n;i++){
            System.out.println(ramanujanNumbers[i]);
        }
    }
}
```

Terminal Test cases

Shivanshojha

19.2.1. Understanding Multi-dimensional Arrays

01:35

In Java, a `multidimensional array` is implemented as an array of arrays.

A `multidimensional array` is an array whose components are themselves arrays. Imagine a medical cabinet like this:

The cabinet has multiple rows of medicines. Each row in the cabinet is an `array` and the whole cabinet is a `multidimensional array`.

In Java, the arrays / rows inside a `multidimensional array` can be of different lengths, like below -

0	1	2	3	4	5	6	7	8	9	→	Array at index 0

47	49	42	45	60	66	→	Array at index 1

15	16	17	18	19	20	21	22	23	24	25	26	→	Array at index 2

32	10	22	→	Array at index 3

Each `array` inside of a `multidimensional array` can be accessed with an `index` number as shown in the above figure. The first `array` is identified with number 0, the second `array` with 1 and

- [x] `int[][] intArray = new int[2][5];`

- [x] `int[][] intArray = new int[3][];`
  `int[] arr1 = new int[5];`
  `int[] arr2 = new int[6];`
  `int[] arr3 = new int[8];`

  `intArray[0] = arr1;`
  `intArray[1] = arr2;`
  `intArray[2] = arr3;`

- [x] `int[] arr1 = new int[5];`
  `int[] arr2 = new int[6];`
  `int[] arr3 = new int[8];`

  `int[][] intArray = {arr1, arr2, arr3};`

- [x] `int[] arr1 = new int[5];`

  `int[][] intArray = {arr1, {10,11,12}, {100,150,200}};`

- [x] `int[][] intArray = {{10,11,12}};`

- [ ] `int[][] intArray = {{10.0,11,12}};`

Shivanshojha

19.2.2. Iterating through a multidimensional array

08:51

The class `MultiDimArrayPrinter` with a `main` method. The program prints a multidimensional array of integers.

An integer two-dimensional array **int2DArr** is declared and initialized in the given code. Your task is to :

- Take an integer **n** from the user.
- Use a `for each` loop to iterate over the each element of the **int2DArr** array
- Print the elements of the array which are less than the given integer **n**.

Note:

- The code provided in the editor reads the input integer using the nextInt() method and stores it in **n** variable.
- Please don't change the package name.

Sample Test Cases +

Explorer MultiDim... Submit Plots

```java
package q10946;
import java.util.*;
public class MultiDimArrayPrinter {
	public static void main(String[] args) {
		int[][] int2DArr = {
		{1},
		{2, 3},
		{4, 5, 6},
		{7, 8, 9, 10}
		};
		Scanner sc = new Scanner(System.in);
		int n = sc.nextInt();
		for(int[] i: int2DArr){
		for(int val : i)
		if(val<n)
		System.out.print(val+" ");
		else
		break;
		}}}
```

Terminal Test cases

19.2.3. Iterating through a multidimensional array.

08:53

The class `MultiDimArrayPrinter` prints a multidimensional array of integers.

An integer two-dimensional array **int2DArr** is declared and initialized in the given code. Your task is to :

- Take an integer **n** from the user.
- Use a nested for loop to iterate over the first **n** rows of the **int2DArr** array

Note:

- The code provided in the editor reads the input integer using the nextInt() method and stores it in **n** variable.
- Constraints: n < 4
- Please don't change the package name.

Sample Test Cases +

Explorer

MultiDim...

Submit

```java
package q10947;
import java.util.*;
public class MultiDimArrayPrinter {
    public static void main(String[] args) {
        int[][] int2DArr = {
            {1},
            {2, 3},
            {4, 5, 6},
            {7, 8, 9, 10}
        };
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

    //Write your code here....
        int j=0;
        for(int[] a : int2DArr){
            for(int val : a){
                if(j<n)
                System.out.print(val+" ");
            }
            j+=1;
            System.out.println();
        }}}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

19.2.4. Program to find Addition of Two matrices

22:31

Write a class `AdditionOfMatrix` with a **public** method `add` which returns the sum of its arguments. If the two arguments are not matrices of the same size, then the method should return null.

Consider the following examples for your understanding:

```
Matrix 1:
Enter number of rows: 2
Enter number of columns: 2
Enter 2 numbers separated by space
Enter row 1: 1 2
Enter row 2: 3 4
Matrix 2:
Enter number of rows: 2
Enter number of columns: 2
Enter 2 numbers separated by space
Enter row 1: 4 5
Enter row 2: 6 7
Sum of the two given matrices is:
5 7
9 11
```

```
Matrix 1:
Enter number of rows: 2
Enter number of columns: 2
Enter 2 numbers separated by space
Enter row 1: 1 2
Enter row 2: 3 4
Matrix 2:
Enter number of rows: 2
Enter number of columns: 3
Enter 3 numbers separated by space
```

Sample Test Cases +

Explorer Addition... AdditionOf... Submit

```java
package q11053;
public class AdditionOfMatrix {

    /**
     * Computes the addition of the two given matrices if the sizes of the matrices are the same.
     * Otherwise, returns null.
     *
     * @return the resultant matrix, null if addition is not possible
     */
    public int[][] add(int[][] matrix1, int[][] matrix2) {
        // Write the code
        if(matrix1==null || matrix2==null || matrix1.length!=matrix2.length || matrix1[0].length !=matrix2[0].length){
            return null;
        }int[][] result=new int[matrix1.length][matrix1[0].length];

        for(int i=0;i<matrix1.length;i++){
            for(int j=0;j<matrix1[0].length;j++){
                result[i][j]=matrix1[i][j]+matrix2[i][j];
            }
        }
        return result;

    }
}
```

Terminal Test cases

Prev Reset Submit Next

19.2.5. Write a Java program to display Transpose of the given Matrix

07:54

Write a class `TransposeMatrix` with a **public** method `transposeMatrix` that takes one parameter `matrix1` of type `int[][]` which returns the transpose of the given matrix.

Consider the following example for your undertanding:

```
Matrix:
Enter number of rows: 3
Enter number of columns: 2
Enter 2 numbers separated by space
Enter row 1: 1 2
Enter row 2: 3 4
Enter row 3: 5 6
Transpose of given matrices is:
1 3 5
2 4 6
```

Note: Please don't change the package name.

Sample Test Cases +

Transpos... Transpose... Submit

```java
package q11105;
public class TransposeMatrix {
/**
 * Computes the transpose of the given matrix.
 *
 * @return the resultant matrix
 */

public int[][] transposeMatrix(int[][] matrix) {

// Write the code
if(matrix==null){
return null;
}
int rows=matrix.length;
int cols=matrix[0].length;

int[][] result=new int[cols][rows];

for(int i=0;i<cols;i++){
for(int j=0;j<rows;j++){
result[i][j]=matrix[j][i];
}
}
return result;
}
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

19.2.6. Program to find Multiplication of Two matrices

41:10

Write a class `MultiplicationOfMatrix` with a **public** method `multiplication` which returns the multiplication result of its arguments. if the first argument column size is not equal to the row size of the second argument, then the method should return null.

Consider the following example for your understanding

```
Matrix 1:
Enter number of rows: 3
Enter number of columns: 2
Enter 2 numbers separated by space
Enter row 1: 1 2
Enter row 2: 4 5
Enter row 3: 7 8
Matrix 2:
Enter number of rows: 2
Enter number of columns: 3
Enter 3 numbers separated by space
Enter row 1: 1 2 3
Enter row 2: 4 5 6
Multiplication of the two given matrices is:
9 12 15
24 33 42
39 54 69
```

```
Matrix 1:
Enter number of rows: 2
Enter number of columns: 2
Enter 2 numbers separated by space
Enter row 1: 1 2
Enter row 2: 3 4
Matrix 2:
```

Sample Test Cases +

Explorer | Multiplica... | Multiplicati... | Submit

```java
package q11106;
public class MultiplicationOfMatrix{
	public int[][] multiplication(int[][] matrix1, int[][] matrix2) {
		/*Return the result if the matrix1 coloumn size is equal to matrix2 row size and print the result.
		* @Return null.
		*/
		// Write your logic here for matrix multiplication
		if(matrix1==null || matrix2==null || matrix1[0].length!=matrix2.length){
			return null;
		}
		int rows1=matrix1.length;
		int cols1=matrix1[0].length;
		int cols2=matrix2[0].length;

		int[][] result=new int[rows1][cols2];
		for(int i=0;i<rows1;i++){
			for(int j=0;j<cols2;j++){
				for(int k=0;k<cols1;k++){
					result[i][j]+=matrix1[i][k] * matrix2[k][j];
				}
			}
		}
		return result;
	}
}
```

Terminal | Test cases

Shivanshojha

20.1.1. Print the elements in an array

55:41

In the given code, the class `RiverNamePrinter` prints the elements of the array containing the names of Indian rivers.

In the `main` method use a `for-each` loop to iterate over the array `namesArr` and print each name.

Note: Please don't change the package name.

Sample Test Cases

Explorer

RiverNam...

Submit

```
package q10936;
public class RiverNamePrinter {
    public static void main(String[] args) {
        String[] namesArr = { "Ganga", "Yamuna", "Godavari", "Krishna", "Narmada", "Kaveri" };

        //write the code to print the River names one after one
        for(String names : namesArr){
            System.out.println(names);
        }
    }
}
```

Plots

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

20.1.2. Print the elements in an array

02:36

Create a class `MountainNamePrinter` with a main method. Create an array with the following names: **Nanda Devi**, **Kamet**, **K12**, **Dunagiri**.

Iterate over the array and print each name on a separate line.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

Mountain...

Submit

```
package q10937;
public class MountainNamePrinter{
    public static void main(String args[]){
        String[] arr={"Nanda Devi","Kamet","K12","Dunagiri"};
        for(String val : arr){
            System.out.println(val);
        }
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

20.1.3. Find array element with index

05:38

Create a class `CityNameFinder` with a **public** method `findElement` that takes one parameter `index` of type `int`.

The `findElement` method should create an array `namesArr` containing these city names: **Mumbai, Delhi, Kolkata, Chennai, Hyderabad, Bangalore.**
Write code in the method to print the element that is present at the **index** passed to the method.

The program should print **Wrong Index** if the value of index is not in the range of valid indices of the array.

For example:

```
Cmd Args : 99
Wrong Index
```

Note: Please don't change the package name.

Sample Test Cases +

CityName... CityName...

```java
package q10939;
public class CityNameFinder {
	/**
	 * Create an array which contains given names
	 *
	 *
	 * @print the index element
	 *
	 */
	
	public void findElement(int index) {
		//Write your code here
		String[] arr=
{"Mumbai","Delhi","Kolkata","Chennai","Hyderabad","Bangalore"};
		if(index<arr.length){
			System.out.println(arr[index]);
		}
		else
		System.out.println("Wrong Index");
	}
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

20.1.4. Find the Index of a given array element

26:43

Write a class `IndexFinder` with **public** method `printIndex` that takes two parameters one is `intArr` of type `int[]` and other is an `element` of type `int` and print all indices of the elements in `intArr` which are equal to the given `element`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Cmd Args : 2 2 2 42 2
Indices of the elements matched with the given element 2 is :
0
1
2
```

Note: Please don't change the package name.

Sample Test Cases +

IndexFin... IndexFind... Submit

```java
package q10940;
//write your code here..
public class IndexFinder{
	public void printIndex(int[] intArr,int element){
		int i=0;
		for(int num:intArr){
			if(num==element)
		{
			System.out.println(i);
		}
		i++;
		}
	}
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

20.1.5. Find the Index of a given array element

03:23

Create a class with name `IndexFinder` with `public` method `printIndex` that takes two parameters one is `intArr` of type `int[]` and second one is `element` of type `int` and returns only the first index match of the element in the `intArr`.

Assumptions:

1. `arr` is never null
2. `arr` may contain duplicate elements but returns the index of the first match of the element

Here is an example:

```
Cmd Args : 69 25 89 54 89
First match of the element 89 index is: 2
```

Note: Please don't change the package name.

Sample Test Cases +

IndexFin... IndexFind... Submit

```java
package ql0941;

public class IndexFinder {
	/**
	 * Find the first index match of the element in the array
	 *
	 * @return index
	 */

	public int printIndex(int[] intArr, int element) {

		//Write your code here
		int i=0;
		for(int val : intArr){
			if(val==element)
				break;
			i++;
		}
		return i;
	}
}
```

Terminal Test cases

Shivanshojha

20.1.6. Count the number of times an element occurs in an array

03:41

Write a class `ArrayElementCounter` with a **public** method `countElement` that takes two parameters one is `arr` of type `int[]` and the other is `element` of type `int` and returns the count of the `element` that occures in the `arr`. The return type of `countElement` should be `int`.

Here is an example:

```
Cmd Args : 3 2 3 3 3 3
The element 3 presents 4 times in the arry
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer ArrayEle... ArrayElem... Submit

```java
package q10942;
//Write your code here..
public class ArrayElementCounter{
    public int countElement(int arr[],int element){
        int count=0;
        for(int val: arr){
            if(val==element)
                count++;
        }
        return count;
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

20.1.7. Count the number of times an element occurs in an array

02:21

Create a class `ArrayElementCounter` with a **public** method `countElement` that takes two parameters one is `arr` of type `int[]` and second one is `element` of type `int` and returns the count of `element` occures in the `arr`. The return type of `countElement` should be `int`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Cmd Args : 1 1 2 3 1
2
```

Note: Please don't change the package name.

Sample Test Cases +

ArrayEle... ArrayElem... Submit

```java
package ql0943;

public class ArrayElementCounter {
	/**
	 * Find number of times the element present in the given array
	 *
	 *
	 *
	 * @retrun count
	 */

	public int countElement(int[] arr, int element) {
		//Write your code here
		int count=0;
		for(int num:arr){
			if(num==element)
				count++;
		}
		return count;

	}
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

20.1.8. Reversing an Array

07:49

Create a class `ArrayReverser` with a **public** method `reverseArray` that takes one parameter `arr` of type `int[]` and returns all the elements in the `arr` in reverse order. The return type of `reverseArray` should be `int`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Cmd Args : 1 2 3 4
4
3
2
1
```

Note: Please don't change the package name.

Sample Test Cases +

ArrayRev... ArrayRev... Submit

```
package q10944;

public class ArrayReverser {
    /**
     * Compute the reverse order of the given array
     *
     *
     * @return resultant array
     */
    public int[] reverseArray(int[] arr) {
        //Write your code here
        int i=arr.length-1;
        int arr1[]=new int[i+1];
        for(int val : arr){
            arr1[i]=val;
            i--;
        }
        return arr1;
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

21.1.1. Write a Java program to Sort elements using Bubble Sort

41:35

`Sorting` specifies the way to arrange data in a particular order either in ascending or descending.

`Bubble sort` is an internal sorting technique in which **adjacent elements** are compared and exchanged if necessary.

The working procedure for **bubble sort** is as follows:

1. Let us consider an array of **n** elements (i.e., **a[n]**) to be sorted.
2. Compare the first two elements in the array i.e., **a[0]** and **a[1]**, if **a[1]** is less than **a[0]** then **interchange** the two values.
3. Next compare **a[1]** and **a[2]**, if **a[2]** is less than **a[1]** then interchange the values.
4. Continue this process till the last two elements are **compared** and **interchanged**.
5. Repeat the above steps for **n - 1** passes.

Let us consider an example of array numbers "50 20 40 10 80", and sort the array from **lowest number** to **greatest number** using bubble sort.

In each step, elements written in **bold** are being **compared**. Number of elements in the array are **5**, so 4 passes will be required.

```
Pass - 1 :
( 50 20 40 10 80 ) -> ( 20 50 40 10 80 ) // Compared the first two elements, and swaps since 50 > 2
( 20 50 40 10 80 ) -> ( 20 40 50 10 80 ) // Swap since 50 > 40.
( 20 40 50 10 80 ) -> ( 20 40 10 50 80 ) // Swap since 50 > 10.
( 20 40 10 50 80 ) -> ( 20 40 10 50 80 ) // Since the elements are already in order (50 < 80), al
```

Total number of elements in the given array are **5**, so in **Pass - 1** total numbers compared are **4**. After completion of **Pass - 1** the **largest element** is moved to the **last position** of the array.

Sample Test Cases +

Explorer BubbleSo... BubbleSor... Submit Plots

```java
package q11039;
public class BubbleSorting{
    public void bubbleSort(int[] array){
        int len=array.length;
        for(int i=0;i<len-1;i++){
            for(int j=0;j<len-i-1;j++){
                if(array[j]>array[j+1]){
                    int temp;
                    temp=array[j];
                    array[j]=array[j+1];
                    array[j+1]=temp;
                }
            }
        }
        for(int num:array){
            System.out.println(num);
        }
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

21.1.2. Program to Sort elements using Insertion Sort

07:47

`Insertion sort` is one that **sorts** a set of elements by inserting an element into the existing sorted elements.

The working procedure for **insertion sort** is as follows:

1. Let us consider an array of **n** elements (i.e., **a[n]**) to be sorted.
2. The **first element a[0]** in the array is itself trivially sorted.
3. The **second element a[1]** is compared with first element **a[0]** and it will be inserted either before or after first element, so that **first** and **second** elements are sorted.
4. The **third element a[2]** is compared with **a[0]** and **a[1]** and it will be inserted into its proper place by checking conditions, so that first three elements are sorted.
5. Repeat the same process for **n - 1** passes.

Let us consider an example of array numbers "**50 20 40 10 30**", and sort the array from **lowest number** to **greatest number** using insertion sort.

In each step, elements written in **color** is **compared** with elements written in **bold**. Number of elements in the array are **5**, so **4** passes will be required.

```
Pass - 1 :
( 50 20 40 10 30 ) -> ( 20 50 40 10 30 ) // The second element a[1] is compared with the first ele
```

Now, **Pass - 2** can compare **a[2]** with **a[0]** and **a[1]**.

```
Pass - 2 :
( 20 50 40 10 30 ) -> ( 20 40 50 10 30 ) // Since 40 > 20 and 40 < 50, so 40 is inserted in betwe
```

Now, **Pass - 3** can compare **a[3]** with **a[0]**, **a[1]** and **a[2]**.

Sample Test Cases +

Explorer | Insertion... | InsertionS... | Submit | Plots

```java
package q11040;
public class InsertionSorting{
    public void insertionSort(int arr[]){
        int n=arr.length;
        for(int i=1;i<n;i++){
            int key=arr[i];
            int j=i-1;
            while(j>=0 && arr[j]>key){
                arr[j+1]=arr[j];
                j=j-1;
            }
            arr[j+1]=key;
        }
        for(int i=0;i<n;i++){
            System.out.println(arr[i]);
        }
    }
}
```

Terminal | Test cases

Shivanshojha

21.1.3. Program to Sort elements using Selection Sort - Largest element method

08:03

`Selection sort` process can be done in **two** ways, one is the `largest element method` and the other is `smallest element method`.

The working procedure for selection sort using the **largest element method** is as follows:

1. Let us consider an array of **n** elements (i.e., **a[n]**) to be sorted.
2. In the first step, the **largest element** in the list is searched. Once the largest element is found, it is exchanged with the element which is placed at the **last position**. This completes the first pass.
3. In the next step, it searches for the **second largest element** in the list and it is interchanged with the element placed at **second last position**. This is done in second pass.
4. This process is repeated for **n - 1** passes to sort all the elements.

Let us consider an example of array numbers "**80 10 50 20 40**", and sort the array from **lowest number** to **greatest number** using selection sort by the largest element.

```
Pass - 1 :
( 80 10 50 20 40 ) -> ( 40 10 50 20 80 ) // First finds the largest element and it is exchanged w
```

After completion of **Pass - 1**, the largest element is moved to the end of the array.

Now, **Pass - 2** can find the next largest element with out considering the last position element.

```
Pass - 2 :
( 40 10 50 20 80 ) -> ( 40 10 20 50 80 ) // Largest in 40 10 50 20 is 50 and it is replaced with
```

After completion of **Pass - 2** the **second largest element** is moved to the **second last position** of the array.

Sample Test Cases +

Selection... Selection... Submit

```java
package q11041;
public class SelectionSortingLargestElement{
    public void selectionSortLargestEle(int arr[]){
        int n=arr.length;
        for(int i=n-1;i>0;i--){
            int max_idx=i;
            for(int j=0;j<i;j++){
                if(arr[j]>arr[max_idx]){
                    max_idx=j;
                }
            }
            int temp=arr[max_idx];
            arr[max_idx]=arr[i];
            arr[i]=temp;
        }
        for(int i=0;i<n;i++){
            System.out.println(arr[i]);
        }
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

21.1.4. Program to Sort elements using Selection Sort - Smallest element method

05:10

The working procedure for **selection sort smallest element method** is as follows:

1. Let us consider an array of **n** elements (i.e., **a[n]**) to be sorted.
2. In the first step, the **smallest element** in the list is searched. Once the smallest element is found, it is exchanged with the element which is placed at the **first position**. This completes the first pass.
3. In the next step, it searches for the **second smallest element** in the list and it is interchanged with the element placed at **second position**. This is done in second pass.
4. This process is repeated for **n - 1** passes to sort all the elements.

Let us consider an example of array numbers **"80 10 50 20 40"**, and sort the array from **lowest number** to **greatest number** using selection sort smallest element method.

```
Pass - 1 :
( 80 10 50 20 40 ) -> ( 10 80 50 20 40 ) // First finds the smallest element and it is exchanged w
```

After completion of **Pass - 1**, the smallest element is moved to the starting position of the array.

Now, **Pass - 2** can find the next smallest element with out considering the first position element.

```
Pass - 2 :
( 10 80 50 20 40 ) -> ( 10 20 50 80 40 ) // Smallest in 80 50 20 40 is 20 and it is replaced wit
```

After completion of **Pass - 2** the **second smallest element** is moved to the **second position** of the array.

Now, **Pass - 3** can find the next smallest element with out considering the **first two position elements** because they are already sorted.

Sample Test Cases +

Explorer Selection... Selection... Submit

```java
package q11042;
public class SelectionSortingSmallestElement{
    public void selectionSortSmallestEle(int arr[]){
        int n=arr.length;
        for(int i=0;i<n-1;i++){
            int min_idx=i;
            for(int j=i+1;j<n;j++){
                if(arr[j]<arr[min_idx])
                    min_idx=j;
            }
            int temp=arr[min_idx];
            arr[min_idx]=arr[i];
            arr[i]=temp;
        }
        for(int i=0;i<n;i++){
            System.out.println(arr[i]);
        }
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

Home Learn Anywhere 12216507.st@lpu.in Support Logout

21.1.5. Write a Java program to Sort elements using Merge Sort

18:24

Write code to **sort** the array elements by using **merge sort** technique.

Write a class `MyMergeSort` with **main** method.

Note: Please don't change the package name.

Sample Test Cases +

Question Hints +

MyMerge... Submit

```
private void doMergeSort(int lowerIndex, int higherIndex) {
if(lowerIndex<higherIndex){
int middle=(lowerIndex+higherIndex)/2;
doMergeSort(lowerIndex,middle);
doMergeSort(middle+1,higherIndex);
mergeParts(lowerIndex,middle,higherIndex);
}
}
private void mergeParts(int lowerIndex, int middle, int higherIndex) {
int tempArray[]=new int[higherIndex-lowerIndex+1];
int leftIndex=lowerIndex;
int rightIndex=middle+1;
int tempIndex=0;
while(leftIndex<=middle && rightIndex<=higherIndex){
if(array[leftIndex]<=array[rightIndex]){
tempArray[tempIndex]=array[leftIndex];
leftIndex++;
}
else{
tempArray[tempIndex]=array[rightIndex];
rightIndex++;
}
tempIndex++;
}
while(leftIndex<=middle){
tempArray[tempIndex]=array[leftIndex];
leftIndex++;
tempIndex++;
}
while(rightIndex<=higherIndex){
```

Terminal Test cases

Prev Reset Submit Next

21.1.5. Write a Java program to Sort elements using Merge Sort

18:24

Write code to **sort** the array elements by using **merge sort** technique.

Write a class `MyMergeSort` with **main** method.

Note: Please don't change the package name.

Sample Test Cases +

Question Hints +

MyMerge...

Submit

```
    higherIndex) {
    int tempArray[]=new int[higherIndex-lowerIndex+1];
    int leftIndex=lowerIndex;
    int rightIndex=middle+1;
    int tempIndex=0;
    while(leftIndex<=middle && rightIndex<=higherIndex){
    if(array[leftIndex]<=array[rightIndex]){
    tempArray[tempIndex]=array[leftIndex];
    leftIndex++;
    }
    else{
    tempArray[tempIndex]=array[rightIndex];
    rightIndex++;
    }
    tempIndex++;
    }
    while(leftIndex<=middle){
    tempArray[tempIndex]=array[leftIndex];
    leftIndex++;
    tempIndex++;
    }
    while(rightIndex<=higherIndex){
    tempArray[tempIndex]=array[rightIndex];
    rightIndex++;
    tempIndex++;
    }
    for(int i=0;i<tempArray.length;i++){
    array[lowerIndex+i]=tempArray[i];
    }
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

21.2.1. Program to Search an element using Linear Search

10:51

`Searching` specifies the way to **search** an element from the list of elements.

`Linear search` (or) `Sequential search` is to scan each entry in the list in a **sequential** manner until the desired element is found. i.e., it means to find a particular **key element** in a list of elements in a sequential manner.

`Linear search` is a searching technique in which it **sequentially** checks each element of the list for the **target value** until a match is found (or) until all the elements have been searched.

The working procedure for **linear search** is as follows:

1. Let us consider an array of **n** elements and a **key element** which is going to be search in the list of elements.
2. Compare the **key element** with the first element **a[0]**, if it is **matched** then stop the process and print the **index** of the key element where it is found, otherwise **repeat** the same process with **a[1]**.
3. Compare the **key element** with the second element **a[1]**, if it is **matched** then stop the process and print the **index** of the key element where it is found, otherwise **repeat** the same process with **a[2]**.
4. Continue this process until a match is found (or) until all the elements have been searched.

Let us consider an example of array numbers **"50 20 40 10 80"**, and the key element is to find is 10.

```
Search - 1 :
Compare 10 with value of a[0] i.e., 50, both are not equal so repeat the same process with a[1].
```

```
Search - 2 :
```

Sample Test Cases +

LinearSe... LinearSea...

```java
package q11044;
public class LinearSearch{
	public void linearSearch(int[] array, int key){
		boolean res=true;
		for(int i =0;i<array.length;i++){
			res=true;
			if(array[i]==key){
				System.out.println("Search element "+key+" is found at position : "+i);
				break;
			}
			else
			res=false;
		}
		if(res==false)
		System.out.println("Search element "+key+" is not found");
	}
}
```

Terminal Test cases

21.2.2. Write a Java program to Search an element using Binary Search

23:17

`Binary search` is **faster** than **linear search**, as it uses **divide and conquer** technique and it works on the sorted list either in ascending or descending order.

`Binary search` (or) `Half-interval search` (or) `Logarithmic search` is a search algorithm that finds the position of a **key element** within a sorted array.

`Binary search` compares the **key element** to the **middle element** of the array; if they are **unequal**, the half in which the **key element** cannot lie is eliminated and the search continues on the remaining half until it is successful.

The working procedure for **binary search** is as follows:

1. Let us consider an array of **n** elements and a **key element** which is going to be search in the list of elements.
2. The main principle of binary search has first divided the list of elements into two halves.
3. Compare the **key element** with the **middle element**.
4. If the comparison result is **true** the print the **index position** where the **key element** has found and stop the process.
5. If the **key element** is greater than the **middle element** then search the key element in the second half.
6. If the **key element** is less than the **middle element** then search the key element in the first half.
7. Repeat the same process for the sub lists depending upon whether **key** is in the **first half** or **second half** of the list until a match is found (or) until all the elements in that half have been searched.

Let us consider an example of array numbers **"50 20 40 10 80"**, and the key element is to find is **10**.

Sample Test Cases +

BinarySe... BinarySea... Submit

```java
package q11045;
public class BinarySearch{
	public void binarySearch(int[] array, int key){
		int low=0;
		int high=array.length - 1;
		int res=0;
		boolean res1=false;
		while(low<=high){
			int mid=(low+high)/2;
			if(array[mid]==key){
				res=mid;
				res1=true;
				break;
			}
			else if(array[mid]<key)
				low=mid+1;
			else
				high=mid-1;
		}
		if(res1==true)
			System.out.println("Search element "+key+" is found at position : "+res);
		else
			System.out.println("Search element "+key+" is not found");
	}
}
```

Terminal Test cases

< Prev Reset Submit Next >

22.1.1. Problem solving with Array

10:23

Write a class `SumOfElements` with a **public** method `sum` that takes one parameter `arr` of type `int[]` and returns the sum of all elements in `arr`. The return type of `sum` should be `long`.

Assumptions:

1. `arr` is never null
2. return **0** in case `arr` is empty

Here is an example:

```
Cmd Args : 3 5 3 2 0
Sum of all elements in the given array is : 13
```

Note how the return type of the function is `long` and not `int`. The reason for this is to overcome errors due to data overflow while adding multiple `int` values.

Note: Please don't change the package name.

Sample Test Cases +

SumOfEl... SumOfEle...

Submit

```java
package ql1046;
public class SumOfElements {

    /**
     * Computes the sum of all the elements in the given array
     *
     *
     * @return the sum
     */
    public long sum(int[] arr) {
        // Write the code
        long sum1=0;
        for(int i=0;i<arr.length;i++){
            sum1+=arr[i];
        }
        return sum1;
    }
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

22.1.2. Problem solving with Array

04:19

Write a class `ReversePrint` with a **public** method `reversePrint` that takes one parameter `arr` of type `int[]` and returns the elements of `arr` in reverse order. The return type of `ReversePrint` should be `int`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Cmd Args : 32 56 85 1
Array in reverse order is :
1
85
56
32
```

Note: Please don't change the package name.

Sample Test Cases +

ReverseP... ReverseP... Submit

```
package q11047;
public class ReversePrint {
    /**
     * write a logic to find the array of elements in reverse order.
     *
     *
     * @return the reverseArray
     */

    public int[] reversePrint(int[] arr) {
        //Wrtie code here
        int[] arr1=new int[arr.length];
        int len=arr.length;
        for(int i=0;i<arr.length;i++){
            arr1[i]=arr[len-1];
            len-=1;
        }
        return arr1;
    }
}
```

Terminal Test cases

Shivanshojha

22.1.3. Problem solving with Array

03:31

Write a class `ElementCheck` with a **public** method `checkFirstOrLast` that takes two parameters one is `arr` of type `int[]` and second one is `arg` of type `int` and returns true if the `arg` is first or last element in the `arr` else returns false. The return type of `checkFirstOrLast` is `boolean`.

Assumption:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the array: 4
Enter elements in the array seperated by space:1 2 3 6
Enter the search element:6
true
```

Note: Please don't change the package name.

Sample Test Cases +

ElementC... ElementC...

Submit

```java
package q11048;

public class ElementCheck {

/** write a logic to check whether the given element is present in the first or last in the array
 *
 *
 *
 *@return true if the element present if not return false
 *
 */
	public boolean checkFirstOrLast(int[] arr, int arg) {
		//Write your code
		boolean res=true;
		if(arr[0]==arg || arr[arr.length-1]==arg)
			res=true;
		else
			res=false;
		return res;
	}
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

Home Learn Anywhere

12216507.st@lpu.in Support Logout

22.1.4. Problem solving with Array

06:29

write a class `ElementCount` with a **main** method which passes an `arr` of type `int[]` and an `element` of type `int`. Print number of times the `element` is present in the `arr`.

Here is an example:

```
int[] arr = {1, 12, 9, 3, 5, 3, 78, 4, 3, 9, 18, 56, 1, 5}
Cmd Args : 1
2
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

ElementC...

Submit

```
package q11049;
public class ElementCount {
    public static void main(String[] args) {
        int[] arr = {1, 12, 9, 3, 5, 3, 78, 4, 3, 9, 18, 56, 1, 5};
        int element = Integer.parseInt(args[0]);
        int count=0;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==element)
            count++;
        }
        System.out.println(count);
    }
}
```

Terminal Test cases

Plots

Shivanshojha

22.1.5. Problem solving with Array

02:20

Write a class `ElementCheck` with a **public** method `checkFirstOrLast` that takes two parameters `arr1` and `arr2` are of type `int[]` and returns true if first or last elements of `arr1` and `arr2` are same. The return type of `checkFirstOrLast` should be `boolean`.

Assumptions:

1. `arr1` and `arr2` will never null
2. `arr1` and `arr2` lengths are not equal

Here is an example:

```
Enter no of elements in the array1:
5
Enter elements in the array1 seperated by space:
3 6 8 7 4
Enter no of elements in the array2:
3
Enter elements in the array2 seperated by space:
6 5 4
true
```

Note: Please don't change the package name.

Sample Test Cases +

ElementC... ElementC... Submit

```java
package q11050;
public class ElementCheck {
	/**
	 * check if first or last elements of the arr1 and arr2 are same or not
	 *
	 * @return result
	 */

	public boolean checkFirstOrLast(int[] arr1, int[] arr2) {
		//Write your code here
		boolean res=true;
		if((arr1[0]==arr2[0] || arr1[arr1.length-1]==arr2[arr2.length-1]))
			res=true;
		else
			res=false;
		return res;
	}
}
```

Terminal Test cases

Prev Reset Submit Next

22.1.6. Problem solving with Array

06:12

Write a class `ElementCheck` with a **public** method `elementFinder` that takes two parameters one is `arr` of type `int[]` second one is `element` of type `int` that returns true if the `element` present in the `arr` only one time.

Assumptions:

1. `arr` is never null

These are examples for your understanding:

```
Enter no of elements in the array:
5
Enter elements in the array seperated by space:
9 5 12 35 6
Enter the search element:
5
true
```

```
Enter no of elements in the array:
4
Enter elements in the array seperated by space:
1 2 2 3
Enter the search element:
2
false
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer ElementC... ElementC... Submit Plots

```java
package q11054;

public class ElementCheck {
    /**
     * Compute if the given elemetn is present in the array only one time
     *
     * @return true if it is present else return false
     */

    public boolean elementFinder(int[] arr, int element) {

        //Write your code here
        int count=0;
        boolean res=true;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==element)
                count++;
        }
        if(count==1)
            res=true;
        else
            res=false;
        return res;
    }
}
```

Terminal Test cases

Shivanshojha

22.1.7. Problem solving with Array

02:44

Write a class `ElementCheck` with a **public** method `elementFinder` that takes one parameter `arr` of type `int[]` and return true if the first and last elements of the `arr` are same else return false. The return type of `elementFinder` should be boolean.

Assumptions:

1. `arr` is never null

Here are examples for your understanding:

```
Cmd Args : 33 25 12 5 33
true
```

```
Cmd Args : 1 2 3 4
false
```

Note: Please don't change the package name.

Sample Test Cases +

ElementC... ElementC...

Submit

```java
package ql1055;

public class ElementCheck {

/**
 * Find first and last elements of the array are same are not
 *
 * @return true if both are same else return false
 */

    public boolean elementFinder(int[] arr) {
        //Write your code here
        boolean res=true;
        if(arr[0]==arr[arr.length-1])
            res=true;
        else
            res=false;
        return res;
    }
}
```

Terminal Test cases

< Prev Reset Submit Next

Shivanshojha

22.1.8. Problem solving with Array

07:20

Write a class `ElementCheck` with a **public** method `elementFinder` that takes one parameter `arr` of type `int[]` and returns true if the first four elements in the `arr` contains number 4 else returns false.

Assumptions:

1. `arr` is never null
2. Length of `arr` may be less than four

These are examples for understanding:

```
Cmd Args : 36 51 42 4
true
```

```
Cmd Args : 1 2
false
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer ElementC... ElementC... Submit Plots

```java
/**
 * Find if the first four elements in the array contains number 4 or not
 *
 * @return result
 */

public boolean elementFinder(int[] arr) {

//Write your code here
boolean res=false;
if(arr.length>=4){
for(int i=0;i<4;i++){
if(arr[i]==4){
res=true;
break;
}
}
}
else{
for(int i=0;i<arr.length;i++){
if(arr[i]==4){
res=true;
break;
}
}
}
return res;
}
}
```

Terminal Test cases

Shivanshojha

22.1.9. Problem solving with Array

01:23

Write a class `SumOfElements` with a **public** method `sum` that takes one parameter `arr` of type `int[]` and returns sum of all positive elements in the `arr`. The return type of `sum` should be `int`.

Assumptions:

1. `arr` is never null
2. `arr` may contain -ve numbers

These are examples:

```
Cmd Args : -35 -52 -12 -99
Sum of all positive elements in the array is : 0
```

```
Cmd Args : 36 12 -11 10
Sum of all positive elements in the array is : 58
```

Note: Please don't change the package name.

Sample Test Cases +

SumOfEl... SumOfEle... Submit

```java
package q11057;

public class SumOfElements {
	/**
	 * Compute sum of all +ve elements in the array excluding -ve numbers
	 *
	 * @return sum
	 */

		public int sum(int[] arr) {

		//Write your code here
		int sum=0;
		for(int i=0;i<arr.length;i++){
			if(arr[i]>0)
			sum+=arr[i];
		}
	return sum;
	}
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

22.1.10. Problem solving with Array

03:18

Write a class `SwapFirstAndLast` with a **public** method `swap` that takes one parameter `arr` of type `int[]`. Write a code to swap the first and last elements of the array and print all the elements of the array.

For example:

```
Cmd Args : 1 5 6 7 8
8
5
6
7
1
```

Note: Please don't change the package name.

Sample Test Cases +

SwapFirs... SwapFirst... Submit

```java
package q11058;
public class SwapFirstAndLast{
	public void swap(int[] arr){
		int temp=arr[0];
		arr[0]=arr[arr.length-1];
		arr[arr.length-1]=temp;
		for(int i=0;i<arr.length;i++){
			System.out.println(arr[i]);
		}
	}
}
```

Terminal Test cases

< Prev Reset Submit Next

Shivanshojha

22.1.11. Problem solving with Array

02:53

Write a class `SequenceCheck` with a **public** method `sequenceCheck` that takes one parameter `arr` of type `int[]` and returns true if 6, 9, 12 present consecutively in the `arr`. The return type of `sequenceCheck` should be `boolean`.

Assumptions:

1. `arr` is never null
2. Elements 6, 9, 12 are appear consecutiviely

Here are examples:

```
Cmd Args : 62 32 6 9 12
true
```

```
Cmd Args : 99 36 6 12 56 9
false
```

Note: Please don't change the package name.

Sample Test Cases +

Sequenc... Sequence... Submit

```java
package q11059;

public class SequenceCheck {
    /**
     * Find the given elements present consecutively in the array or not
     *
     * @return result
     */

    public boolean sequenceCheck(int[] arr) {

        //Write your code here
        boolean res=false;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==6){
                if(arr[i+1]==9){
                    if(arr[i+2]==12){
                        res=true;
                    }
                }
            }
        }
        return res;
    }
}
```

Terminal Test cases

< Prev Reset Submit Next

22.1.12. Problem solving with Array

06:08

Write a class `SequenceCheck` with a **public** method `sequenceCheck` that takes one parameter `arr` of type `int[]` and returns true if the elements 1,2,3 are present in the `arr`.The returen type of `sequenceCheck` should be `boolean`.

Assumptions:

1. `arr` is never null
2. The elements need not be in consecutive order

Here are examples:

```
Cmd Args : 1 6 3 2
true
```

```
Cmd Args : 3 6 4 7 8
false
```

Sample Test Cases +

Sequenc... Sequence... Submit

```
public class SequenceCheck {
    /**
     * check if the given array contains the elements 1,2,3
     *
     *
     *
     * @return true if contain else return false
     */


    public boolean sequenceCheck(int[] arr) {
        //Write your code here
        boolean res=false,res1=false,res2=false;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==1)
                res=true;
            else if(arr[i]==2)
                res1=true;
            else if(arr[i]==3)
                res2=true;
        }
        if(res && res1 && res2)
            return true;
        else
            return false;
    }
}
```

Terminal Test cases

Shivanshojha

22.1.13. Problem solving with Array

02:46

Write a class `FindMiddle` with a **public** method `findMiddle` that takes one parameter `arr` of type `int[]` and print the middle element in the `arr`

Assumptions:

1. `arr` is never null
2. `arr` length is even it should print the middle two numbers
3. `arr` length is odd it prints the middle element

Here are examples for your understanding:

```
Cmd Args : 1 6 3 5 4
3
```

```
Cmd Args : 3 2 1 6 5 4
1
6
```

Note: Please don't change the package name.

Sample Test Cases +

FindMidd... FindMiddl...

Submit

```java
package q11061;

public class FindMiddle {
    /**
     * Find the middle element in the given array
     *
     *
     *
     * @ return element
     */
    public void findMiddle(int[] arr) {

        //Write your code here
        int len=arr.length;
        if(len%2==0) {
            System.out.println(arr[(len/2)-1]);
            System.out.println(arr[len/2]);
        }
        else
            System.out.println(arr[len/2]);
    }
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

22.1.14. Problem solving with Array

06:15

Write a class `SequenceCount` with a **public** method `sequenceCount` that takes one parameter `arr` of type `int[]` and returns the sequence count 1,1 in the `arr`. The return type of `sequenceCount` should be `int`.

Assumptions:

1. `arr` is never null
2. Overlapping of counting is allowed

Here is an example:

```
Enter no of elements in the array:
7
Enter elements in the array seperated by space:
1 -1 1 1 1 2 3 1
2
```

Note: Please don't change the package name.

Sample Test Cases +

Sequenc... Sequence... Submit

```java
package q11062;

public class SequenceCount {
    /**
     * Find the sequence count 1,1 int given array
     *
     * @return count
     */

    public int sequenceCount(int[] arr) {

        //Write your code here
        int count=0;
        for(int i=0;i<arr.length-1;i++){
            if( arr[i]==1 && arr[i+1]==1)
                count++;
        }
        return count;
    }
}
```

Terminal Test cases

Shivanshojha

22.1.15. Problem solving with Array

02:56

Write a class `InitializeArray` with a **public** method `initialize` that takes two parameters `len` and `ele` are of type `int` returns an array of length `len` and set all the elements in the array to `ele`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter length of array:
4
Enter element in the array:
3
The output array is:
3
3
3
3
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

InitializeA... InitializeAr...

Submit

```
package q11063;

public class InitializeArray {
    /**
     * Set all the elemets in the array to given element and set to length of array to given length
     *
     *
     *
     * @return arry
     */

    public int[] initialize(int len, int ele) {
        //Write your code here
        int[] arr = new int[len];
        for(int i=0;i<len;i++){
            arr[i]=ele;
        }
        return arr;
    }
}
```

Plots

Terminal Test cases

Shivanshojha

22.1.16. Problem solving with Array

15:11

Write a class `SequenceCount` with a **public** method `sequenceCount` that takes one parameter `arr` of type `int[]` and returns the count of sequences present in the `arr`. The return type of `sequenceCount` should be `int`.

Assumptions:

1. `arr` is never null
2. `arr` may contain zero or more sequences

A sequence is defined as a combination of three numbers in continuous sequence which are of values: 1x, 2x, 3x.

Here are some examples for your understanding:

```
Cmd Args : 9 7 3 6 9
1
```

```
Cmd Args : 2 6 8 2 1
0
```

```
Cmd Args : 10 20 30 1 2 3
2
```

Note: Please don't change the package name.

Sample Test Cases +

Sequenc... Sequence... Submit

```java
package q11064;

public class SequenceCount {
    /**
     * Find the count of sequences present in the given arry.
     *
     *
     *
     * @return count
     */

    public int sequenceCount(int[] arr) {

        //Write your code here
        int count=0;
        for(int i=0;i<arr.length-2;i++){
            if((arr[i+1]==(arr[i]*2) )&& (arr[i+2]==(arr[i]*3)))
                count++;
        }
        return count;
    }
}
```

Terminal Test cases

< Prev Reset Submit Next

Shivanshojha

22.1.17. Problem solving with Array

04:28

Write a class `SumOfArrays` with a **public** method `sum` that takes two parameters `arr1` and `arr2` are of type `int[]` and returns the sum of `arr1` and `arr2` elements in to the third array.

Assumptions:

1. `arr1` and `arr2` will never null
2. `arr1` and `arr2` are of same length

Here is an example:

```
Enter no of elements in the arr1:
3
Enter elements in the arr1 seperated by space:
1 2 3
Enter no of elements in the arr2:
3
Enter elements in the arr2 seperated by space:
4 5 6
5
7
9
```

Note: Please don't change the package name.

Sample Test Cases +

SumOfAr... SumOfArr... Submit

```java
package q11065;
public class SumOfArrays {
    /**
     * Find sum of two array elements and store the elements in the third array
     *
     *
     *
     *
     * @return resultant arry
     */

    public int[] sum(int[] arr1, int[] arr2) {
        //Write your code here
        int[] sum1=new int[arr1.length];
        for(int i=0;i<arr1.length;i++){
            sum1[i]=arr1[i]+arr2[i];
        }
        return sum1;
    }
}
```

Terminal Test cases

Shivanshojha

Home Learn Anywhere 12216507.st@lpu.in Support Logout

22.1.18. Problem solving with Array

00:47

Write a class `CompareArrays` with a **public** method `compareArrays` that takes two parameters `arr1` and `arr2` are of type `int[]` and returns true if the lengths of `arr1` and `arr2` are equal.

Here is an example:

```
Enter lenght of the arr1:
5
Enter lenght of the arr2:
5
true
```

Note: Please don't change the package name.

Sample Test Cases +

Compare... Compare... Submit

```java
package ql1066;

public class CompareArrays {
    /**
     * Find the length of the two arrays are equal or not
     *
     * @return result
     */

    public boolean compareArrays(int[] arr1, int[] arr2) {
        //Write your code here
        boolean res=false;
        if(arr1.length==arr2.length)
            res=true;
        return res;
    }
}
```

Terminal Test cases

Shivanshojha

22.1.19. Problem solving with Array

04:38

Write a class `CompareArrays` with a **public** method `compareArrays` that takes two parameters `arr1` and `arr2` are of type `int[]` and returns true if `arr1` and `arr2` are of equal length and also have same elements. else returns false

Here are examples for your understanding:

```
Enter no of elements in the arr1:
3
Enter elements in the arr1 seperated by space:
8 9 7
Enter no of elements in the arr2:
3
Enter elements in the arr2 seperated by space:
8 9 7
true
```

```
Enter no of elements in the arr1:
3
Enter elements in the arr1 seperated by space:
3 6 7
Enter no of elements in the arr2:
4
Enter elements in the arr2 seperated by space:
3 6 7 1
false
```

Hint: Iterate through the first array and compare each element with the corresponding element in the second array.

Note: Please don't change the package name.

Sample Test Cases +

Compare... Compare... Submit

```java
package q11067;

public class CompareArrays {
    /** Compare lengths and elements of the arr1 and arr2 are equal or not
     *
     *
     *
     * @return result
     */

    public boolean compareArrays(int[] arr1, int[] arr2) {

        //Write your code here
        boolean res=false;
        if(arr1.length==arr2.length){
            res=true;
            for(int i=0;i<arr1.length;i++){
                if(arr1[i]==arr2[i])
                    res=true;
                else
                    res=false;
            }
        }
        return res;
    }
}
```

Terminal Test cases

Shivanshojha

22.1.20. Problem solving with Array

03:41

Write a class `ElementCheck` with a **public** method `checkElement` that takes three parameters one is `arr` of type `int[]` other are `arg1` and `arg2` are of type `int` and returns true if `arr` contains either `arg1` or `arg2` elements only.

These are examples for your understanding:

```
Enter no of elements in the array:
4
Enter elements in the array separated by space:
22 33 22 33
Enter arg1 element:
22
Enter arg2 element:
33
true
```

```
Enter no of elements in the array:
5
Enter elements in the array separated by space:
11 22 11 22 11
Enter arg1 element:
11
Enter arg2 element:
12
false
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer ElementC... ElementC... Submit Plots

```java
package q11068;
public class ElementCheck {
    /**
     * Find if the arr contains only arg1 or arg2 elements
     *
     *
     * @return result
     */

    public boolean checkElement(int[] arr, int arg1, int arg2) {

        //Write your code here
        boolean res=false;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==arg1 || arr[i]==arg2)
                res=true;
            else{
                res=false;
                break;
            }
        }
        return res;
    }
}
```

Terminal Test cases

Shivanshojha

22.1.21. Problem solving with Array

01:23

Write a class `CountEvens` with a **public** method `countEvens` that takes one parameter `arr` of type `int[]` and returns the count of even numbers present in the `arr`.

Assumptions:

1. `arr` is never null

These are examples:

```
Enter no of elements in the array:
5
Enter elements in the array seperated by space:
1 2 3 4 6
3
```

```
Enter no of elements in the array:
5
Enter elements in the array seperated by space:
1 3 7 9 5
0
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer | CountEve... | CountEve... | Submit | Plots

```java
package q11069;


public class CountEvens {
	/**
	 * Find the count of even numbers in the given array
	 *
	 *
	 *
	 * @return count
	 *
	 */

	public int countEvens(int[] arr) {
		
		//Write your code here
		int count=0;
		for(int i:arr){
			if(i%2==0)
				count++;
		}
		return count;
	}
}
```

Terminal | Test cases

22.1.22. Problem solving with Array

06:12

Write a class `ConcatenateArrays` with a **public** method `concatenate` that takes two parameters `arr1` and `arr2` are of type `int[]`. Create a new array that has the elements of both `arr1` and `arr2` in the same order and print the elements.

Example:

```
Enter no of elements in the arr1:
3
Enter elements in the arr1 separated by space:
1 2 3
Enter no of elements in the arr2:
3
Enter elements in the arr2 separated by space:
4 5 6
The resultant array is :
1
2
3
4
5
6
```

Note: Please don't change the package name.

Sample Test Cases +

Concaten... Concaten... Submit

```java
package q11070;
public class ConcatenateArrays{
	public void concatenate(int[] arr1, int[] arr2){
		int len1=arr1.length;
		int len2=arr2.length;
		int[] arr3=new int[len1+len2];
		int k=0;
		for(int i=0;i<len1;i++){
			arr3[i]=arr1[i];
			k++;
		}
		for(int i=0;i<len2;i++){
			arr3[k]=arr2[i];
			k++;
		}
		for(int i=0;i<len1+len2;i++){
			System.out.println(arr3[i]);
		}
	}
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

22.1.23. Problem solving with Array

04:12

Write a class `ElementDiff` with a **public** method `findDiff` that takes one parameter `arr` of type `int[]` and returns the difference between largest and smallest elements in the `arr`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the arr:
5
Enter elements in the arr seperated by space:
33 78 95 14 45
Difference between largest and smallest elements in the array is: 81
```

Note: Please don't change the package name.

Sample Test Cases +

ElementD... ElementDi... Submit

```java
package q11071;

public class ElementDiff {
	/**
	 * Compute the difference between large and small elements in the given arry
	 *
	 *
	 * @return result
	 */

	public int findDiff(int[] arr) {

		//Write your code here
		int min=arr[0],max=arr[0];
		for(int i=1;i<arr.length;i++){
			if(arr[i]<min)
				min=arr[i];
			if(arr[i]>max)
				max=arr[i];
		}
		int diff=max-min;
		return diff;
	}
}
```

Terminal Test cases

Prev Reset Submit Next

22.1.24. Problem solving with Array

01:34

Write a class `FindAverage` with a **public** method `findAverage` that takes one paremeter `arr` of type `int[]` and returns the average of elements in the `arr`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the arr:
4
Enter elements in the arr seperated by space:
1 2 3 4
Avg elements of the array is:
2
```

Note: Please don't change the package name.

Sample Test Cases +

FindAver... FindAvera... Submit

```java
package q11072;

public class FindAverage {
    /**
     * Compute the average of elements in the given array
     *
     *
     * @return result
     */

    public int findAverage(int[] arr){

        //Write your code here
        int sum=0;
        for(int i=0;i<arr.length;i++){
            sum+=arr[i];
        }
        int avg=sum/(arr.length);
        return avg;
    }
}
```

Terminal Test cases

Shivanshojha

22.1.25. Problem solving with Array

03:40

Write a class `FindDuplicate` with a **public** method `findDuplicate` that takes two parameters one is `arr` of type `int[]` and second one is `arg` of type `int` and returns true if `arg` present more than once in the `arr`. The return type of `findDuplicate` should be `boolean`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the array:
6
Enter elements in the array seperated by space:
999 77 77 88 54 -8
Enter the element you want to search:
77
true
```

Note: Please don't change the package name.5

Sample Test Cases +

FindDupli... FindDupli... Submit

```java
package q11073;

public class FindDuplicate {
    /**
     * Find the arg element occures in the arr more than once
     *
     *
     * @return result
     */

    public boolean findDuplicate(int[] arr, int arg) {
        //Write your code here
        int count=0;
        boolean res=false;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==arg)
                count++;
        }
        if(count!=1)
            res=true;
        else
            res=false;
        return res;
    }
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

22.1.26. Problem solving with Array

02:47

Write a class `SumOfSameNumber` with a **public** method `findSumOf` that takes two parameters one is `arr` of type `int[]` and second one is `arg` of type `int` and returns true if sum of all `arg` elements present in the `arr` is greater than or equal to 10.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the array:
6
Enter elements in the array seperated by space:
1 3 4 4 4 5
Enter the search element:
4
true
```

Note: Please don't change the package name.

Sample Test Cases +

SumOfSa... SumOfSa... Submit

```java
package q11074;

public class SumOfSameNumber {
    /**
     * Compute the sum of all arg elments in the arr is greater than 10 or not
     *
     *
     *
     * @return result
     */

    public boolean findSumOf(int[] arr, int arg) {
        //Write your code here
        int sum=0;
        boolean res=false;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==arg)
                sum+=arr[i];
        }
        if(sum>=10)
            res=true;
        return res;
    }
}
```

Terminal Test cases

Prev Reset Submit Next

22.1.27. Problem solving with Array

02:56

Write a class `CountOfTwoNumbers` with a **public** method `compareCountOf` that takes three parameters one is `arr` of type `int[]` and other two are `arg1` and `arg2` are of type `int` and returns true if count of `arg1` is greater than `arg2` in `arr`. The return type of `compareCountOf` should be `boolean`.

Assummptions:

1. `arr` is never null
2. `arg1` and `arg2` may be same

Here are an example:

```
Enter no of elements in the array:
6
Enter elements in the array seperated by space:
1 2 2 3 5 2
Enter the arg1 element:
2
Enter the arg2 element:
5
true
```

```
Enter no of elements in the array:
4
Enter elements in the array seperated by space:
99 -10 99 -1
Enter the arg1 element:
99
Enter the arg2 element:
99
false
```

Sample Test Cases +

Explorer CountOfT... CountOfT... Submit

```java
package q11075;

public class CountOfTwoNumbers {
    /**
     * Find the count of arg1 is more than arg2 in the arr or not
     *
     *
     * @return result
     */

    public boolean compareCountOf(int[] arr, int arg1, int arg2) {
        //write your code here
        int count1=0,count2=0;
        boolean res=false;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==arg1)
                count1++;
            if(arr[i]==arg2)
                count2++;
        }
        if(count1>count2)
            res=true;
        return res;
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

22.1.28. Problem solving with Array

03:03

Write a class `CheckAlternateNo` with a **public** method `checkAlternate` that takes two parameters one is `arr` of type `int[]` and second one is `arg` of type `int` and returns true the element `arg` is present at every odd position of the `arr`. The return type of `checkAlternate` is `boolean`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the array:
5
Enter elements in the array seperated by space:
32 65 32 84 32
Enter the arg element to find:
32
true
```

Note: Please don't change the package name

Sample Test Cases +

CheckAlt... Submit

```java
package q11076;

public class CheckAlternateNo {
	/**
	 * Check if the arg element is present at every odd postition of the arr
	 *
	 * @return result
	 */

	public boolean checkAlternate(int[] arr, int arg) {

		//Write your code here
		boolean res=false;
		for(int i=0;i<arr.length;i+=2){
			if(arr[i]==arg)
				res=true;
			else{
				res=false;
				break;
			}
		}
		return res;
	}
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

23.1.1. Problem solving with Array

08:11

Write a class `ElementCheckInEveryPair` with a **public** method `checkElement` that takes two parameters one is `arr` of type `int[]` and second one is `arg` of type `int` and returns true if every pair of `arr` contains at least one `arg`.

Assumptions:

1. `arr` is never null

These are examples for your understanding:

```
Enter no of elements in the array:
6
Enter elements in the array seperated by space:
52 2 3 2 65 2
Enter the search element:
2
true
```

```
Enter no of elements in the array:
6
Enter elements in the array seperated by space:
4 5 4 1 1 4
Enter the search element:
4
false
```

Note: Please don't change the package name.

Sample Test Cases +

ElementC... ElementC...

Submit

```java
package q11082;

public class ElementCheckInEveryPair {
    /**
     * Find every pair of array contains atleast one arg
     *
     *
     *
     *@return result
     */

    public boolean checkElement(int[] arr, int arg) {
        //Write your code here
        boolean res=false;
        for(int i=0;i<arr.length-1;i++){
            if(arr[i]==arg ||arr[i+1]==arg){
                res=true;
            }
            else{
                res=false;
                break;
            }
        }
        return res;
    }
}
```

Terminal Test cases

Shivanshojha

23.1.2. Problem solving with Array

03:27

Write a class `CompareArrays` with a **public** method `compare` that takes two parameters `arr1` and `arr2` of type `int[]` find the difference between the corresponding `arr1` and `arr2` elements and returns the count of elements whose difference is less than 2 and greater than -2. The return type of `compare` should be `int`.

Assumptions:

1. `arr1` and `arr2` never will null
2. `arr1` and `arr2` have same length

Here is an example:

```
Enter no of elements in the arr1:
3
Enter elements in the arr1 seperated by space:
1 2 3
Enter no of elements in the arr2:
3
Enter elements in the arr2 seperated by space:
2 3 4
3
```

Note: Please don't change the package name.

Sample Test Cases +

Compare... Compare... Submit

```java
package q11087;

public class CompareArrays {
	/**
	 * Compute the difference between two arrays
	 *
	 * Find the count of elements which have difference less than 2 and greater than -2
	 *
	 *
	 * @return count
	 */

	public int compare(int[] arr1, int[] arr2) {
		//Write your code here
		int count=0;
		int[] arr3=new int[arr1.length];
		for(int i=0;i<arr3.length;i++){
			arr3[i]=arr1[i]-arr2[i];
			if(arr3[i]<2 && arr3[i]>-2)
				count++;
		}
		return count;
	}
}
```

Terminal Test cases

Shivanshojha

23.1.3. Problem solving with Array

02:15

Write a class `checkSurroundedElement` with a **public** method `checkElement` that takes one parameter `arr` of type `int[]` and print all the elements in the `arr` that are surrounded by left and right elements and not equal to the left and right elements.

Here is an example:

```
Enter no of elements in the arr1:
5
Enter elements in the arr1 seperated by space:
1 2 1 1 3
2
```

Note: Please don't change the package name.

Sample Test Cases +

CheckSur... CheckSur...

Submit

```
package q11088;

public class CheckSurroundedElement {
    /**
     * Find the elements in the array surrounded by left and right and not equal to the left and right elements
     *
     *
     *
     * @print the result
     */

    public void checkElement(int[] arr) {
        //Write your code here
        for(int i=1;i<arr.length-1;i++){
            if(arr[i]!=arr[i-1] && arr[i]!=arr[i+1]){
                System.out.println(arr[i]);
            }
    }
}
```

Terminal Test cases

Shivanshojha

23.1.4. Problem solving with Array

36:46

Write a class `ReorderArray` with a **public** method `reorder` that takes one parameter `arr` of type `int[]` and returns the `arr` such that all zeros should come in front of the `arr`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the arr:
8
Enter elements in the arr seperated by space:
55 0 21 0 63 0 45 0
0
0
0
55
21
63
45
```

Note: Please don't change the package name.

Sample Test Cases +

ReorderA... ReorderAr... Submit

```java
package q11089;

public class ReorderArray {
    /**
     *Arrange all the zeros should come infront of the array
     *
     *
     *
     *@return array
     *
     */
     .
    public int[] reorder(int[] arr) {
        //Write your code here
        int arr1[]=new int[arr.length];
        int j=0;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==0){
                arr1[j]=0;
                j++;
            }
        }
        for(int k=0;k<arr1.length;k++){
            if(arr[k]!=0){
                arr1[j]=arr[k];
                j++;
            }
        }
        return arr1;
    }
}
```

Terminal Test cases

Prev Reset Submit Next

23.1.5. Problem solving with Array

05:49

Write a class `ReorderArray` with a **public** method `reorder` that takes one parameter `arr` of type `int[]` and returns the `arr` such that all even numbers in the array come to the front of the `arr`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the array:
6
Enter elements in the array seperated by space:
3 5 6 4 2 4
6
4
2
4
3
5
```

Note: Please don't change the package name.

Sample Test Cases

ReorderA... ReorderAr...

Submit

```java
public class ReorderArray {
    /**
     *Arrange all even numbers to infront of the array
     *
     *
     *
     * @return result
     */

    public int[] reorder(int[] arr) {

        //Write your code here
        int[] arr1=new int[arr.length];
        int j=0;
        for(int i=0;i<arr.length;i++){
            if(arr[i]%2==0){
                arr1[j]=arr[i];
                j++;
            }
        }
        for(int k=0;k<arr.length;k++){
            if(arr[k]%2!=0){
                arr1[j]=arr[k];
                j++;
            }
        }
        return arr1;
    }
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

23.1.6. Problem solving with Array

05:59

Write a class `MultiplesInArray` with a **public** method `findMultiples` that takes three parameters `arr` of type `int[]` and other two are `m1` and `m2` are of type `int`. Print all the elements in the array, but if any element in the `array` is a multiple of `m1`, print **multiple of** (actual value of m1 should be printed instead of). If it is a multiple of `m2`, print **multiple of** . If it is a multiple of both `m1` and `m2`, **print multiple of m1 and m2.**

For example:

```
Enter no of elements in the array:
6
Enter elements in the array separated by space:
1 2 34 5 6 7
Enter the first multiple element:
2
Enter the second multiple element:
3
1
2 is multiple of 2
34 is multiple of 2
5
6 is multiple of 2 and 3
7
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

MultiplesI... MultiplesI...

Submit

```java
package q11091;
public class MultiplesInArray{
    public void findMultiples(int[] arr,int m1,int m2){
        for(int i=0;i<arr.length;i++){
            if(arr[i]%m1==0 && arr[i]%m2==0)
            System.out.println(arr[i]+" is multiple of "+m1+" and "+m2);
            else if(arr[i]%m1==0)
            System.out.println(arr[i]+" is multiple of "+m1);
            else if(arr[i]%m2==0)
            System.out.println(arr[i]+" is multiple of "+m2);
            else
            System.out.println(arr[i]);
        }
    }
}
```

Terminal Test cases

Plots

Shivanshojha

23.1.7. Problem solving with Array

07:11

Write a class `FindCenteredAverage` with a **public** method `findCenteredAverage` that takes one parameter `arr` of type `int[]` and returns the centered average of the elements in the `arr`

Hint: exclude the biggest and smallest numbers from the array and compute the average of the remaining numbers. If there is more than one smallest value excludes only one of those. Similarly for biggest also.

Here is an example:

```
Enter no of elements in the array:
9
Enter elements in the array separated by space:
1 5 1 1 9 9 1 9 2
4
```

Note: Please don't change the package name.

Sample Test Cases +

FindCent... FindCente... Submit

```
package q11092;
public class FindCenteredAverage{
    public int findCenteredAverage(int[] arr){
        int sum=0;
        int min=arr[0];
        int max=arr[0];
        for(int i=1;i<arr.length;i++){
            if(arr[i]<min)
                min=arr[i];
            if(arr[i]>max)
                max=arr[i];
        }
        for(int i=0;i<arr.length;i++){
            sum+=arr[i];
        }
        int avg=(sum-min-max)/(arr.length-2);
        return avg;
    }
}
```

Terminal Test cases

Shivanshojha

23.1.8. Problem solving with Array

03:54

Write a class `SymmetricalArrayCheck` with a **public** method `checkSymmetry` that takes two parameters one is `arr` of type `int[]` and second one is `n` of type `int` and returns true if the first `n` numbers are same as the last `n` numbers in the `arr`.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the array:
8
Enter elements in the array seperated by space:
1 2 3 5 6 1 2 3
Enter the search number you want to search:
3
true
```

Note: Please don't change the package name.

Sample Test Cases +

Symmetri... Symmetrl... Submit

```java
package q11096;

public class SymmetricalArrayCheck {
    /**
     * Find if the first n numbers or equal to the last n numbers or not
     *
     *
     * @return result
     */

    public boolean checkSymmetry(int[] arr, int n) {
        //Write your code here
        boolean res=false;
        int len=arr.length;
        for(int i=0;i<n;i++){
            if(arr[i]==arr[len-n])
                res=true;
        // else{
        // res=false;
        // break;
        // }
    }
    return res;
    }
}
```

Terminal Test cases

Home Learn Anywhere 12216507.st@lpu.in Support Logout

23.1.9. Problem solving with Array

15:52

Write a class `SequenceCheck` with a **public** method `checkSequence` that takes one parameter `arr` of type `int[]` and returns true if any three consecutive elements in `arr` are in incremental order.

Assumptions:

1. `arr` is never null

Here is an example:

```
Enter no of elements in the array:
6
Enter elements in the array seperated by space:
1 2 3 7 4 6
true
```

Note: Please don't change the package name.

Sample Test Cases +

Sequenc... Sequence... Submit

```java
package q11097;

public class SequenceCheck {
	/**
	 * Find three consecutive elements in the array are in incremental order or not
	 *
	 *
	 *
	 * @return result
	 */

	public boolean checkSequence(int[] arr) {
		//Write your code here
		boolean res=false;
		for(int i=0;i<arr.length-2;i++){
			if(arr[i+1]==arr[i]+1 && arr[i+2]==arr[i+1]+1 ){
				res=true;
				break;
			}
		}
		return res;
	}
}
```

Terminal Test cases

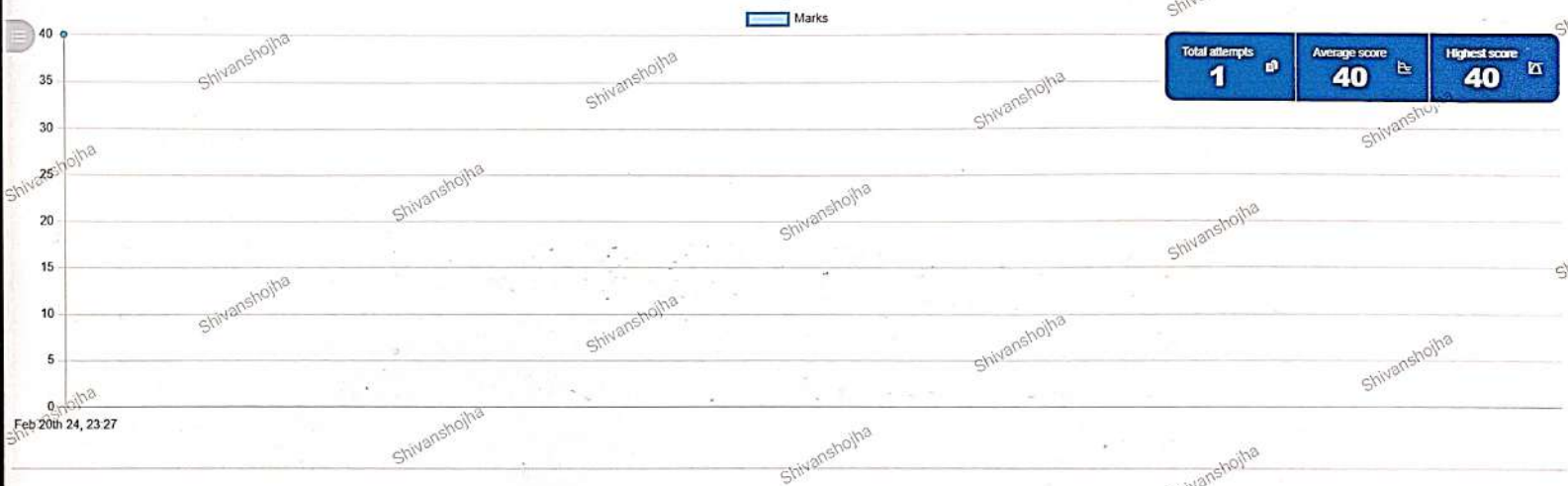
Shivanshojha

Home Learn Anywhere 12216507.st@lpu.in Support Logout

Java - Arrays [Unscheduled]

Try Again

2 hours 40 marks(s)



Total attempts	Average score	Highest score
1	40	40

Attempt	Test date	Results	Status	Actions
1	February 20th 2024, 11:27:58 pm	40/40	PASSED	Details

Shivanshojha

24.1.1. Understanding Procedural Progamming

04:40

A **programming paradigm** defines the methodology of designing and implementing programs using the key features and other building blocks of a programming language.

A **programming paradigm** gives you an idea how problems are generally analyzed and solved in a particular programming language.

Below are a few commonly used programming paradigms :

1. Procedural Programming
2. Object Based Programming
3. Object Oriented Programming

Procedural programming lays more emphasis on **procedure** than **data**. **C** language uses procedural programming style.

C++ and **Java** can also be used as a procedural programming language with some enhanced features such as **type checking, reference variables, inline functions, default arguments** etc.

Below are a few **pros** and **cons** of procedural programming languages:

Pros

1. For smaller programs the code is usually small, more readable and result roriented.
2. They are usually very efficient since they are written for a very specific purpose.

Cons

1. Since programs are function based they do not relate to a real life object.
2. When large amount of code is written inside functions, it becomes very difficult to maintain or make modifications.
3. Changing the way data is stored can required large amount of changes in code, unlike in Object oriented programs, making it difficult to maintain large systems.

Select all the correct statements given below.

- [x] A programming paradigm informs how problems are **analyzed** and **solved** in a programming language.
- [ ] Procedural programming lays more emphasis on **data** and **procedures**.
- [x] Large systems developed using procedural programming language can be difficult to maintain.

Shivanshojha

24.1.2. Understanding Object Based and Object Oriented Progamming

00:16

Object based programming is a paradigm that implements some features of object oriented programming but not all.

In object based programming, data and its associated meaningful functions are enclosed in one single entity called `class`.

Classes enforce information hiding and abstraction thereby separating the implementation details.

In object based programming, whenever there is any **change** in the definition of type, user's interface remains unaffected generally.

The object based programming can be thought of as a **subset** of object oriented programming as it implements some of the features implemented by object oriented programming like **information hiding, abstraction, classes, function overloading,** etc.

The object based programming implements **inheritance** and **polymorphism**.

The advantages of object based programming are:

- it overcomes most shortcomings of procedural programming
- it localizes changes and hides implementation details from user
- it supports user-defined types
- implements information hiding and abstraction

However, object based programming suffers from one major **limitation** and that is its inability to represent real world relationships that exist among object.

Object oriented programming paradigm is a superset of object based programming. It offers all the features of object based programming and overcomes its limitation by implementing **inheritance**.

Select all the correct statements given below.

- [x] A class is a combination of data and its associated meaningful functions.
- [x] Object based programming implements information hiding and abstraction.
- [ ] Object oriented programming does not implement inheritance and polymorphism.

Shivanshojha

24.1.3. Understanding OOP Concepts

00:11

The **object oriented programming (OOP)** approach views a problem in terms of **objects** or **entities** involved rather than the procedures to be implemented.

For instance, a `BMW X5` car is a real life **object**, whose **characteristics** are **color**, **top speed**, **number of seats**, etc. We can consider the car instance (also called object) to exhibit **behaviors** like: **start**, **accelerate**, **break**, etc.

While programming using **OOP** approach, the **characteristics** of an object are represented by its **data** and its **behavior** is represented by its **functions**.

Therefore, in object oriented programming `object` represents an entity that can store **data** and has its interface with external world through **functions**.

Another important term in object oriented programming is a `class`, which is actually a representation of **characteristics** and **behavior** of the same category (or family) of **objects**. Hence, an **object**, `BMW X5`, can be considered as an instance of a **class** called `Car`.

Select all the correct statements given below.

- [x] A class is a combination of data members and member functions.
- [x] An **object** is an instance of a **class**.
- [x] Data in an object captures the characteristics (or attributes) of that object.

Shivanshojha

24.1.4. Understanding Data abstraction and encapsulation

01:46

The object oriented programming evolved to overcome the **limitations** of procedural programming approach.

The fundamental concepts in **OOP** are:

1. Data Abstraction
2. Data Encapsulation
3. Modularity
4. Inheritance
5. Polymorphism

`Data abstraction` refers to the act of representing essential features without including the background details and explanations.

Let us consider an example of **switch board**, you can only press certain switches according to your requirement. What is happening inside, how it is happening, you need not know.

This is **abstraction**, you know only essential things to operate on **switch board** without knowing the background details of switch board.

`Data encapsulation` is the way of combining both data and functions that operate on the data under a single unit.

In this, the **data** is not accessed to the outside world directly and only through member functions, the data can be accessed.

The insulation of data from direct access by the program is called **data hiding**.

Abstraction and encapsulation are complementary concepts, where as **abstraction** focuses upon the observable behavior of an object and **encapsulation** focuses upon the implementation that gives rise to this behavior.

Encapsulation is most often achieved through **information hiding**, which is the process of hiding all

- [x] Abstraction is representing only essential things without knowing the background details.
- [ ] A class contains only member functions to secure the data.
- [x] Encapsulation is combining both data and functions.

Shivanshojha

24.2.1. What Is a Class?

(02:21)

According to English dictionary, a `class` is defined as ***a set or category of things having some property or attribute in common***.

For example, we can consider a banana, a mango and an orange to belong to a class called fruit.

In an object-oriented programming language like Java, a `class` acts as a `type`, a **template** a `blueprint`, from which instances of that type can be created.

Below is the syntax for a simple `class` called `Student`:

```
public class Student {

}
```

1. The word `Student` is the `name (identifier)` of the class.
2. The keyword `class` qualifies/tags this identifier `Student` as a **class**.
3. The word `public` is called access modifier, it is used to control access. We will learn more about access modifiers in the ensuing sections.

A class can contain members such as fields, methods, nested classes, interfaces, instance, static initializers and constructors.

In Java, `String` is a pre-defined `class` representing a sequence of characters.
So when we say :

```
String text1 = new String("Ganga");
String text2 = new String("River");
```

we are creating two instances, namely, `text1`, `text2` of type String which hold "Ganga" and "River" character sequences respectively.

Select all the correct statements given below.

- [ ] A class cannot contain another class.
- [ ] A class need not contain the keyword `class` in its declaration.
- [ ] `int` and `Integer` are predefined classes in Java.
- [x] In the statement;

```
Integer age = new Integer(25);
```

`age` is an instance of class Integer.

Shivanshojha

24.2.2. Class and Its Fields

25:48

If we consider a vehicle, say a car, we would immediately think of its model, colour, maximum speed etc. These are the various attributes also called properties of an instance of a car whose values could vary from model to model.

Similarly, when we consider a class say, Student, we can list the most common attributes/properties of a Student class as id, name, age, gender.

Note the spellings of id, name, age and gender they are written using camel-case convention and they all start with a lower case letter.

private is an access modifier like public and it is a Java keyword written in lower case.

int and char are primitive types written in lower case.
String is a class name and hence is written with the starting letter in upper case.

In the "StudentMain.java" file:
The code declares a class called "StudentMain".
It contains a main method that serves as the entry point for the program.
Inside the main method, an object of the "Student" class is created
Finally, the message "Private variable found: <variable>" is printed to the console.
Now, In **Student.java**, declare a class named **Student** with private member variables **id** (String), **name** (String), **age** (int), and **gender** (char).

Note: Please don't change the package name.

Sample Test Cases +

Explorer Student.j... StudentM... Submit

```
package q11154;
// write your code here
public class Student{
    private String id;
    private String name;
    private int age;
    private char gender;
}
```

Terminal Test cases

Shivanshojha

24.2.3. Student Class with Fields

03:49

In a class the fields are usually declared with `private` access modifier. We will learn more about access modifiers in the ensuing sections.

A field declaration could also contain initialization. For example:

```
private int age = 25;
```

Field declarations of similar types can be done as a single statement. For example:

```
private int age = 25, totalMarks = 100;
```

Note that every statement must be terminated by a semicolon.

Identify the errors and correct them.

Note: Please don't change the package name.

Sample Test Cases

Explorer

Student.j... StudentM...

Submit

```
package q11155;
public class Student {
	private String id;
	private String name;
	private int age;
	private char gender;
}
```

Terminal Test cases

Plots

Shivanshojha

Home Learn Anywhere

12216507.st@lpu.in Support Logout

24.2.4. Correct the errors

00:20

Identify the errors in below code and correct them.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

Student.j... StudentM...

Submit

```
package q11217;
public class Student {
	private String id;
	private String name;
	private int age;
	private char gender;
}
```

Terminal Test cases

Plots

Shivanshojha

25.1.1. What Is a Constructor?

05:22

A **constructor** is a special method (block of code) which is used to create instances of a class.

A constructor is also used to initialize the fields of a class.

A constructor's name **must** always **be the same** as the class name.

A constructor is a special method which does not have a **return type**.

It is a good practice to provide the arguments in the same order as they are declared in the class, if they happen to be the fields of the class.

Complete the given code in **Student.java** file to understand the usage of `this` that differentiates the identifiers with same names in different scopes.

For example, in the statement **this.id = id;**, `this.id` refers to the instance field `private String id;`, and the `id`, which is after the assignment operator (`=`) refers to the `String id` passed as parameter into the constructor.

Sample Test Cases +

StudentM... Student.java Submit

```java
import java.util.Scanner;

public class StudentMain {
    public static void main(String[] args) {
        //Student st = new Student("56", "Rama", 29, 'M');
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the student's ID: ");
        String id = scanner.nextLine();

        System.out.print("Enter the student's name: ");
        String name = scanner.nextLine();

        System.out.print("Enter the student's age: ");
        int age = scanner.nextInt();

        System.out.print("Enter the student's gender (M/F): ");
        char gender = scanner.next().charAt(0);

        Student st = new Student(id, name, age, gender);

        System.out.println("Student details:");
        System.out.println("ID: " + st.getId());
        System.out.println("Name: " + st.getName());
        System.out.println("Age: " + st.getAge());
        System.out.println("Gender: " + st.getGender());
        System.out.println("Good job!");
    }
}
```

Terminal Test cases

Home Learn Anywhere 12216507.st@lpu.in Support Logout

25.1.2. Correct the errors

00:20

In a class if the name of the constructor is different from the name of the class, Java treats it as a different method.

Press Submit to see compilation error and then correct the error.

[**Hint**: Java is case sensitive, meaning change in the case of characters will make it a different name.]

Note: Please don't change the package name.

Sample Test Cases +

Explorer Student.j... StudentM... Submit Plots

```java
package q11143;
public class Student {
	private String id;
	private String name;
	private int age;
	private char gender;
	public Student(String id, String name, int age, char gender) {
		this.id = id;
		this.name = name;
		this.age = age;
		this.gender = gender;
	}
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

Home Learn Anywhere

12216507.st@lpu.in Support Logout

25.1.3. Correct the errors

00:38

Identify the errors in the given code and correct them.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

Student.j... StudentM...

Submit

```
package q11159;
public class Student {
	private String id;
	private String name;
	private int age;
	private char gender;
}
```

Terminal Test cases

Plots

< Prev Reset Submit Next >

Shivanshojha

25.1.4. Correct the errors

01:29

Identify the errors and correct it.

Note: Please don't change the package name

Sample Test Cases +

Student.j... StudentM... Submit

```java
package q11160;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;

    public Student(String id, String name, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
}
```

Terminal Test cases

Shivanshojha

25.1.5. Correct the error

00:28

A constructor must be written inside the class body (i.e. inside the open brace { and the closing brace } of the class body).

Identify the errors and correct them.

Note: Please don't change the package name

Sample Test Cases +

Explorer | Student.j... | StudentM... | Submit | Plots

```
package ql1161;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;


    public Student(String name, String rollNo, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
}
```

Terminal | Test cases

Shivanshojha

26.1.1. A method and its structure

04:59

In Java, a method (also called as function) is a named block of code that performs a task.

Like a mathematical function `f(x)` (read as ***f of x***), in Java, a method (or a function) can act on some input data to produce an output.

The input is not always compulsory, for example we could have a method called `void printCurrentTime()`, which could internally fetch the System's current time and print the same.

Similarly, the output of a function need not always be printed to a console, the function could store the output to a file, or a database or even return the calculated value to be used for further processing.

There are 5 basic parts to a method:

1. Method Name - the name used identify and invoke/execute the method.
2. Method Body - contains the code statements which will be executed when the method is called/invoked.
3. Method Parameters - [***optional***] contain the input values which are used by the statements in the method body.
4. Method Return Type - indicates what type of value is returned, if the method returns nothing, then `void` is used as the return type.
5. Modifiers - [***optional***] modifiers constitute one or more keywords that inform how the method can be used. We will learn more about them later.

Below is an example of a method which has parameters and returns a value:

```
public int sum(int num1, int num2) {
    return num1 + num2;
}
```

1. `sum` is the method name.
2. the content between opening `{` and closing `}` braces constitutes the method body.
3. `int num1, int num2`, which are passed between opening `(` and closing `)` parentheses are called the method parameters.

- [ ] A method can be called only once.
- [ ] A method should contain at least one parameter.
- [ ] There should not be a space between method name and the opening parenthesis `(` which follows it.
- [ ] There should be only one space between the closing parenthesis `)` and the opening brace `{` which marks the beginning of the method body.
- [x] Method name should always start a lowercase letter.

Shivanshojha

26.1.2. A class with fields, constructor and toString() method

10:54

`toString()` is a special method used to convert an object's state information into a human readable text.

Note the spelling of `toString()`

The `toString()` method does not take any parameters and will always return a `String`.

Here in the editor, We have two Java files.

- **StudentMain.java**
- **Student.java**

The functionality of the **StudentMain.java** class is The main method that prompts the user to enter the details of a student, creates a Student object using the provided information, and displays the details of the student using the **toString** method.

Now, Write a Java program that creates a **Student** class with private fields for id, name, age, and gender. The Student class should have a constructor to initialize these fields and an overridden **toString** method to provide a string representation of the Student object.

Refer to the displayed test cases for better understanding

Note:

- Please don't change the package name
- The code provided in the editor of class **StudentMain** reads the inputs using the nextLine() and nextInt() methods and stores them in id, name, age, and gender variables.

Sample Test Cases +

Explorer | StudentM... | Student.java | Submit

```java
package q11126;
import java.util.*;
public class StudentMain {
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        String id= sc.nextLine();
        String name=sc.nextLine();
        int age = sc.nextInt();
        char gender = sc.next().charAt(0);
        Student st = new Student(id,name,age,gender);
        System.out.println("The student is: " + st);
    }
}
```

Terminal | Test cases

Shivanshojha

Home Learn Anywhere

12216507.st@lpu.in Support Logout

26.1.3. Correct the errors

00:51

The `toString()` is a special method because when the object is involved in the string concatenation then the toString() method returns the String representation of that object. Hence a `toString()` method's return type is `String` and it does not take any parameters.

Identify and correct the errors based on the above information provided. Hint: Press Submit to find out the error.

Note: Please don't change the package name

Sample Test Cases +

Explorer

Student.j... StudentM...

Submit

```java
package q11125;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;

    public Student(String id, String name, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
```

Plots

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

26.1.3. Correct the errors

00:51

The `toString()` is a special method because when the object is involved in the string concatenation then the toString() method returns the String representation of that object. Hence a `toString()` method's return type is `String` and it does not take any parameters.

Identify and correct the errors based on the above information provided. Hint: Press Submit to find out the error.

Note: Please don't change the package name

Sample Test Cases

Student.j... StudentM...

Submit

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public char getGender() {
        return gender;
    }

    public String toString() {
        return "Student [id = " + id + ", name = " + name + ", age = " + age + ", gender = " + gender + "]";
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

26.1.4. Correct the errors

01:07

The `toString()` method like the constructor of a class is part of the class body. It must be written inside a class scope (inside the opening and closing brace of the class).

Hint: You can select some code and press ALT + Up key or ALT + Down key to move the selected code up and down. (**Note**: This works only in Windows and not in MAC and Linux)

Identify and correct the error.

Note: Please don't change the package name

Sample Test Cases +

Explorer Student.j... StudentM... Submit Plots

```
package q11128;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;

    public Student(String name, String id, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
    public String toString() {
        return "Student [name = " + name + ", id = " + id + ", age = " + age + ", gender = " + gender + "]";
    }

}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

Home Learn Anywhere 12216507.st@lpu.in Support Logout

26.1.5. Correct the errors

00:48

Identify the errors and correct them.

Hint: Press Submit to see the errors and note the spellings.

Note: Please don't change the package name.

Sample Test Cases +

Explorer Student.j... StudentM... Submit Plots

```
package q11129;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;

    public Student(String id, String name, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    public String toString() {
        return "Student [name = " + name + ", id = " + id + ", age = " + age + ", gender = " + gender + "]";
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

26.1.6. Correct the errors

00:41

Whenever on compilation, you see an error saying: `xyz cannot be resolved`, it means that the compiler is unable to resolve the symbol `xyz` to a type.

On such errors first verify if the symbol `xyz` is spelt correctly.

Identify the error and correct it.

Note: Please don't change the package name

Sample Test Cases

Student.j... StudentM...

Submit

```java
package q11130;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;

    public Student(String id, String name, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    public String toString() {
        return "Student [name = " + name + ", id = " + id + ", age = " + age + ", gender = " + gender + "]";
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

26.1.7. Correct the error.

01:57

You will notice that the line `public class Student {` is indented to the left most column.

At the same time you will also notice that the lines inside the class body are indented by one level.

Indenting code is indented to make code easily readable. It helps clearly identify the start and end of independent code blocks, like classes, methods loops etc.

One can either place the opening brace (`{`) in a new line or at the end of the statement, In all our examples we will place them at the end of the statement so that we can see more lines on the screen.

The general rules for indentation are as follows:

1. Press the tab key to indent and not space bar.
2. Indent the line which immediately follows the opening brace.
3. Indent the closing brace (`}`) to match with the beginning of the line which has the opening brace (`{`).

Identify the error and correct it.

Hint: Press the Submit button to find the error. Note if the correct operator is used for String concatenation.

Note: Please don't change the package name

Sample Test Cases +

Student.j... StudentM... Submit

```
package q11131;
public class Student {
	private String id;
	private String name;
	private int age;
	private char gender;
	
	public Student(String id, String name, int age, char gender) {
		this.id = id;
		this.name = name;
		this.age = age;
		this.gender = gender;
	}
	
	public String toString() {
		return "Student [name = " + name + ", id = " + id + ", age = " + age + ", gender = " + gender + "]";
	}
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

26.1.8. What are accessor methods?

06:26

The state information of an instance of Student class is stored in its instance fields.

These fields are usually marked as `private`.

These fields are initialized using the constructor. Hence, after the values are initialized, getXXX and setXXX methods are written to retrieve and modify their values.

A setXXX method is called a setter method. For a field named age, the setter method would be written as:

```
public void setAge(int age) {
    this.age = age;
}
```

A getXXX method is called a getter method. For a field named age, the getter method would be written as:

```
public int getAge() {
    return age;
}
```

These `getter` and `setter` methods are also called as `accessor` methods.

Note how the `getter` and `setter` method names follow camel-case naming convention.

There are two Java files **StudentMain** and **Student**.

Functionality of the class **StudentMain.java:**

Sample Test Cases +

Explorer

Student.j... StudentM... Submit

```
package q11132;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;

    public Student(String id, String name, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
    public void setId(String id){
        this.id=id;
    }
    public String getId(){
        return id;
    }
    public void setName(String name) {
        this.name=name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age=age;
    }

```

Terminal Test cases

All Plots

Shivanshojha

26.1.8. What are accessor methods?

06:26

The state information of an instance of Student class is stored in its instance fields.

These fields are usually marked as `private`.

These fields are initialized using the constructor. Hence, after the values are initialized, getXXX and setXXX methods are written to retrieve and modify their values.

A setXXX method is called a setter method. For a field named age, the setter method would be written as:

```
public void setAge(int age) {
    this.age = age;
}
```

A getXXX method is called a getter method. For a field named age, the getter method would be written as:

```
public int getAge() {
    return age;
}
```

These `getter` and `setter` methods are also called as `accessor` methods.

Note how the `getter` and `setter` method names follow camel-case naming convention.

There are two Java files **StudentMain** and **Student**.

Functionality of the class **StudentMain.java**:

Sample Test Cases +

Student.j... StudentM... Submit

```
        return id;
    }
    public void setName(String name) {
        this.name=name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age=age;
    }

    public int getAge() {
        return age;
    }

    public void setGender(char gender) {
        this.gender=gender;
    }

    public char getGender() {
        return gender;
    }

    public String toString() {
        return "Student [id = "+id+", name = "+name+", age = "+age+", gender = "+gender+"]";
    }
}
```

Terminal Test cases

Shivanshojha

26.1.9. Correct the errors

00:57

Identify the errors and correct them.

Note: Please don't change the package name.

Sample Test Cases

Student.j... StudentM...

Submit

```java
package q11134;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;

    public Student(String id, String name, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

26.1.9. Correct the errors

00:57

Identify the errors and correct them.

Note: Please don't change the package name.

Sample Test Cases +

Student.j... StudentM...

Submit

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public char getGender() {
        return gender;
    }

    public String toString() {
        return "Student [id = " + id + ", name = " + name + ", age = " + age + ", gender = " + gender + "]";
    }
}
```

Terminal Test cases

26.1.10. Correct the errors

01:11

A `setter` method will always have `void` as return type and takes the field's type as an argument into the method.

`void` return type indicates that the method does not return a value. It only performs some operation.

For example :

```
public void setName(String name) {
    this.name = name;
}
```

Identify the errors and correct them.

Note: Please don't change the package name

Sample Test Cases +

Student.j... StudentM... Submit

```
    return id;
}

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public char getGender() {
        return gender;
    }

    public String toString() {
        return "Student [id = " + id + ", name = " + name + ", age = " + age + ", gender = " + gender + "]";
    }
```

Terminal Test cases

Shivanshojha

26.1.11. Correct the errors

01:11

A `getter` method of a field will always have the field's type as the return type and will not have any argument.

It does not take any argument since it simply returns the value of the field held by the instance.

For example :

```
public String getName() {
    return name;
}
```

Using the above information identify the errors in the below code and correct them. Press Submit to spot the errors.

Note: Please don't change the package name

Sample Test Cases

Student.j... StudentM...

Submit

```
package q11136;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;

    public Student(String id, String name, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

26.1.11. Correct the errors

01:11

A `getter` method of a field will always have the field's type as the return type and will not have any argument.

It does not take any argument since it simply returns the value of the field held by the instance.

For example :

```
public String getName() {
    return name;
}
```

Using the above information identify the errors in the below code and correct them. Press Submit to spot the errors.

Note: Please don't change the package name

Sample Test Cases

Student.j... StudentM...

Submit

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public char getGender() {
        return gender;
    }

    public String toString() {
        return "Student [id = " + id + ", name = " + name + ", age = " + age + ", gender = " + gender + "]";
    }
}
```

Terminal Test cases

Shivanshojha

Home Learn Anywhere

12216507.st@lpu.in Support Logout

26.1.12. Correct the errors

03:24

In the below you will find that some of the `getter` and `setter` methods have wrong return types.

Use the declared type of the field to correct the return types of `getter` methods and ponder upon what should be the return types of the `setter` methods to correct them.

Identify the errors and correct them. Hint: Press Submit to see the errors.

Note: Please don't change the package name

Sample Test Cases +

Explorer

Student.j... StudentM...

Submit

```java
package q11137;
public class Student {
    private String id;
    private String name;
    private int age;
    private char gender;

    public Student(String id, String name, int age, char gender) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
```

Plots

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

26.1.12. Correct the errors

03:24

In the below you will find that some of the `getter` and `setter` methods have wrong return types.

Use the declared type of the field to correct the return types of `getter` methods and ponder upon what should be the return types of the `setter` methods to correct them.

Identify the errors and correct them. Hint: Press Submit to see the errors.

Note: Please don't change the package name

Sample Test Cases +

Student.J... StudentM... Submit

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public char getGender() {
        return gender;
    }

    public String toString() {
        return "Student [id = " + id + ", name = " + name + ", age = " + age + ", gender = " + gender + "]";
    }
}
```

Terminal Test cases

Shivanshojha

26.1.13. Correct the errors

01:18

In the below code you will find that wrong parameter types have been provided to the `setter` methods.

Use the declared field's type to correct the parameter type of the `setter` methods.

Identify the errors and correct them. Hint: Press Submit to view the errors.

Note: Please don't change the package name

Sample Test Cases

Student.j... StudentM...

Submit

```java
package q11138;
public class Student {
	private String id;
	private String name;
	private int age;
	private char gender;

	public Student(String id, String name, int age, char gender) {
		this.id = id;
		this.name = name;
		this.age = age;
		this.gender = gender;
	}

	public void setId(String id) {
		this.id = id;
	}

	public String getId() {
		return id;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public void setAge(int age) {
```

Terminal Test cases

Shivanshojha

26.1.13. Correct the errors

01:18

In the below code you will find that wrong parameter types have been provided to the `setter` methods.

Use the declared field's type to correct the parameter type of the `setter` methods.

Identify the errors and correct them. Hint: Press Submit to view the errors.

Note: Please don't change the package name

Sample Test Cases

Student.j... StudentM...

Submit

```java
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public char getGender() {
        return gender;
    }

    public String toString() {
        return "Student [id = " + id + ", name = " + name + ", age = " + age + ", gender = " + gender + "]";
    }
}
```

Terminal Test cases

Shivanshojha

Home Learn Anywhere

12216507.st@lpu.in Support Logout

26.1.14. Correct the errors

00:31

The `toString()` is a special method which is used to convert the state information stored in the fields into a String and return it.

Hence a `toString()` method's return type is `String` and it does not take any parameters.

Identify and correct the errors based on the above information provided. Hint: Press Submit to find out the error.

Note: Please don't change the package name

Sample Test Cases +

Student.j... StudentM... Submit

```java
package q11140;
public class Student {
	private String id;
	private String name;
	private int age;
	private char gender;

	public Student(String id, String name, int age, char gender) {
		this.id = id;
		this.name = name;
		this.age = age;
		this.gender = gender;
	}

	public void setId(String id) {
		this.id = id;
	}

	public String getId() {
		return id;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public void setAge(int age) {
```

Terminal Test cases

Shivanshojha

Home Learn Anywhere

12216507.st@lpu.in Support Logout

26.1.14. Correct the errors

00:31

The `toString()` is a special method which is used to convert the state information stored in the fields into a String and return it.

Hence a `toString()` method's return type is `String` and it does not take any parameters.

Identify and correct the errors based on the above information provided. Hint: Press Submit to find out the error.

Note: Please don't change the package name

Sample Test Cases +

Student.j... StudentM...

Submit

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public char getGender() {
        return gender;
    }

    public String toString() {
        return "Student [id = " + id + ", name = " + name + ", age = " + age + ", gender = " + gender + "]";
    }
}
```

Terminal Test cases

Shivanshojha

26.1.15. Difference between Parameters and Arguments

00:26

The list of variables/references that are present in the method declaration are called **parameters**.

When the method is invoked the actual values passed are called **arguments**.

Select all the correct statements for the below code:

```
public int sum(int num1, int num2) { // statement 1
    return num1 + num2;
}

int total = sum(2, 3); // statement 2
```

- [x] In statement 1, `num1` and `num2` are called parameters.
- [ ] In statement 1, `num1` and `num2` are called arguments.
- [x] In statement 2, values `2` and `3` are called arguments.
- [ ] In statement 2, values `2` and `3` are called parameters.

Shivanshojha

27.1.1. Write a Java program to implement a Class mechanism

17:20

Write a Java program with a class name `Employee` which contains the data members **name** (String), **age** (int), **designation** (String), **salary** (double) and the methods **setData()**, **displayData()**.

The member function **setData()** is used to initialize the data members and **displayData()** is used to display the given employee data.

Write the **main()** method with in the class which will receive four arguments as **name, age, designation**and **salary**.

Create an object to the class `Employee` within the main(), call **setData()** with arguments and finally call the method **displayData()** to print the output.

If the input is given as command line arguments to the **main()** as **"Saraswathi", "27", "Teacher", "37250"** then the program should print the output as:

```
Name : Saraswathi
Age : 27
Designation : Teacher
Salary : 37250.0
```

Note: Please don't change the package name.

Sample Test Cases +

Employe... Submit

```java
package q11115;
public class Employee{
	String name;
	int age;
	String designation;
	double salary;
	// void setData();
	// void displayData();
	// public static void main(String[] args){
	// 	String name=args[0];
	// 	int age=Integer.parseInt(args[1]);
	// 	String designation=args[2];
	// 	double salary=Double.parseDouble(args[3]);

		void setData(String name,int age,String designation,double salary){
			this.name=name;
			this.age=age;
			this.designation=designation;
			this.salary=salary;
		}

		void displayData(){
			System.out.println("Name : "+name);
			System.out.println("Age : "+age);
			System.out.println("Designation : "+designation);
			System.out.println("Salary : "+salary);
		}
		public static void main(String[] args){
			String name=args[0];
			int age=Integer.parseInt(args[1]);
```

Terminal Test cases

Shivanshojha

27.1.1. Write a Java program to Implement a Class mechanism

17:20

Write a Java program with a class name `Employee` which contains the data members **name** (String), **age** (int), **designation** (String), **salary** (double) and the methods **setData()**, **displayData()**.

The member function **setData()** is used to initialize the data members and **displayData()** is used to display the given employee data.

Write the **main()** method with in the class which will receive four arguments as **name, age, designationand salary.**

Create an object to the class `Employee` within the main(), call **setData()** with arguments and finally call the method **displayData()** to print the output.

If the input is given as command line arguments to the **main()** as **"Saraswathi", "27", "Teacher", "37250"** then the program should print the output as:

```
Name : Saraswathi
Age : 27
Designation : Teacher
Salary : 37250.0
```

Note: Please don't change the package name.

Sample Test Cases +

Employe... Submit

```java
// String name=args[0];
// int age=Integer.parseInt(args[1]);
// String designation=args[2];
// double salary=Double.parseDouble(args[3]);

    void setData(String name,int age,String designation,double salary){
        this.name=name;
        this.age=age;
        this.designation=designation;
        this.salary=salary;
    }

    void displayData(){
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
        System.out.println("Designation : "+designation);
        System.out.println("Salary : "+salary);
    }
    public static void main(String[] args){
        String name=args[0];
        int age=Integer.parseInt(args[1]);
        String designation=args[2];
        double salary=Double.parseDouble(args[3]);

        Employee obj=new Employee();
        obj.setData(name,age,designation,salary);
        obj.displayData();
    }
}
```

Terminal Test cases

Shivanshojha

27.1.2. Write a Java program to implement a Constructor

03:56

Write a Java program with a class name `Staff`, contains the data members **id** (int), **name** (String) which are initialized with a **parameterized constructor** and the method **show()**.

The member function **show()** is used to display the given staff data.

Write the **main()** method with in the class which will receive two arguments as **id** and **name**.

Create an object to the class `Staff` with arguments **id** and **name** within the **main()**, and finally call the method **show()** to print the output.

If the input is given as command line arguments to the **main()** as **"18", "Gayatri"** then the program should print the output as:

```
Id : 18
Name : Gayatri
```

Note: Please don't change the package name.

Sample Test Cases +

Staff.java

Submit

```java
package q11116;
public class Staff{
	int id;
	String name;
	Staff(int id,String name){
		this.id=id;
		this.name=name;
	}
	void show(){
		System.out.println("Id : "+id);
		System.out.println("Name : "+name);
	}
	public static void main(String[] args){
		int id=Integer.parseInt(args[0]);
		String name=args[1];
		Staff obj=new Staff(id,name);
		obj.show();
	}
}
```

Terminal Test cases

Shivanshojha

27.1.3. Write a Java program to Access the instance variables by using this keyw...

10:05

Write a Java program to demonstrate how to access instance variables by using `this` keyword.

Create a class `Box` with

- **length**, **breadth** and **height** as data members of type **float**
- parameterized constructor with **three** arguments to initialize the data members with the argument values
- method to find the `volume` of the box
- method to compare the `volumes` of two boxes

Write a **main()** function to create objects of the class and to compare the objects.

If the input is given as command line arguments to the **main()** as **["2.5", "3.4", "2.9", "1.6", "2.1", "1.9"]** then the program should print the output as:

```
box2 is smaller than box1
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

Box.java

Submit

```java
package q11117;
public class Box {

    // Declare variables
    float length;
    float breadth;
    float height;

    public Box(float l, float b, float h) {

        // Write the code
        length=l;
        breadth=b;
        height=h;
    }
    public float volume() {

        // Write the code
        return length*breadth*height;
    }
    public int compare(Box b) {
        if(this.volume()>b.volume())
            return 1;
        else
            return -1;
        // Write the code
    }
    public static void main(String args[]) {
        int flag;
        Box box1 = new Box(Float.parseFloat(args[0]),
Float.parseFloat(args[1]), Float.parseFloat(args[2]));
```

Terminal Test cases

Shivanshojha

27.1.4. What are objects?

06:35

An instance of a class in memory during program execution is called an object. It encapsulates state information using fields and behaviour using methods.

Instances of a class are created by using the `new` keyword followed by the constructor. For example:

```
String name = new String("Albert Einstien");
Date today = new Date();
Integer luckyNumber = new Integer(7);
```

In the above code `name`, `today` and `luckyNumber` are called *references* which store the address of the objects created in memory during execution.

In Java, dot `.` is used to access a member (field or a method) of an object. For example:

```
name.toLowerCase();
name.substring(6);
```

In the given Editor, Follow the comment lines to Understand and complete the code.
Note: Please don't change the package name.

Sample Test Cases +

Explorer

Person.java

Submit

```
package q11119;
import java.util.*;
public class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
        //intialize the instance variables(name and age)
        this.name=name;
        this.age=age;
    }
    public String getName() {
        //return the value of the name variable
        return name;
    }
    public String toString() {
        /*provide the string representation
        of the "Person" object as shown in displayed
        test cases */
        return "person = Person ["+"name = "+name+", age = "+age+"]";

    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);//Scanner class is used to read input from the user
        String p_name = sc.nextLine();//person's name is reading from the user.
        int p_age = sc.nextInt();//person's age is also reading from the user.
        /* create a new "Person" object and
        print the string representation as displayed
```

Terminal Test cases

Plots

27.1.4. What are objects?

06:35

An instance of a class in memory during program execution is called an object. It encapsulates state information using fields and behaviour using methods.

Instances of a class are created by using the `new` keyword followed by the constructor. For example:

```
String name = new String("Albert Einstien");
Date today = new Date();
Integer luckyNumber = new Integer(7);
```

In the above code `name`, `today` and `luckyNumber` are called ***references*** which store the address of the objects created in memory during execution.

In Java, dot `.` is used to access a member (field or a method) of an object. For example:

```
name.toLowerCase();
name.substring(6);
```

In the given Editor, Follow the comment lines to Understand and complete the code.
Note: Please don't change the package name.

Sample Test Cases +

Person.java

Submit

```
        this.name=name;
        this.age=age;
    }
    public String getName() {
        //return the value of the name variable
        return name;
    }
    public String toString() {
        /*provide the string representation
        of the "Person" object as shown in displayed
        test cases */
        return "person = Person ["+"name = "+name+", age = "+age+"]";

    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);//Scanner class is used to read input from the user
        String p_name = sc.nextLine();//person's name is reading from the user.
        int p_age = sc.nextInt();//person's age is also reading from the user.
    /* create a new "Person" object and
    print the string representation as displayed
    in the test cases*/
        Person obj=new Person(p_name,p_age);
        System.out.println(obj);
        System.out.println("name of person is : "+obj.getName());

    }
}
```

Terminal Test cases

27.2.1. What is a package and how to use it?

05:22

In Java related classes and interfaces can be stored together in folders known as packages.

For example in Java, all the common classes like `String`, `System` etc which we have used are in a package called `java.lang`.

There are many such packages in Java, for example:

1. `java.util` - this package contains utility and data structure related classes like Date, ArrayList, HashSet, etc., which we will learn later.
2. `java.io` - this package contains classes which help in reading input and writing output to files and streams.
3. `java.net` - this package contains networking related classes.

If we want to use a `Date` class which is in `java.util` package in our class, we need to inform Java compiler that we are interested in using the `Date` class of `java.util` package.

This can be done in the below three ways, either by using an import statement or fully qualified class name:

1. By importing all class of java.util package - `import java.util.*;` - this statement should occur above the class declaration.
2. By importing only the required class of java.util package - `import java.util.Date;` - this statement should occur above the class declaration.
3. By directly using the fully qualified name of the class in the declaration statement - `java.util.Date date = new java.util.Date();` - we need not have an import when we do this.

Note: `java.lang` package which contains all commonly used Java classes is by default imported. Which means for using a `String` or an `Integer` or the `System` class and the like, one need not import the `java.lang` package.

Java allows for custom packages to organize the classes. We need to use a package statement like below at the top of a source file to indicate that our class belongs to a certain package. For example:

```
package com.mycompany.games;
```

- [ ] Example1 will not compile since the compiler does not know what is a `Date` class.
- [x] Example2 will compile properly as there is an import statement in the class.
- [ ] Example2 is more efficient than Example1 as it does not import all class, which the Example1 is doing by using `import java.util.*;`
- [ ] User defined package names should always start with `java.` like `java.lang` etc.
- [x] Example3 will compile even though there is no import statement.

Shivanshojha

27.2.2. Encapsulation meaning and usage

02:23

Encapsulation is a feature in object-oriented languages, like Java where we can selectively hide the data and members, so that they are not accidentally corrupted by code.

For example in the below class:

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    private void setName(String name) {
        this.name = name;
    }

    private void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
        //we can also flag an error when invalid age is passed
    }
}
```

`String name` and `int age` are declared with `private` access modifier.

`private` modifier indicates that those members are accessible only by code of that class. They are not visible to other classes.

Sample Test Cases +

Explorer

Test.java

Submit

```
package q11121;
class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
    if (age > 0) {
        this.age = age;
    }
    }

    public String toString() {
        return "Person [name = " + name + ", age = " + age + "]";
    }
```

Terminal Test cases

Shivanshojha

27.2.2. Encapsulation meaning and usage

02:23

Encapsulation is a feature in object-oriented languages, like Java where we can selectively hide the data and members, so that they are not accidentally corrupted by code.

For example in the below class:

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    private void setName(String name) {
        this.name = name;
    }

    private void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
        //we can also flag an error when invalid age is passed
    }
}
```

`String name` and `int age` are declared with `private` access modifier.

`private` modifier indicates that those members are accessible only by code of that class. They are not visible to other classes.

Sample Test Cases +

Explorer

Test.java

Submit

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
    }

    public String toString() {
        return "Person [name = " + name + ", age = " + age + "]";
    }
}

public class Test {
    public static void main(String[] args) {
        Person p1 = new Person("Albert Einstein", 25);
        //System.out.println("name of p1 is : " + p1.name); //the line with error
        System.out.println("name of p1 is : " + p1.getName());
    }
}
```

Plots

Terminal Test cases

Shivanshojha

27.2.3. Different access modifiers in Java

01:21

Access modifiers are special keywords which can be associated to members like fields, methods, classes. These access modifiers define the degree of access control to those members.

The four different access control modifiers are:

1. `private` - marks the member as accessible only in the declaring class
2. `package/default` - marks the member as accessible to classes in the same package
3. `protected` - marks the members accessible in subclasses of the class and to classes in the same package
4. `public` - marks the members as accessible to all classes

Note: When none of the access modifier keywords like `public`, `private` or `protected` are used, the default or package access restrictions apply.

Select all the correct statements for the below code:

```
class A {
    private int x;
    public int y;
    public void sayHello() {
        System.out.println("Hello");
    }
    private void sayBye() {
        System.out.println("Hello");
    }
}
class B {
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.x); // Statement 1
        System.out.println(a1.y); // Statement 2
        System.out.println(a1.sayHello());  // Statement 3
        System.out.println(a1.sayBye()); // Statement 4
    }
}
```

- [ ] Statement 1 will not produce a compilation error.
- [ ] Statement 2 will produce a compilation error.
- [x] Statement 3 will not produce a compilation error.
- [ ] Statement 4 will not produce a compilation error.

Shivanshojha

27.3.1. Understanding different types of variables

03:01

In Java, we can classify variables into two types:

1. `primitive variables` - these are also simply called as `variables`
2. `references`

Below are examples of `variables` (formally called as `primitive variables`):

```
int x = 3;
char gender = 'F';
boolean isCorrectCode = true;
```

In the above declarations, you will notice that the identifiers `x`, `gender` and `isCorrectCode` are of primitive integer (`int`), character (`char`) and `boolean` data types respectively.

Henceforth we will use the word `variable` to refer to variables whose type is of primitive type (`int`, `short`, `long`, `float`, `byte`, `double`, `char` and `boolean`).

Below are examples of `references` (formally called as `reference variables`):

```
String firstName = "Appolonius";
Date currentDate = new Date();
Student st1 = new Student("1007", "Ganga", 25, 'F'); //assuming there is a custom class called Stud
int[] luckyNumbersArr = {3, 5, 7};
String[] cityNamesArr = {"Jerusalem", "Varanasi", "Rome"};
```

Hereafter we will refer to all variables of class types or interface types or even array types as `references`.

We can assume a `variable` as a placeholder that stores a value. For example, in the above declarations variable `x` is like a cup that stores a value `3`.

Similarly we can assume a `reference` as a placeholder (cup) that points to the object.

In Java, instances of arrays (can be primitive array like `int[]` or a array of a class like `String[]`)

- [ ] `age` is called a `reference`
- [ ] The line containing `studentArr` will result in a compilation error, since we cannot create an array of type Student class.
- [x] `currentTimeInMilliSeconds` is a variable and not reference.
- [ ] `studentArr` is not a reference.
- [x] `welcomeMsg` is a reference.
- [x] `flagsArr` is a reference.

Shivanshojha

27.4.1. Usage of void

01:29

In Java, the `void` keyword is a placeholder to indicate that there is no return type for a method.

For example, the below method adds `num1` and `num2` and returns the result.

```
public int sum(int num1, int num2) {
    return num1 + num2;
}
```

Since the type of the result being returned is also `int`, the method's return type is marked `int`.

In the below example, you will notice that the method `printSum` does not return but prints the summation to the System's console.

```
public void printSum(int num1, int num2) {
    System.out.println( num1 + num2);
}
```

Since the method `printSum` does not return any value, the return type of the method is `void`. By this the compiler understands that this method does not return any.

Considering the above two methods are available, select all the correct statements from the below code:

```
int total1 = printSum(4, 5); //Statement 1
int total2 = sum(4, 5); //Statement 2
System.out.println(printSum(1, 2)); //Statement 3
System.out.println(sum(1, 2)); //Statement 4
```

- [ ] In Statement 1, the value of `total1 = 9`.
- [x] In Statement 2, the value of `total2 = 9`.
- [ ] Statement 3 prints a value of 3 to console.
- [x] Statement 4 prints a value of 3 to console.

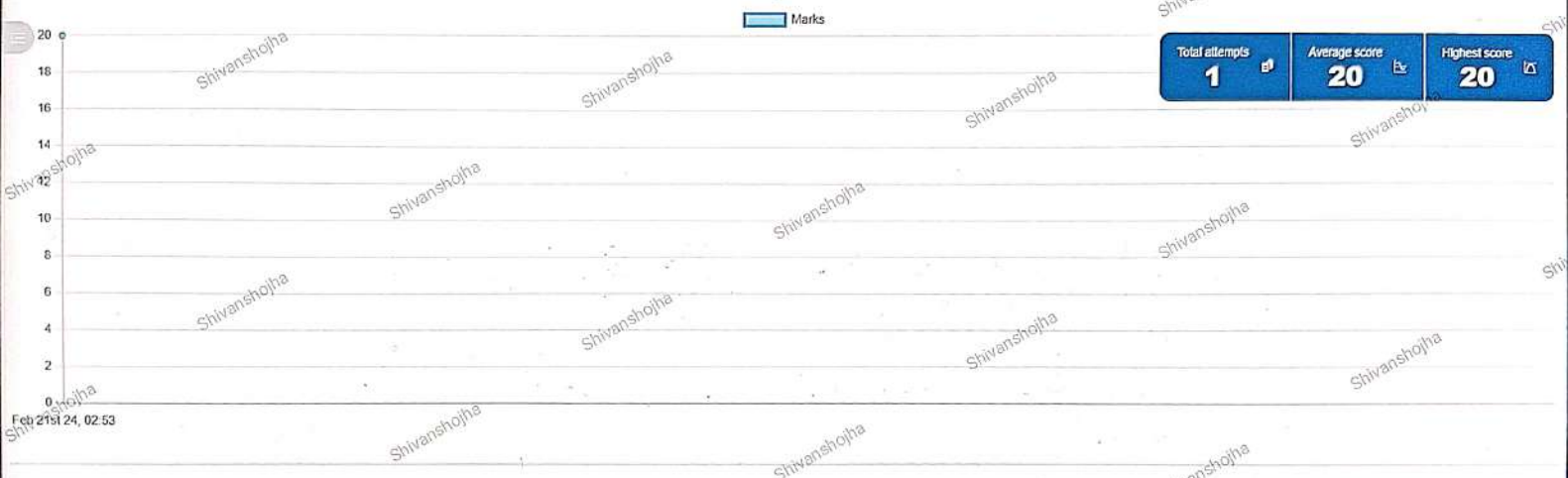
Shivanshojha

Home Learn Anywhere 12216507.st@lpu.in Support Logout

Java - Oops concepts [Unscheduled]

Try Again

2 hours 20 marks(s)



Total attempts	Average score	Highest score
1	20	20

Attempt	Test date	Results	Status	Actions
1	February 21st 2024, 2:53:31 am	20/20	Passed	Details

Shivanshojha

28.1.1. String literals and String class

14:42

A string literal consists of zero or more characters (also called Unicode code points) enclosed in double quotes. Some examples of string literals are

```
"Ganga" , "123", "", " ", "Practice makes perfect", " 24@3yApp_"
```

String literals are instances of java.lang.String class.

An instance of `String` is immutable. Meaning, once a string object is created in memory, the contents of the string cannot be changed. For example, when a string is created as given below :

```
String str = "Ganga";
```

The contents of "Ganga" cannot be changed. However, `str` can be made to point to some other string object like "Yamuna".

Write a Java program in the Editor that includes a class named **StringLiterals** with the following requirements:

1. Declare a string variable and initialize it with the value **"Himalayas"**.
2. Declare a string variable and initialize it with the value **"are beautiful"**.
3. Declare a string variable and initialize it with a **single space character**.
4. Declare a string variable and initialize it with an **empty string.**
5. Print the Statement as displayed in the test case given by concatenating the declared Strings.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

StringLite...

Submit

```
package q11144;
public class StringLiterals {
	public static void main(String args[]){
	//Write your code here...
	String a ="Himalayas";
	String b ="are beautiful.";
	//String c=" ";
	String d="";
	//String e=(a+c+b+"."+d);
	System.out.println(a+" "+b+d);
}
}
```

Terminal Test cases

< Prev Reset Submit Next

Shivanshojha

28.1.2. String class

06:48

A string object can also be created using any of the constructors of the `String` class. For example:

```
String aStr = new String("Lemuria");
char charArr[] = {'A', 'L', 'l', 'a', 'n', 't', 'i', 's'};
String anotherStr = new String(charArr);
```

Instances of `String` class are **immutable**, meaning their values cannot be changed after they are created. If any alteration has to be made to a string instance, a new string object with the required content must be created.

In the below example, you will find the correct way of comparing two strings:

```
String aString = "abc";
String bString = new String("abc");
System.out.println(aString == bString); //This prints false
System.out.println(aString.equals(bString)); //This prints true
```

In the above code, when we use `==` comparison operator, we are comparing the address stored in `aString` with the address stored in `bString`. Since `bString` is created using the constructor, it creates a new object which has similar content as that of `aString`. Since `bString` points to a new object, its address is different from the address stored in `aString`. Hence, when we compare addresses, it returns `false`.

There is a equals method in String class which is used to compare the contents stored in two string objects. This returns true if the contents are the same and false if they differ. In our example, since the contents are same, we get true as the output.

Hence, the correct way of comparing contents of two strings is to use `equals` method.

Sample Test Cases +

StringLite... Submit

```
package q11145;
import java.util.*;
public class StringLiterals {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		String str1=sc.nextLine();

//Write your code here....
	String str2=new String(str1);
		System.out.print("str1 == str2 : ");
	System.out.println(str1==str2);
	System.out.println("str1.equals(str2) : "+str1.equals(str2));
	}

	}
}
```

Terminal Test cases

28.2.1. Escape sequences

10:24

Since a string literal is enclosed in double quotes, if we want to have a literal with a double quote as part of its content, we need to escape the double quote.

The way to escape a double quote is by providing the escape character `\` before the double quote.

For example:

```
String text = "He said, \"Hi\"";
```

Escape character is also used to represent non-graphic characters like the new line(`\n`) and tab(`\t`) characters.

Your task is to write a code to print the text **She said, "It is wrongly called 'common sense' because it is rarely used".** in one line and **What a wise saying!** in the next line without any changes.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

EscapeS...

Submit

```
package q11146;
public class EscapeSequence {
    //write your code here..
    public static void main(String args[]){
        System.out.println("She said, \"It is wrongly called \'common sense\' because it is rarely used\". ");
        System.out.println("What a wise saying!");
    }
}
```

Terminal Test cases

Plots

Shivanshojha

28.2.2. Understanding Escape sequences

00:38

Fix the text in the **main** method using an escape character to produce the below output.

```
There is a single double-quote before "BLUE
```

Note: Please don't change the package name.

Sample Test Cases +

EscapeS...

Submit

```
package q11147;
public class EscapeSequence {
    public static void main(String[] args) {
        String text = "There is a single double-quote before \"BLUE";
        System.out.println(text);
    }
}
```

Terminal Test cases

Shivanshojha

28.2.3. Escape sequences

00:22

Fix the text in the **main** method using escape characters to produce the below output.

```
There are double-quotes around "Red"
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

EscapeS...

Submit

```java
package q11148;
public class EscapeSequence {
	public static void main(String[] args) {
		String text1 = "There are double-quotes around \"Red\"";
		System.out.println(text1);
	}
}
```

Plots

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

28.2.4. Escape sequences

01:21

Fix the text in the **main** method using escape characters to produce the below output.

```
insert into Employee(id, name) values("10","Meka");
insert into Employee(id, name) values("11","Reka");
```

Note: Please don't change the package name.

Sample Test Cases +

EscapeS...

Submit

```
package ql1149;
public class EscapeSequence {
    public static void main(String[] args) {
        String text1 = "insert into Employee(id, name) values(\"10\",\"Meka\");";
        String text2 = "insert into Employee(id, name) values(\"11\",\"Reka\");";
        System.out.println(text1);
        System.out.println(text2);
    }
}
```

Terminal Test cases

Shivanshojha

28.3.1. String Concatenation

01:22

Strings can be concatenated using the + operator.

For example:

```
String helloStr = "Hello " + "World";
```

If one operand in the expression is of type String and the other operand is not a String, the latter is converted to a string by automatically invoking the `toString()` method on it.

The result of string concatenation is a reference to a new String object that is created by merging the contents of the two strings.

Some examples of string concatenation are given below:

"Hello " + "World"	Hello World
"The square root of 2 is " + Math.sqrt(2)	The square root of 2 is 1.4142135623730952
3 + 2 + " monkeys"	5 monkeys
"" + 3 + 2 + " monkeys"	32 monkeys
"monkeys " + 3 + 2	monkeys 32

The + operator is syntactically left-associative, i.e. it evaluates the left side of an expression first.

Sample Test Cases +

Explorer StringCo... Submit

```
package q11150;
public class StringConcatenation {
    public static void main(String[] args) {
        System.out.println("Hello!" + " I am learning Java.");
        System.out.println("Total = " + 3 + 2);
        System.out.println("Total = " + (3 + 2));
    }
}
```

Terminal Test cases

Shivanshojha

29.1.1. String methods : length(), charAt(), toLowerCase(), toUpperCase().

04:16

The length of a string can be obtained using the method public int length() of the String class.

For Example:

```
"Pentagon".length() returns 8.
```

The characters in a string are indexed from 0, just like in an array. The method public char charAt(int index) returns the char value at the specified index. An index ranges from 0 to length() - 1. The first char value of the string is at index 0, the next at index 1, and so on, as for array indexing.

If we want to convert a string to lowercase, we should use public String toLowerCase() method. It returns a new string that has all the characters of the original string in lower case. Similarly, we have a public String toUpperCase() method for converting to upper case.

Click on the String Methods Demo button to see live execution of the above mentioned methods. Make sure to provide your custom text to see the changes.

Write a Java program that demonstrates the usage of several String methods. The program should perform the following tasks:

- Find and print the length of the string.
- Retrieve and print the 2nd character of the string.
- Convert the string to uppercase and print the result.
- Convert the string to lowercase and print the result.

Note:

- The code provided in the editor reads the input string using the nextLine() method and stores it in a

Sample Test Cases +

StringMet... Submit

```java
package q11151;
import java.util.*;
public class StringMethods {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.nextLine();
    //Write your code here...
        System.out.println("Length : "+str.length());
        System.out.println("2nd character : "+str.charAt(1));
        System.out.println("Upper case : "+str.toUpperCase());
        System.out.println("Lower case : "+str.toLowerCase());
    }
}
```

Terminal Test cases

Shivanshojha

29.1.2. String methods : length()

01:13

Create a class `TestStringMethods` with a **main** method. The method receives one command line argument. Calculate the length of the argument and print the output.

For Example:

```
Cmd Args : Independance
12
```

Note: Please don't change the package name.

Sample Test Cases +

TestStrin...

Submit

```java
package q11152;
public class TestStringMethods{
    public static void main(String args[]){
        String a = args[0];
        System.out.println(a.length());
    }
}
```

Terminal Test cases

Shivanshojha

29.1.3. String methods : charAt()

01:05

Create a class `TestStringMethods` with a **main** method. The method receives one command line argument. Print the 5th character in the argument.

For Example:

```
Cmd Args : Strange
n
```

Note: Please don't change the package name.

Sample Test Cases

TestStrin...

Submit

```
package q11153;
public class TestStringMethods{
    public static void main(String args[]){
        String str=args[0];
        System.out.println(str.charAt(4));
    }
}
```

Terminal Test cases

Shivanshojha

29.1.4. String methods :toLowerCase()

01:48

Create a class `TestStringMethods` with a **main** method. The method receives one command line argument. Convert the argument to **lowercase** and print the output.

For Example:

```
Cmd Args : KRISHNA
krishna
```

Note: Please don't change the package name.

Sample Test Cases +

TestStrin...

Submit

```java
package q11162;
public class TestStringMethods{
	public static void main(String args[]){
		String str=args[0];
		System.out.println(str.toLowerCase());
	}
}
```

Terminal Test cases

Shivanshojha

29.1.5. String methods : toUpperCase()

16:18

Create a class `TestStringMethods` with a **main** method. The method receives one command line argument. Convert the argument to **uppercase** and print the result.

For Example:

```
Cmd Args : India
INDIA
```

Note: Please don't change the package name.

Sample Test Cases +

TestStrin...

```
package ql1163;
public class TestStringMethods{
    public static void main(String[] args){
        String str= args[0];
        System.out.println(str.toUpperCase());
    }
}
```

Submit

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

29.1.6. String methods : equals(), equalsIgnoreCase(), startsWith(), endsWith()

13:26

To check if two strings have the same content or not, we can use public boolean equals(Object anObject) method. The method returns true if and only if the argument is not null and is a String object that represents the same sequence of characters as the string.

Click on the String Equals Method Demo button to see live execution of the **equals** method. Make sure to provide your custom text to see the change in its working.

To compare two strings ignoring the case, there is a public boolean equalsIgnoreCase(String aStr) method. The method returns true if and only if the argument is not null and is a String object that represents the same sequence of characters as the string, ignoring the case.

Click on the String EqualsIgnoreCase Method Demo button to see live execution of the **equalsIgnoreCase** method. Make sure to provide your custom text to see the change in its working.

To check if a string starts with a certain prefix, the method public boolean startsWith(String prefix) can be used.

Click on the String StartsWith Method Demo button to see live execution of the **startsWith** method. Make sure to provide your custom text and also change the **startsWith** text to see the change in its working.

Similarly, we can use public boolean endsWith(String suffix) to check if a string ends with the specified suffix.

Click on the String EndsWith Method Demo button to see live execution of the **endsWith** method. Make sure to provide your custom text and also change the **endsWith** text to see the change in its working.

Sample Test Cases +

StringMet... Submit

```java
package q11164;
import java.util.*;
public class StringMethods {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str1 = sc.nextLine();
        String str2 = sc.nextLine();
        //Write your code here....
        System.out.println("equals : "+str1.equals(str2));
        System.out.println("equalsIgnoreCase: "+str1.equalsIgnoreCase(str2));
        System.out.println("startsWith "+str1.startsWith("J"));
        System.out.println("endsWith "+str2.endsWith("is"));

    }
}
```

Terminal Test cases

Shivanshojha

29.1.7. String methods : equals()

01:08

Write a class `StringCompare` with a **main** method. The method receives two command line arguments and prints true if both are equal.

For Example:

```
Cmd Args : India India
true
```

Note: Please don't change the package name.

Sample Test Cases +

StringCo...

Submit

```
package q11165;
public class StringCompare{
    public static void main(String[] args){
        String s1=args[0];
        String s2=args[1];
        System.out.println(s1.equals(s2));
    }
}
```

Terminal Test cases

Shivanshojha

29.1.8. String methods : equalsIgnoreCase()

01:10

Write a class `StringCompare` with a **main** method. The method receives two command line arguments and prints true if both are equal.

Assumptions:

1. Ignoring the case of the two arguments

For Example:

```
Cmd Args : Congrats conGrats
true
```

Note: Please don't change the package name.

Sample Test Cases

StringCo...

Submit

```java
package q11166;
public class StringCompare{
	public static void main(String[] args){
		String s1=args[0];
		String s2=args[1];
		System.out.println(s1.equalsIgnoreCase(s2));
	}
}
```

Terminal Test cases

Shivanshojha

29.1.9. String methods : startsWith()

01:19

Write a class `StringCompare` witha a **main** method. The method receives one command line argument. Print **true** if the argument **starts with** the prefix **pre**, otherwise print **false**.

For Example:

```
Cmd Args : preparation
true
```

Note: Please don't change the package name.

Sample Test Cases

StringCo...

Submit

```java
package ql1167;
public class StringCompare{
    public static void main(String[] args){
        String s1=args[0];
        System.out.println(s1.startsWith("pre"));
    }
}
```

Terminal Test cases

29.1.10. String methods : endsWith()

01:10

Write a class `StringCompare` with a **main** method. The method receives one command line argument. Print **true** if the argument **ends with** the suffix **ed**, else print **false**.

For Example:

```
Cmd Args : prefixed
true
```

Note: Please don't change the package name.

Sample Test Cases +

StringCo...

Submit

```
package q11168;
public class StringCompare{
    public static void main(String[] args){
        String s1=args[0];
        System.out.println(s1.endsWith("ed"));
    }
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

29.1.11. String methods : equals() and ==

02:04

Write a class `StringCompare` with a **main** the method. The method receives two command line arguments compare the first and second arguments using **equals()** method and also by using **==** operator and print the result.

For example :

```
Cmd Arge : Ganga Ganga
The result with equals : true
The result with == : false
```

Note: Please don't change the package name.

Sample Test Cases +

StringCo...

Submit

```java
package q11169;
public class StringCompare {
	public static void main(String[] args) {
		String s1=args[0];
		String s2=args[1];
		System.out.println("The result with equals : "
+s1.equals(s2));
		System.out.println("The result with == : " + (s1==s2));
	}
}
```

Terminal Test cases

< Prev Reset Submit Next

Shivanshojha

29.1.12. String methods : length(), charAt(), toLowerCase(), toUpperCase()

09:35

The `String` class provides a method public int compareTo(String anotherString) to perform a lexicographical comparison of itself with another string passed to it.

It returns `0` (zero) if the strings are **equal**.
It returns a **positive number** `( > 0 )`, when the string appears before the string argument passed.
It returns a **negative number** `( < 0 )`, when the string appears after the string argument passed.

For example:

```
"Moon".compareTo("Moon") returns 0
"Earth".compareTo("Moon") returns -8
"Moon".compareTo("Earth") returns 8
```

Write a class `StringCompareTo` with a **main** method.

Assume the main method receives **two** string arguments. Write code to print the below output by comparing first argument and the second argument.

```
arg1 is lesser than arg2 //if first argument lexicographically appears before the second argument
arg1 is greater than arg2 //if first argument lexicographically appears after the second argument
arg1 and arg2 are equal//if first argument is equal to the second argument
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

StringCo...

Submit

```
package q11170;
public class StringCompareTo{
    public static void main(String[] args){
        String arg1=args[0];
        String arg2=args[1];
        int res=arg1.compareTo(arg2);
        if(res==0)
            System.out.println("arg1 and arg2 are equal");
        else if(res<0)
            System.out.println("arg1 is greater than arg2");
        else
            System.out.println("arg1 is lesser than arg2");
    }
}
```

Plots

Terminal Test cases

< Prev Reset Submit Next

30.1.1. String methods : IndexOf(), lastIndexOf()

11:50

Some times in text processing, we have to check if a string or a particular character is contained within another string and if so at what position. The following methods help in handling such cases.

The method public int indexOf(String str) returns the index of the first occurrence of the specified substring. Example:

```
"man and woman and child".indexOf("and") returns 4
as the index of the first occurrence of 'and' is 4 within the string.
```

If we want to search for the index of a substring after a certain position within a string, we should use the method public int indexOf(String str, int fromIndex). It returns the index of the first occurrence of the specified substring, starting the search at the specified index. Example:

```
"man and woman and child".indexOf("and", 5) returns 14
as the index of the first occurrence of 'and' after index 5 is 14.
```

If we want the index position of the last occurrence of a string within another string, the method lastIndexOf(String str) can be used. It returns the index of the last occurrence of the specified substring. Example:

```
"rats and cats and dogs".lastIndexOf("s") returns 21
as the index of the last occurrence of 's' is 21.
```

Suppose we want to search for the last occurrence of a string within another string, but we want the search to begin at a certain index and then go backwards. The method lastIndexOf(String str, int fromIndex) is useful for doing this.

Sample Test Cases +

StringMet... Submit

```java
package q11171;
public class StringMethods {
    public static void main(String[] args) {
        String str = "special char &";
        //write your code here
        System.out.println(str.indexOf("char"));
        System.out.println(str.indexOf(" "));
        System.out.println(str.lastIndexOf(" "));
        System.out.println(str.lastIndexOf("a",8));
        System.out.println(str.indexOf("sp",4));
    }}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

Home Learn Anywhere

30.1.2. String methods : IndexOf()

01:23

Write a class `StringIndexOf` with a **main** method. The method receives one command line argument. Print the **index** of the **first occurrences** of the string **CA** in the argument.

For Example:

```
Cmd Args : TCA*CX*CX*CAT
1
```

Note: Please don't change the package name.

Sample Test Cases +

StringInd...

Submit

```java
package q11172;
public class StringIndexOf{
	public static void main(String args[]){
		String s1=args[0];
		System.out.println(s1.indexOf("CA"));
	}
}
```

Terminal Test cases

Shivanshojha

30.1.3. String methods : indexOf()

01:55

Write a class `StringIndexOf` with a **main** method. The method receives one command line argument. Print the second occurrence of the string **CA** in the argument.

For Example:

```
Cmd Args : CALIFORNICA
9
```

Note: Please don't change the package name.

Sample Test Cases +

StringInd...

Submit

```
package q11173;
public class StringIndexOf{
    public static void main(String[] args){
        String s1=args[0];
        System.out.println(s1.indexOf("CA",2));
    }
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

30.1.4. String methods : lastIndexOf()

01:09

Write a class `StringLastIndexOf` with a **main** method. The method receives one command line argument. Print the **index of the last occurrence** of the character * in the argument.

For Example:

```
Cmd Args : TCA*TX*TX*
9
```

Note: Please don't change the package name.

Sample Test Cases +

StringLas...

Submit

```java
package q11174;
public class StringLastIndexOf{
	public static void main(String[] args){
		String s1=args[0];
		System.out.println(s1.lastIndexOf("*"));
	}
}
```

Terminal Test cases

Shivanshojha

Home Learn Anywhere

12216507.st@lpu.in Support Logout

30.1.5. String methods : lastIndexOf()

08:05

Write a class `StringLastIndexOf` with a **main** method. The method receives one command line argument. Print the index of the **last but one** occurrence of the string **TX** in the given argument.

For example:

```
Cmd Args : TX12TX3TX4
4
```

Hint: You may want to store the lastIndex of **TX** first into a variable. And later explore how to use the method String.lastIndexOf(String str, int fromLastIndex)

Note: Please don't change the package name.

Sample Test Cases +

StringLas...

Submit

```java
package q11175;
public class StringLastIndexOf{
	public static void main(String args[]){
		String s1=args[0];
		int index=s1.lastIndexOf("TX");
		System.out.println(s1.lastIndexOf("TX",index-1));
	}
}
```

Terminal Test cases

Shivanshojha

30.1.6. String methods : substring()

08:38

To extract a string from another string, we use public String substring(int beginIndex) method. It returns a substring that starts at the specified index and extends to the end of the string. Examples:

```
"toBeOrNot".substring(2) returns "BeOrNot"
"Summers".substring(3) returns "mers"
"emptiness".substring(9) returns "" (an empty string)
```

If **beginIndex** is negative or larger than the length of the string, the method throws `IndexOutOfBoundsException`.

If we know at what indices the substring should begin and end in another string, we can use the method public String substring(int beginIndex, int endIndex) to extract the substring. The substring begins at the specified **beginIndex** and extends up to the character at **endIndex - 1**. **Note:** The character at the **endIndex** is **not** included in the resulting substring. Examples:

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

The method throws `IndexOutOfBoundsException` if the **beginIndex** is negative, or **endIndex** is larger than the length of this String object, or **beginIndex** is larger than **endIndex**.

Click on the Substring Demo button to see live execution of the above mentioned methods. Make sure to provide your custom text and also change the **beginIndex** and **endIndex** to see the change in their working.

Write a Java program that demonstrates the use of the **substring()** method to extract substrings from a given string. The functionality of the program is described below:

- It uses the **substring()** method on the **str** string object to extract a substring starting from the given

Sample Test Cases +

Explorer SubStrin... Submit Plots

```
package q11176;
import java.util.*;
public class SubString {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.nextLine();
        String substr = sc.nextLine();
        int index = sc.nextInt();

        //write your code here...
        String s1=str.substring(index);
        int s2=str.indexOf(substr);
        System.out.println(s1);
        System.out.println(str.substring(s2));
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

30.1.7. String methods : substring()

01:14

Write a class `SubstringText` with a **main** method. The method receives one command line argument. Print a **substring** of the argument that starts from index **7**.

For Example:

```
Cmd Args : Synchronicity
nicity
```

Note: Assume that the first argument will always contain more than 7 characters.

Note: Please don't change the package name.

Sample Test Cases +

Substring...

Submit

```java
package q11177;
public class SubstringText{
	public static void main(String[] args){
		String s1=args[0];
		System.out.println(s1.substring(7));
	}
}
```

Terminal Test cases

Shivanshojha

30.1.8. String methods : substring()

01:09

Write a class `SubstringText` with a **main** method. The method receives one command line argument. Print the **substring** of the argument that starts at index **5** and ends at index **12** (should include the character at index 12).

Assumptions:

1. The argument contains more than 12 characters

For Example:

```
Cmd Args : Merry-go-round
-go-roun
```

Note: Please don't change the package name.

Sample Test Cases +

Substring...

Submit

```
package q11178;
public class SubstringText{
    public static void main(String[] args){
        String s1=args[0];
        System.out.println(s1.substring(5,13));
    }
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

30.1.9. String methods : replace(), trim()

07:17

If we want to replace every character in a string with another character, the method public String replace(char target, char replacement) can be used.

Example:

```
"An Apple".replace('A', '@') returns "@n @pple"
```

There is an equivalent method to the above one where we can use strings as arguments instead of chars.
Examples:

```
"An Apple".replace("An", "The") returns "The Apple"
```

Many times during text processing, we should make sure that a string has no preceding or trailing space characters. A typical scenario is, if we are reading a name from a file, it is possible that there are extra spaces after the name in the file. When we compare it with an expected value, the comparison fails because of the spaces.

For example, if we read "Giza " from file and compare it with "Giza" using `"Giza ".equals("Giza")`, the method returns false. Solution to this problem is to use the public String trim() method. This method returns a string whose value is this string, with any leading and trailing white spaces removed.
Examples:

```
"  Galaxy  ".trim() returns "Galaxy"
"   Galaxy".trim() returns "Galaxy"
"Galaxy   ".trim() returns "Galaxy"
"Ga la xy".trim() returns "Ga la xy"
```

Sample Test Cases +

StringMet... Submit

```java
package q11179;
public class StringMethods {
    public static void main(String[] args) {
        String str = " Six seasons  ";
        //Write your code here...
        System.out.println("length = "+str.length());
        String s1=str.trim();
        System.out.println("length after trim = "+s1.length());
        System.out.println(str.replace(" ","*"));
        System.out.println(s1.replace("Six","Four"));
    }}
```

Terminal Test cases

Prev Reset Submit Next

30.1.10. String methods : trim()

01:08

Write a class `StringTrim` with a **main** method. The method receives one command line argument, remove any **leading** or **trailing** white spaces from the argument and print the argument.

For Example:

```
Cmd Args : blank
blank
```

Note: Please don't change the package name.

Sample Test Cases

StringTri...

Submit

```java
package ql1180;
public class StringTrim{
	public static void main(String[] args){
		String s1=args[0];
		System.out.println(s1.trim());
	}
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

30.1.11. String methods : replace()

01:39

Create a class `StringReplace` with a **main** method. The method receives one command line argument , **replace** all the occurrences of character **c** with **k** and print the result.

For Example:

```
Cmd Args : Acceptance
Akkeptanke
```

Note: Please don't change the package name.

Sample Test Cases

StringRe...

Submit

```java
package q11181;
public class StringReplace{
    public static void main(String[] args){
        String s1=args[0];
        System.out.println(s1.replace("c","k"));
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

30.1.12. String methods : split()

04:22

Suppose we have a string that contains a list of comma separated items like, "Apples, Oranges, Peaches, Bananas". We want to split the string extract all the items into an array. String class provides a method for doing this, called public String[] split(String regex).

The above string can be split using split method like this :

```
"Apples, Oranges, Peaches, Bananas".split(",").
It returns an array containing the items in the list.
When we print the array, it prints the following:
Apples
 Oranges
 Peaches
 Bananas
Note the extra space before each item (except Apples).
```

The argument for the split method should be a regular expression. You will learn about using regular expressions in Java in some other section.

Write a Java program that takes user input and splits the string into substrings using the **delimiter "/"** and stores the results in an array. Then, Print each substring on a separate line.

Note:

- The code provided in the editor reads the input string using the nextLine() method and stores it in a str variable.
- Please don't change the package name.

Sample Test Cases +

StringMet... Submit

```
package q11182;
import java.util.*;
public class StringMethods {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.nextLine();

        //Write your code here...
        String[] arr=str.split("/");
        for(String i:arr){
            System.out.println(i);
        }
    }
}
```

Terminal Test cases

Prev Reset Submit Next

Shivanshojha

30.1.13. String methods : split()

03:45

Create a class `StringSplit` with a **main** method. The method receives two command line arguments. First argument is a **string**. Second argument is a **regular expression**.

A regular expression is a sequence of characters that define a search pattern.

Example:

```
+, -, *, ]
```

Split the first argument using the regular expression and print the items. Make sure there are no leading or trailing spaces in the items.

For Example:

```
Cmd Args: Rama-Lakshmana-Bharatha-Satrugna -
Rama
Lakshmana
Bharatha
Satrugna
```

Note: Please don't change the package name.

Sample Test Cases +

StringSpl...

Submit

```
package q11183;
public class StringSplit{
    public static void main(String[] args){
        String str=args[0];
        String str1=args[1];
        String[] arr=str.split(str1);
        for(String val:arr){
            val=val.trim();
            System.out.println(val);
        }
    }
}
```

Terminal Test cases

Shivanshojha

30.1.14. Program to check the given String is Palindrome or not

20:43

Create a class **PalindromeOrNot** with a **main** method. The method receives one command line argument. Check the given argument is palindrome or not.

For example:

```
Cmd Args : madam
The given string madam is a palindrome
Cmd Args : Godavari
The given string Godavari is not a palindrome
```

Note: Please don't change the package name.

Sample Test Cases +

Palindro... Submit

```
package q11184;
public class PalindromeOrNot{
	public static void main(String[] args){
		String str=args[0];
		int len=str.length();
		boolean res=true;
		for(int i=0;i<len/2;i++){
			if(str.charAt(i)!=str.charAt(len-i-1)){
				res=false;
				break;
			}
		}
		if(res)
			System.out.println("The given string "+str+" is a palindrome");
		else
			System.out.println("The given string "+str+" is not a palindrome");
	}
}
```

Terminal Test cases

Shivanshojha

31.1.1. printf style formatting in Java

04:09

Java provides support for `printf style formatting` of strings. **Printf** style formatting contains a **format string** and a number of different data type **parameters**. The **format string** contains **plain text** as well as **format specifiers**, which are special characters that format the parameters. The function produces an output string with parameters applied as per the format specifiers.

For example:

```
System.out.format("Fees = %f,  Age = %d and Name = %s", 12.95, 24, "Raj");
produces an output of : Fees = 12.950000,  Age = 24 and Name = Raj
```

In the above example,

- the format string is : `"Fees = %f, Age = %d and Name = %s"`
- the format specifiers are : `%f`, `%d` and `%s`
- the parameters are **12.95**, **24** and **"Raj"**

The `format(...)` method formatted the **parameter values** as per the **format specifiers** and produced the output string.

Java's PrintStream.format(String format, Object... args), PrintStream.printf(String format, Object... args) and String.format(String format, Object... args) methods are equivalent methods that provide this functionality.

There are many converters, flags and specifiers, which are documented in java.util.Formatter class.

For example, if you want to know about the formatting character **%n**, click on the above link to **Formatter** and go to the **formatting character conversion table** under **Conversions** section.

We will learn more about formatting in the later sections.

Sample Test Cases +

StringFor... Submit

```
package q11185;
public class StringFormatting {
    public static void main(String[] args) {
        int iVar = 432000;
        System.out.printf("Integer variable = %d\n", iVar);
        double dVar = 3.14159;
        System.out.printf("Double variable = %f\n", dVar);
        String str = "Demo";
        System.out.printf("String variable = %s\n", str);
    }
}
```

Terminal Test cases

Shivanshojha

31.1.2. printf style formatting in Java - Integer, float and String

18:01

The format specifier for integers is `%d`. For float or doubles, it is `%f`. For String, the format specifier is `%s`. To put a new line character in the string, we should use `%n`.

Some additional flags can be specified to these data types.

A number after %. Example: **%8d**	Means the output value should have a minimum of 8 characters for width. If the number is smaller, it puts spaces in the front.
Zero and a number after %. Example: **%08d**	Same as above, but instead of spaces, it puts zeros in the front.
+ Example: **%+d**	Shows a + or - sign in the output depending on the parameter value.
, Example: **%,d**	Formats the output with commas (or any other locale specific grouping characters).
- Example: **%-8d**	Left justifies the output value. Should be used in combination with a number.
. Example: **%.3f** or **%8.3f**	For displaying decimal points. Should be followed by a number which indicates the number of decimal points.

Invalid combination of the formatters will result in an error.
For example, `%.3d` is invalid because it is trying to specify 3 decimal places for integer data type.

Sample Test Cases +

Explorer

StringFor...

Submit

```
1  package q11186;
2  public class StringFormatting {
3      public static void main(String[] args) {
4          int iVar = 2435;
5
6          //write your code here...
7          System.out.printf("%6d%n%06d%n%0+6d%n%+,6d%n%,4d
   \n",iVar,iVar,iVar,iVar,iVar);
8
9      }
10 }
```

Terminal Test cases

Shivanshojha

31.1.3. printf style formatting in Java - Integer, float and String

10:09

Create a Java program called StringFormatting that contains a main method. Inside the main method, declare a float variable and initialize it with the value 3.141f. Then, print the value of the variable in the following formats:

- A simple floating-point number.
- with three decimal places.
- with two decimal places.
- with four decimal places and a total width of 8 characters.
- with four decimal places, a total width of 8 characters, and a plus sign for positive values,.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

StringFor...

Submit

```
package q11187;
public class StringFormatting {
    public static void main(String[] args) {
        float dVar = 3.141f;

        //write your code here..

        System.out.printf("%.6f%n%.3f%n%.2f%n%08.4f%n%+08.4f\n",dVar,dVar,
dVar,dVar,dVar);
    }
}
```

Terminal Test cases

Util Plots

Shivanshojha

31.1.4. printf style formatting in Java - Integer, float and String

03:57

Create a class `TestFormatter` with a **main** method. The method receives one command line argument, convert it to an integer.

Format and print the integer such that it has comma (or locale specific) delimiters, has a minimum width of 6 characters (put spaces in the front if the number is smaller), and has a sign indicator (+ or -).

[Note: You need not use **%n** at the end of your formatting text.]

Note: Please don't change the package name.

Sample Test Cases

TestForm...

Submit

```
package q11188;
public class TestFormatter{
    public static void main(String[] args){
        int num=Integer.parseInt(args[0]);
        System.out.printf("%,+6d",num);
    }
}
```

Terminal Test cases

Shivanshojha

31.1.5. printf style formatting in Java - Integer, float and String

04:40

Create a class `TestFormatter` with a **main** method. The method receives one command line argument.

Convert the argument received in the main method into a `double` value.

Format and print the `double` value such that it has :

1. comma (or locale specific) delimiters
2. a minimum width of 12 characters (put numbers in the front if the number is smaller)
3. only 3 decimal points
4. a sign indicator (+ or -)
5. instead of spaces put zeros(0's) to the left if the number is smaller

[Note: You need not use **%n** at the end of your formatting text.]

Note: Please don't change the package name.

Sample Test Cases +

Explorer

TestForm...

Submit

```
package q11189;
public class TestFormatter{
	public static void main(String[] args){
		double num=Double.parseDouble(args[0]);
		System.out.printf("%+,012.3f",num);
	}
}
```

Terminal Test cases

Shivanshojha

Home Learn Anywhere 12216507.st@lpu.in Support Logout

Java - String class [Unscheduled]

Try Again

2 hours 25 marks(s)

Total attempts	Average score	Highest score
1	25	25

Attempt	Test date	Results	Status	Actions
1	February 22nd 2024, 2:19:53 am	25/25	Passed	Details

Shivanshojha

32.1.1. String concatenation

01:49

Write a class `Greeting` with a main method. The method receives one command line argument and prints the output as shown below.

For example:

```
Cmd Args : Ram
Hello Ram!
```

Note: Make sure to use `println` and not `print` method. If you want to use `printf`, ensure to print a `\n` character at the end.

Note: Please don't change the package name.

Sample Test Cases +

Greeting.j...

Submit

```java
package q11191;
public class Greeting{
	public static void main(String[] args){
		String str=args[0];
		System.out.println("Hello "+str+"!");
	}
}
```

Terminal Test cases

Shivanshojha

32.1.2. String concatenation

01:45

Write a class `PrefixCheck` with a **main** method. The method receives one command line argument. If the argument starts with **en** print it as it is, else prepend **en** and print the output.

These are examples for your understanding :

```
Cmd Args : bye
enbye
```

```
Cmd Args : entertainment
entertainment
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

PrefixChe...

Submit

```java
package q11192;
public class PrefixCheck{
    public static void main(String[] args){
        String str=args[0];
        if(str.startsWith("en"))
            System.out.println(str);
        else
            System.out.println("en"+str);
    }
}
```

Terminal Test cases

All Plots

Shivanshojha

32.1.3. String concatenation

02:06

Write a class `ExpandAString` with a **main** method. The program receives two command line arguments. The first argument is a string and the second argument is a number. Print the first argument as many number of times as the second argument.

For Example:

```
Cmd Args : Ganga 3
GangaGangaGanga
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

ExpandA...

Submit

```java
package q11193;
public class ExpandAString {
	public static void main(String[] args){
		String str=args[0];
		int n=Integer.parseInt(args[1]);
		String s1="";
		for(int i=0;i<n;i++){
			s1+=str;
		}
		System.out.println(s1);
	}
}
```

Terminal Test cases

Shivanshojha

32.1.4. String parsing exercise

01:46

Write a class `TestSubstring` with a **main** method. The method receives one command line argument and print the output formed by the first three characters of the argument.

For example:

```
Cmd Args : Important
Imp
```

Note: Please don't change the package name.

Sample Test Cases +

TestSubs...

Submit

```
package q11194;
public class TestSubstring{
    public static void main(String[] args){
        String str=args[0];
        System.out.println(str.substring(0,3));
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

32.1.5. String parsing exercise

01:56

Write a class `TestSubstring` with a **main** method. The method receives one command line argument and print the **first half** of the argument.

Here is an example:

```
Cmd Args : Eight
Ei
```

Note: Please don't change the package name.

Sample Test Cases +

TestSubs... Submit

```java
package q11195;
public class TestSubstring {
	public static void main(String[] args){
		String str=args[0];
		int len=str.length();
		System.out.println(str.substring(0,len/2));
	}
}
```

Terminal Test cases

Shivanshojha

32.1.6. String parsing exercise

01:33

Write a class `TestSubstring` with a **main** method. The method receives one command line argument and prints the output by removing the first and last characters of the argument.

For example:

```
Cmd Args : January
anuar
```

Note: Please don't change the package name.

Sample Test Cases +

TestSubs...

Submit

```java
package q11196;
public class TestSubstring{
    public static void main(String[] args){
        String str=args[0];
        int len=str.length();
        System.out.println(str.substring(1,len-1));
    }
}
```

Terminal Test cases

Shivanshojha

32.1.7. String parsing exercise

05:31

Write a class `StringStairs` with a **main** method. The method receives one command line argument and prints the first char in one line, then the first two chars in the next line, etc.

For example:

```
Cmd Args : Delhi
D
De
Del
Delh
Delhi
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

StringSta...

Submit

```java
package q11197;
public class StringStairs{
	public static void main(String[] args){
		String str=args[0];
		int len=str.length();
		int j=1;
		for(int i=0;i<len;i++){
			System.out.println(str.substring(0,j));
			j++;
		}
	}
}
```

Terminal Test cases

Shivanshojha

32.1.8. String parsing exercise

08:09

Write a class `RemoveChar` with a **main** method. The method receives one command line argument and prints the output by removing the 3rd character from the argument.

Assumptions:

1. Length of the argument is greater than or equal to 3 characters

For example:

```
Cmd Args : Victory
Vitory
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

RemoveC...

Submit

```
package q11198;
public class RemoveChar{
    public static void main(String args[]){
        String str=args[0];
        int len=str.length();
        String s1="";
        for(int i=0;i<len;i++){
            if(i==2)
                continue;
            else
                s1+=str.charAt(i);
        }
        System.out.println(s1);
    }
}
```

Terminal Test cases

Shivanshojha

32.1.9. String parsing

02:45

Write a class `ExtractTag` with a **main** method. The method receives one command line argument in between tags like **[ ]**, extract the argument between the tags and print the output.

For Example:

```
Cmd Args : [Independent]
Independent
```

Note: Please don't change the package name.

Sample Test Cases +

ExtractTa...

Submit

```java
package q11199;
public class ExtractTag{
    public static void main(String[] args){
        String str=args[0];
        int len=str.length();
        System.out.println(str.substring(1,len-1));
    }
}
```

Terminal Test cases

Shivanshojha

32.1.10. String parsing

02:39

Write a class `AttachTag` with a **main** mehod. The method receives two command line arguments make the **second argument** as a special tag and attach before and after the **first argument** and print the value.

For example:

```
Cmd Args : Delhi New
[New]Delhi[/New]
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

AttachTa...

Submit

```
package q11200;
public class AttachTag{
    public static void main(String[] args){
        String str=args[0];
        String strl=args[1];
        System.out.println("["+strl+"]"+str+"[/"+strl+"]");
    }
}
```

Terminal Test cases

32.1.11. String parsing

04:32

Write a class `PutInsideTag` with a **main** method. The method receives two command line arguments. First one is a tag like '[]' or '{{()}}' etc. The second argument is a string. Write logic to print a new word where the second argument is kept exactly in the middle of the tag.

For example:

```
Cmd Args : {{{()}}} Hyderabad
{{{{Hyderabad}}}}
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

PutInside...

Submit

```java
package q11201;
public class PutInsideTag{
    public static void main(String[] args){
        String str1=args[0];
        String str2=args[1];
        int len=str1.length();
        String str3=str1.substring(0,len/2);
        String str4=str1.substring(len/2);
        System.out.println(str3+str2+str4);
    }
}
```

Terminal Test cases

Shivanshojha

32.1.12. String parsing

17:04

Write a class `ReverseString` with a **main** method. The method receives one command line argument. Write a logic to **reverse** it and print the output.

For Example:

```
Cmd Args : Ram lakshman
namhskal maR
```

Note: Please don't change the package name.

Sample Test Cases +

Explorer

ReverseS...

Submit

```java
package q11202;
public class ReverseString{
	public static void main(String[] args){
		String str=args[0];
		String str1="";
		int len=str.length();
		char ch;
		for(int i=0;i<len;i++){
			ch=str.charAt(len-i-1);
			str1+=ch;
		}
		System.out.println(str1);
	}
}
```

Terminal Test cases

Plots

Shivanshojha

32.1.13. String parsing

02:20

Write a class `MiddleTwoChars` with a **main** method. The method receives one command line argument extract the **middle two characters** from the argument and print the output.

Assumptions:

1. The string has even number of characters

For example:

```
Cmd Args : java
av
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

MiddleTw...

Submit

```java
package q11203;
public class MiddleTwoChars {
	public static void main(String[] args){
		String str=args[0];
		int len=str.length();
		String str1=str.substring((len/2)-1,(len/2)+1);
		System.out.println(str1);
	}
}
```

Terminal Test cases

Shivanshojha

32.1.14. String parsing

01:24

Write a class `EndsWith` with a **main** method. The method receives one command line argument. Print **true** if the argument ends with **bad**, else print **false**.

For Example:

```
Cmd Args : Hyderabad
true
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

EndsWith...

Submit

```
package q11204;
public class EndsWith{
    public static void main(String args[]){
        String str=args[0];
        System.out.println(str.endsWith("bad"));
    }
}
```

Terminal Test cases

Shivanshojha

32.1.15. String parsing

02:47

Write a class `EndsWith` with a **main** method. The method receives two command line arguments. Print **true** if the first argument ends with the second argument (ignoring case), else print **false**.

Example:

```
Cmd Args : Godavari ri
true
```

[**Hint:** Since we have to verify ignoring the case, you may want to first convert both either to lower or upper case before comparing.]

Note: Please don't change the package name.

Sample Test Cases +

EndsWith... Submit

```java
package q11205;
public class EndsWith {
	public static void main(String[] args){
		String str=args[0];
		String str1=args[1];
		String str2=str.toUpperCase();
		String str3=str1.toUpperCase();
		System.out.println(str2.endsWith(str3));
	}
}
```

Terminal Test cases

Shivanshojha

32.1.16. String parsing

05:00

Write a class `TestString` with a **main** method. The method receives one command line argument. Create a new string by concatenating the **first three** and **last three** characters of the argument and print the output.

Assumptions:

1. String length is at least **six**

For Example:

```
Cmd Args : Hyderabad
Hydbad
```

Note: Please don't change the package name.

Sample Test Cases +

TestStrin...

Submit

```
package q11206;
public class TestString{
    public static void main(String[] args){
        String str=args[0];
        String str1=str.substring(0,3);
        String str2=str.substring(str.length()-3);
        System.out.println(str1+str2);
    }
}
```

Terminal Test cases

Shivanshojha

32.1.17. String parsing

04:34

Write a class `TestString` with a **main** method. The method receives one command line argument. Create a string by extracting the **first** character and appending it to the **end** of the argument , then append aa at the end.

For example:

```
Cmd Args : Europe
uropeEaa
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

TestStrin...

Submit

```java
package ql1207;
public class TestString{
	public static void main(String[] args){
		String str=args[0];
		String str1=str.substring(1);
		char ch=str.charAt(0);
		System.out.println(str1+ch+"aa");
	}
}
```

Terminal Test cases

Shivanshojha

32.1.18. String parsing

14:42

Write a class `MiddleChar` with a **main** method. The method receives one command line argument. Print the **middle** character of the argument.

Assumptions:

1. The length of the string is odd

Example:

```
Cmd Args : Three
r
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

MiddleCh...

Submit

```java
package q11208;
public class MiddleChar{
    public static void main(String[] args){
        String str=args[0];
        int len=str.length();
        System.out.println(str.charAt(len/2));
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

32.1.19. String parsing

04:54

Write a class `CheckChars` with a **main** method. The method receives one command line argument and prints **true** if the argument has the characters **w**, **e** and **b**.

For Example:

```
Cmd Args : westbengal
true
```

Note: Please don't change the package name.

Sample Test Cases +

CheckCh... Submit

```java
package q11209;
public class CheckChars{
    public static void main(String[] args){
        String str=args[0];
        boolean res=str.contains("w")&&str.contains("e")&&str.contains("b");
        System.out.println(res);
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

Shivanshojha

32.1.20. String parsing

04:10

Write a class `AppendChars` with a **main** method. The method receives one commmand line argument. If the argument length is less than 10, append '*' at the end to make the length 10.

For example:

```
Cmd Args : Smile
Smile*****
```

Note: Please don't change the package name.

Sample Test Cases

AppendC...

Submit

```java
package q11210;
public class AppendChars {
	public static void main(String[] args){
		String strargs[0];
		int len=str.length();
		String str1=str;
		for(int i=len;i<10;i++){
			str1+="*";
		}
		System.out.println(str1);
	}
}
```

Terminal Test cases

Shivanshojha

32.1.21. String parsing

01:38

Write a class `RemoveChars` with a **main** method. The method receives one command line argument and prints the output by removing all x characters from the argument.

For example:

```
Cmd Args : prefix
prefi
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

RemoveC...

Submit

```
package q11211;
public class RemoveChars{
    public static void main(String[] args){
        String str=args[0];
        System.out.println(str.replace("x",""));
    }
}
```

Terminal Test cases

Shivanshojha

32.1.22. String parsing

04:45

Write a class `RemoveSuffix` with a **main** method. The method receives one command line argument. If the argument has the **same prefix** and **suffixes** up to `3` characters, remove the **suffix** and print the argument.

Example:

```
Cmd Args : systemsys
system
```

Note: Please don't change the package name.

Sample Test Cases +

RemoveS...

Submit

```java
package q11212;
public class RemoveSuffix {
	public static void main(String[] args){
		String str=args[0];
		String str1=str.substring(0,3);
		String str2=str.substring(str.length()-3);
		if(str1.equals(str2))
			System.out.println(str.substring(0,str.length()-3));
		else
			System.out.println(str);
	}
}
```

Terminal Test cases

Shivanshojha

32.1.22. String parsing

04:45

Write a class `RemoveSuffix` with a **main** method. The method receives one command line argument. If the argument has the **same prefix** and **suffixes** up to `3` characters, remove the **suffix** and print the argument.

Example:

```
Cmd Args : systemsys
system
```

Note: Please don't change the package name.

Sample Test Cases +

RemoveS...

Submit

```java
package q11212;
public class RemoveSuffix {
	public static void main(String[] args){
		String str=args[0];
		String str1=str.substring(0,3);
		String str2=str.substring(str.length()-3);
		if(str1.equals(str2))
			System.out.println(str.substring(0,str.length()-3));
		else
			System.out.println(str);
	}
}
```

Terminal Test cases

< Prev Reset Submit Next >

32.1.23. String parsing

24:49

Write a program in a class `RemovePrefix` with a **main** method. The program receives one command line argument.

The program should remove the **first two characters** from the argument and print the output, except in one condition. The program should skip removal of **x** or **y** if it encounters them in the first two positions.

These are some examples for your understanding:

Sample Input Output 1

```
Cmd Args : xyz
xyz
```

Sample Input Output 2

```
Cmd Args : abTree
Tree
```

Sample Input Output 3

```
Cmd Args : ayFlower
yFlower
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

RemoveP... Submit

```java
package q11213;
public class RemovePrefix {
    public static void main(String[] args){
        String str=args[0];
        char a=str.charAt(0);
        char b=str.charAt(1);
        if((a=='x' && b=='y') ||(b=='x' && a=='y')){
            System.out.println(str);
        }
        else if(b=='x'||b=='y')
            System.out.println(str.substring(1));
        else if(a=='x'||a=='y')
            System.out.println(str.substring(0,1)+str.substring(2));
        else
            System.out.println(str.substring(2));
    }
}
```

Terminal Test cases

Shivanshojha

32.1.24. String parsing

04:01

Write a class `CountChars` with a **main** method. The method receives one command line argument and prints the number of **o's** (it is not zero, it is the alphabet 'o') present in the argument.

Example:

```
Cmd Args : ozone
2
```

Note: Make sure to use `println` and not `print` method.

Note: Please don't change the package name.

Sample Test Cases +

CountCh...

Submit

```java
package q11214;
public class CountChars{
    public static void main(String[] args){
        String str=args[0];
        int len=str.length();
        int count=0;
        for(int i=0;i<len;i++){
            if(str.charAt(i)=='o')
                count++;
        }
        System.out.println(count);
    }
}
```

Terminal Test cases

Shivanshojha

32.1.25. Write a Java program to Delete and Remove characters using StringBuff...

08:01

Write a class `JavaStringBufferDelete` with a **main** method to delete characters from a string using StringBuffer class.

Follow the given instructions.

1. Consider a string "Hello India" and delete **0** to **6** characters in that and print the result.
2. Consider another string "Hello World", delete characters from position **0** to **length** of the entire string and print the result.
3. Consider another string "Hello Java", remove 0th character and then print the result.

Note: Please don't change the package name.

Sample Test Cases +

Explorer JavaStrin... Submit

```java
package q11215;
public class JavaStringBufferDelete{
    public static void main(String args[]){
        String str="Hello India";
        String str1="Hello World";
        String str2="Hello Java";
        System.out.println(str.substring(6));
        System.out.println("");
        System.out.println(str2.substring(1));
    }
}
```

Terminal Test cases

Shivanshojha

32.1.25. Write a Java program to Delete and Remove characters using StringBuff...

08:01

Write a class `JavaStringBufferDelete` with a **main** method to delete characters from a string using StringBuffer class.

Follow the given instructions.

1. Consider a string "Hello India" and delete **0** to **6** characters in that and print the result.
2. Consider another string "Hello World", delete characters from position **0** to **length** of the entire string and print the result.
3. Consider another string "Hello Java", remove 0th character and then print the result.

Note: Please don't change the package name.

Sample Test Cases +

Explorer

JavaStrin...

Submit

```
package q11215;
public class JavaStringBufferDelete{
    public static void main(String args[]){
        String str="Hello India";
        String str1="Hello World";
        String str2="Hello Java";
        System.out.println(str.substring(6));
        System.out.println("");
        System.out.println(str2.substring(1));
    }
}
```

Terminal Test cases

Shivanshojha

32.1.26. Choose correct answers

05:41

Given :

```
public static void main(String[] args) {
    String str = "null";
    if (str == null) {
        System.out.println("null");
    } else (str.length() == 0) {
        System.out.println("zero");
    } else {
        System.out.println("some");
    }
}
```

- [ ] null
- [ ] zero
- [ ] some
- [x] Compilation fails.
- [ ] An exception is thrown at runtime.

Shivanshojha

33.1.1. Understanding String and StringBuilder

09:22

In Java, we have learnt that `String` class represents an immutable sequence of characters.

For example:

```
String text = "I am back!";
text = "I am back, again!";
```

In the above code, you will notice that the reference `text` can be assigned to any other String literal but the previous String literal itself cannot be modified.

It means that once a String object is created, the contents of that object (meaning the sequence of characters in that object) cannot be modified.

Such classes whose objects cannot be modified are called **immutable classes**.

A class becomes **immutable** when it does not provide any methods to manipulate the state information stored in its fields.

The `String` class does not have even a single method which can manipulate or change the sequence of characters it represents. The methods like substring, replace, toUpperCase, toLowerCase etc., do not change the contents of the existing string, instead these methods create new string objects with the modified contents and return their references.

The StringBuilder class represents a mutable sequence of characters. Unlike the String class, the StringBuilder class provides methods to change the sequence of characters it represents.

Below are some of the most commonly used methods in `StringBuilder` class which help to

Sample Test Cases +

StringBui... Submit

```
package q11307;
import java.util.*;
public class StringBuilderDemo {
    public static void main(String[] args) {
        //Create a StringBuilder object and initialize it with the string "Ganga".
        StringBuilder abc=new StringBuilder("Ganga");
        Scanner sc = new Scanner(System.in);
        String str = sc.nextLine();
        /*Append the user-provided string to the StringBuilder object and Print it.
        Delete the first two characters from the StringBuilder object and Print it.*/
        System.out.println(abc.append(str));
        System.out.println(abc.delete(0,2));
    }
}
```

Terminal Test cases

33.1.2. Introduction to StringBuilder Class

02:17

StringBuilder objects are like String objects, except that they can be modified, means internally these objects are treated like variable-length arrays that contain a sequence of characters. At any point, the length and content of the sequence can be changed through method invocations.

String builders offer certain advantages as they offer better performance and simpler code when compared to strings. For example, if you need to concatenate a large number of strings, appending to a StringBuilder object is more efficient.

Length and Capacity: The StringBuilder class, like the String class, has a method length() that returns the length of the character sequence in the builder.

Unlike strings, every string builder also has a capacity, the number of character spaces that have been allocated. The capacity, which is returned by the capacity() method, is always greater than or equal to the length (usually greater than) and will automatically expand as necessary to accommodate additions to the string builder.

StringBuilder Constructors:

- **StringBuilder()**: Creates an empty string builder with a capacity of 16 (16 empty elements).
- **StringBuilder(CharSequence cs)**: Constructs a string builder containing the same characters as the specified CharSequence, plus an extra 16 empty elements trailing the CharSequence.
- **StringBuilder(int initCapacity)**: Creates an empty string builder with the specified initial capacity.
- **StringBuilder(String s)**: Creates a string builder whose value is initialized by the specified string, plus an extra 16 empty elements trailing the string

- [x] Initial capacity of string builder is 16
- [x] String builders are mutable.
- [ ] Strings provide better performance than string builders.

Shivanshojha

33.1.3. StringBuilder Operations

00:16

The principal operations on a StringBuilder that are not available in String are the **append()** and **insert()** methods, which are overloaded so as to accept data of any type. Each converts its argument to a string and then appends or inserts the characters of that string to the character sequence in the string builder.

The append method always adds the characters at the end of the existing character sequence, while the insert method adds the characters at a specified point.

There are a number of methods of the StringBuilder class. These are few examples

- **StringBuilder append(Object obj)** Appends the argument to this string builder. The data is converted to a string before the append operation takes place.
- **StringBuilder delete(int start, int end)** This method deletes the subsequence from start to end-1 (inclusive) in the StringBuilder's char sequence
- **StringBuilder insert(int offset, int i)** Inserts the second argument into the string builder. The first integer argument indicates the index before which the data is to be inserted. The data is converted to a string before the insert operation takes place.

StringBuilder reverse(): Reverses the sequence of characters in this string builder. **String toString()**: Returns a string that contains the character sequence in the builder.

- [ ] append() method always adds the character at the beginning.
- [x] append() and insert() operations are not available in strings
- [x] insert() method adds the character at the specified index

Shivanshojha

33.1.4. Problem Solving

00:24

Given

```
public class Main {
        public static void main(String args[]) {
            StringBuilder sb = new StringBuilder("Hi! Good Morning.");
            System.out.println(sb.length());
        }
}
```

Choose the correct output for the above program form the below options.

- [x] 17
- [ ] 16
- [ ] 15
- [ ] 18

Shivanshojha

33.1.5. Problem Solving

00:27

Given

```
public class Main {
        public static void main(String args[]) {
            StringBuilder sb = new StringBuilder("Hi! Good Morning.");


    }

}
```

Write an expression that refers to the letter M in the string referred to by sb. Choose the correct option from the below.

- [ ] sb.charAt(8)
- [x] sb.charAt(9)
- [ ] sb.charAt(5)
- [ ] sb.charAt(11)

Shivanshojha

33.1.6. Problem Solving

12:21

The program has a class `Example` with the main method. The program takes input from the command line argument. Print the output by appending all the capital letters in the input.

Sample Input and Output:

```
Cmd Args : HYderaBad
The result is: HYB
```

Sample Test Cases +

Example.j...

Submit

```java
// Type Content here...
package q24212;
public class Example{
    public static void main(String[] args){
        String str=args[0];
        StringBuilder strl=new StringBuilder();
        for(int i=0;i<str.length();i++){
            char c=str.charAt(i);
            if(Character.isUpperCase(c))
                strl.append(c);
        }
        System.out.println("The result is: "+strl.toString());
    }
}
```

Terminal Test cases

Shivanshojha

33.1.7. Problem Solving

06:10

The below code is used to understand the difference between `String` and `StringBuilder` objects. When we try to concatenate two strings using string operations a new object is created without changing the old one. In `StringBuilder` existing object is modified. In the below program this can be illustrated by comparing Hash Code for `String` object after every concat operation. Fill the missing code in the below program and observe the output.

Sample Input and Output:

```
In Strings before concatenation Hash Code is: 2081
In Strings after concatenation Hash Code is: 64578
In StringBuilder before concatenation Hash Code is: 321001045
In StringBuilder after concatenation Hash Code is: 321001045
```

Explorer

StringBui...

Submit

Plots

```java
package q24216;
public class StringBuilderDemo {
    public static void main(String args[]) {
        String s = new String("AB");
        System.out.print("In Strings before concatenation Hash Code is: ");
        System.out.println(s.hashCode());
        s += "C";
        // print hash code after concatenating
        System.out.println("In Strings after concatination Hash Code is: ");
        StringBuilder sb = new StringBuilder("AB");
        // print hash code before concatenating
        System.out.println("In StringBuilder before concatination Hash Code is: ");
        // add string C to AB
        System.out.println(sb.append("C"));
        // print hash code after concatenating
        System.out.println("In StringBuilder after concatination Hash Code is: ");
        // and observe the output
        System.out.println(sb.hashCode());
    }
}
```

Terminal Test cases

Shivanshojha

33.1.8. Problem Solving

00:33

Given

```
public class StringBuilderDemo {
    public static void main(String args[]) {
        String s1 = new String("ABC");
        String s2 = new String("ABC");
        System.out.println(s1.equals(s2));
        StringBuilder sb1 = new StringBuilder("ABC");
        StringBuilder sb2 = new StringBuilder("ABC");
        System.out.println(sb1.equals(sb2));
    }
}
```

What will be the output for the above program. Choose the correct option form the below.

- true true
- false true
- true false
- false false

Shivanshojha

33.1.9. Problem Solving

00:18

Given

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

What will be the output for the above program. Choose the correct option form the below.

- [ ] Hello World
- [x] Hello
- [ ] World
- [ ] hello

Shivanshojha

33.1.10. Problem Solving

00:14

Given

```
public class StringBuilderDemo {
    public static void main(String args[]) {
    StringBuilder sb = new StringBuilder("Hello ");
        sb.append("World");
        System.out.println(sb);
    }
}
```

What will be the output for the above program. Choose the correct option form the below.

- [ ] HelloWorld
- [x] Hello World
- [ ] World
- [ ] Hello

Shivanshojha

33.2.1. Understanding String concatenation

01:40

The Java compiler apart from compiling the source code into **.class** files, at times also performs certain optimizations.
For example, the below code

```
String text1 = "I ";
String text2 = "AM ";
String text3 = "THAT ";
String text4 = "I AM";
String fullText = text1 + text2 + text3 + text4;
```

is optimized by the Java compiler as

```
String text1 = "I ";
String text2 = "AM ";
String text3 = "THAT ";
String text4 = "I AM";
String fullText = new StringBuilder().append(text1).append(text2).append(text3).append(text4).toStr
```

By doing this the compiler ensured that during the runtime, a single `StringBuilder` object is created in memory and used to concatenate all the four `String` objects.

However when we perform concatenation in loops, the optimization may not be as efficient. For example:

```
String finalText = "";
for (String text : reallyBigStringArr) {
    finalText = finalText + text;
}
```

will be optimized by the compiler as below code

Sample Test Cases +

Explorer

Concaten...

Submit

```
package q11308;
public class ConcatenationDemo {
    public static void main(String[] args) {
        String[] wordsArr = {"I", "AM", "THAT", "I AM"};
        //complete the code...
        StringBuilder sb=new StringBuilder();
        for(String val:wordsArr){
            sb.append(val).append(" ");
        }
        String theRealMe = sb.toString();
        System.out.println("theRealMe = " + theRealMe);
    }
}
```

Plots

Terminal Test cases

Shivanshojha

33.3.1. Introduction to StringBuffer

01:01

The StringBuffer class is a thread-safe, mutable sequence of characters.These are different types of string buffer constructors. They are

- **public StringBuffer()**: Constructs a string buffer with no characters in it and an initial capacity of 16 characters.
- **public StringBuffer(int capacity)**: Constructs a string buffer with no characters in it and the specified initial capacity. It throws `NegativeArraySizeException` if the capacity argument is less than 0
- **public StringBuffer(String str)**: Constructs a string buffer initialized to the contents of the specified string. The initial capacity of the string buffer is 16 plus the length of the string argument. It throws `NullPointerException` if **str** is null
- **public StringBuffer(CharSequence seq)**: Constructs a string buffer that contains the same characters as the specified CharSequence. The initial capacity of the string buffer is 16 plus the length of the CharSequence argument. If the length of the specified CharSequence is less than or equal to zero, then an empty buffer of capacity 16 is returned. It throws `NullPointerException` if **seq** is null

- [x] Initial capacity of string buffer constructor is 16 characters.
- [ ] public StringBuffer(int capacity), if capacity is less than 0 it throws NullPointerException.
- [x] public StringBuffer(CharSequence seq), if seq is null it throws NullPointerExceotion.

Shivanshojha

33.3.2. Problem Solving

04:02

The below program explains different StringBuffer constructors. Follow the comments given below and write the missing code.

The below program has a class `StringbufferExample` with main method. The program takes input from the command line arguments. Print the output as follows.

Sample Input and Output:

```
Cmd Args : Hello World
Initial capacity is: 16
Capacity after passing parameter is: 27
Creating a StringBuffer object with the given capacity: 50
```

Sample Test Cases +

Stringbuf... Submit

```java
package q24215;
public class StringbufferExample {
	public static void main (String args[]) {
		// create instance of StringBuffer
		// find the initial capacity
		//find the capactiy after passing a parameter args[0] using command line argument
		// find the capatity by intializing capatity to 50
		StringBuffer sb=new StringBuffer();
		System.out.println("Initial capacity is: "+sb.capacity());
		sb=new StringBuffer(args[0]);
		System.out.println("Capacity after passing parameter is: "+sb.capacity());
		sb=new StringBuffer(50);
		System.out.println("Creating a StringBuffer object with the given capacity: "+sb.capacity());
	}
}
```

Terminal Test cases

34.1.1. Fundamentals of Enumeration

06:37

Enumeration means a list of named constant. In Java, enumeration defines a class type.

An enum is defined similarly to how a `class` would be defined. It uses the `enum` instead of the class keyword. The constant values are listed in the enum's body. It is common to write the constants in all uppercase letters to make it easier to recognize them.
This is an example of how to define an enum.

```
enum Subject {
    JAVA, CPP, C, DBMS
}
```

1. The elements inside the **Subject** enum are called **enumeration constants**
2. By default each enumeration constant is public, static and final.
3. Variables of Enumeration can be defined directly without using new keyword. Ex Subject = sub1;
4. Variables of Enumeration type can have only enumeration constants as value. We define a variable in enum as **enum_variable = enum_type.enum_constant;**. This can be done as follows. From the above example we can define variable as

```
sub1 = Subject.JAVA;
```

5. Enum declaration can be done either outside a Class or inside a Class. But, we cannot declare Enum inside the method
6. Enum can have fields, constructors and methods
7. Enum may implement many interfaces but cannot extend any class because it internally extends java.lang.Enum, so it can't extend other classes

In the next lesson we will learn how to Implement enums with switch case.

- [x] Enumeration means list of named constant
- [ ] An enum can be defined using class keyword
- [ ] Enumeration constants are private by default.
- [ ] Enum can extend class
- [x] Enum internally extends a class called java.lang.Enum

Shivanshojha

34.1.2. Understanding enum types

01:34

In **Java**, along with **primitives**, **classes**, **interfaces** and **abstract classes**, there is yet another type called `enum`.

An **enum** is essentially a set of named values called as members or elements. Like a class, an **enum** can also have fields and methods.

For example, let us consider the four directions East, West, North and South. We can represent them as primitive `int` constants or as `String` literals. Below example uses `int` constants:

```
final int EAST = 1;
final int WEST = 2;
final int NORTH = 3;
final int SOUTH = 4;
int direction = SOUTH;
switch (direction) {
    case EAST:
        System.out.println("East stands for Light!");
        break;
    case WEST:
        System.out.println("West stands for Strength!");
        break;
    case NORTH:
        System.out.println("North stands for Silence!");
        break;
    case SOUTH:
        System.out.println("South stands for Love!");
        break;
    default:
        System.out.println("Unknown direction value : " + direction);
}
```

Sample Test Cases +

Explorer

EnumDe...

Submit

```
package q11288;
public class EnumDemo {
    public static void main(String[] args) {
        System.out.println(printDirectionMessage(Direction.EAST));
        System.out.println(printDirectionMessage(Direction.WEST));
        System.out.println(printDirectionMessage(Direction.NORTH));
        System.out.println(printDirectionMessage(Direction.SOUTH));
    }
    public static String printDirectionMessage(Direction direction) {
        switch (direction) {
            case EAST:
                return "East stands for Light!";
            case WEST:
                return "West stands for Strength!";
            case NORTH:
                return "North stands for Silence!";
            case SOUTH:
                return "South stands for Love!";
        }
        return "Unknown direction. This case will never occur when we use enums.";
    }
}
enum Direction {
    EAST, WEST, NORTH, SOUTH;
}
```

Terminal Test cases

Shivanshojha

34.1.3. values() method in enum

01:12

values():

We can obtain an array of all the possible values of a Java enum type by calling its static method **values()**. All enum types get a static values() method automatically by the Java compiler. Here is an example of iterating all values of an enum:

```
enum Subject{
JAVA, C, PYTHON
}
```

Iteration can be done for the above enum class using the method value() can be done by

```
for(Subject sub : Subject.values()) {
    System.out.println(sub);
}
```

Write the missing code in the below program.

Sample Test Cases +

Explorer

Test.java

Submit

```
package q24199;
public class Test {
    public static void main(String args[]) {

        // Iterate over the elements in enum and print them
        for(Languages val:Languages.values()){
            System.out.println(val);
        }
    }
}
enum Languages {
    JAVA,
    PYTHON,
    C,
    CPP,
    DBMS
}
```

Terminal Test cases

Shivanshojha

34.1.4. valueOf() method in enum

04:14

valueOf():

The valueOf() method can be used to obtain an instance of the enum class for a given String value. Here is an example.

```
enum Subject{
JAVA, C, PYTHON
}
```

Iteration can be done for the above enum class using the method value() can be done by

```
Subject sub = Subject.valueOf("JAVA");
}
```

Write the missing code in the below program.

Sample Test Cases +

Test.java

Submit

```
package q24200;
public class Test {
    public static void main(String args[]) {

        // print the instances of all enumeration constants in enum class Languages
        System.out.println(Languages.valueOf("JAVA"));
        System.out.println(Languages.valueOf("PYTHON"));
        System.out.println(Languages.valueOf("C"));
        System.out.println(Languages.valueOf("CPP"));
        System.out.println(Languages.valueOf("DBMS"));

    }
}
enum Languages {
    JAVA,
    PYTHON,
    C,
    CPP,
    DBMS
}
```

Terminal Test cases

Shivanshojha

34.1.5. Enum constructors

00:55

A Java enum type can have a private constructor that can be used to initialize instance fields. Which means Java Enum can have a constructor to pass data while creating Enum constants. This feature allows you to associate related data together. Let us consider this with example.

```
public enum TrafficSignal{

    RED("wait"), GREEN("go"), ORANGE("slow down"); //this will call enum constructor with one Strin

    private String action;

    public String getAction(){ // enum constructor - can not be public or protected
        return this.action;
    }


    TrafficSignal(String action){
        this.action = action;
    }
}
```

In the above example, Enum, where we pass the action to each Enum instance e.g. GREEN, is associated with go, RED is associated with stop and ORANGE is associated with the slowdown.. We can also provide one or more constructor to Enum as it also supports constructor overloading like normal Java classes.

Just remember that constructor in enums an only be either private or package level it can't be public or protected hence access modifier public and protected are not allowed to Enum constructor, it will result in compile-time error

- [ ] Enum constructor can be public
- [x] In enum the constructor is only private
- [x] Constructor overloading can be applied in enum

Shivanshojha

34.1.6. Problems on Enum using Constructor

05:16

The below program has an enum class with `Student` which has marks of different students. Define a constructor and get the marks of each student in the following way.

```
Total Marks:
Radha got 90 marks.
Seetha got 62 marks.
Ram got 50 marks.
```

Sample Test Cases +

Main.java

Submit

```java
package q24202;
enum Student {
  Radha(90),Seetha(62),Ram(50);
  private int total;
      //write your code here
      //define a constructor Student and get total marks
      int marks;
    int getmarks(){
        return marks;
    }
    Student (int marks){
        this.marks=marks;
    }

}
public class Main {
    public static void main(String args[]){
        System.out.println("Total Marks: ");
        Student stu;
    System.out.println("Radha got "+Student.Radha.getmarks()+" marks.");
    System.out.println("Seetha got "+Student.Seetha.getmarks()+" marks.");
    System.out.println("Ram got "+Student.Ram.getmarks()+" marks.");
        // write your code here
    }
}
```

Terminal Test cases

Shivanshojha

35.1.1. Importance of java.lang package

01:36

`java.lang` is a package which contains many fundamental classes used in writing Java code. It is present in JDK and JRE installations.

For example, it contains classes like :

1. Object - The `Object` class is the root class of all classes in Java.
2. System - The `System` class provides access to the standard input, output and error streams of the running Java program. It also has many other useful methods.
3. Wrapper Classes - Whenever we want to represent a primitive `int` value as an object, we use its wrapper class `Integer`. The `java.lang` package contains all the wrapper classes: Boolean, Byte, Character, Short, Integer, Long, Float and Double.
4. String, StringBuilder classes.
5. Math - The `Math` class contains many useful methods for numeric operations like the square root, power, trigonometric operations etc.
6. Class - The class called `Class` represents the loaded classes and interfaces in memory during runtime.
7. Other classes - The `java.lang` package also has many other useful classes like ClassLoader, Process, Runtime, Thread, Runnable, Throwable, Exception, Error, etc.

All public classes present in `java.lang` package (like the classes mentioned above) are by default imported in every Java class. Meaning, we can directly start using these classes without using the import statement.

- [x] The `System` class is present in `java.lang` package.
- [ ] The `Date` class is present in `java.lang` package.
- [x] The class called `Class` is present in `java.lang` package.
- [ ] We must declare `import java.lang.System;` statement for code statements like `System.out.println("Hello");` to work.

Shivanshojha

35.2.1. Useful methods In Math class

04:09

The Math class present in `java.lang` package contains many useful static methods. These static methods are also called as utility methods.

Classes like `Math` that have only static methods are also called Utility classes.

Note: Most of the methods in `Math` class use `double` as data type, so we will see fractional part in the results.

In the given editor, Follow the comment lines to understand and complete the code

Note: Please don't change the package name.

Sample Test Cases +

MathExa...

Submit

```java
package q11239;
import java.util.*;
//Define MathExamples class
class MathExamples{
	public static void main(String[] args){

//Define the main method


		Scanner sc = new Scanner(System.in);
		double base = sc.nextDouble();
		/*Read the exponenet input from the user as base
		using nextDouble method*/
		double expo=sc.nextDouble();
		System.out.println(base+" to the power "+expo+" is : "+Math.pow(base,expo));
		System.out.println("Square root of "+base+" is : "+Math.sqrt(base));

		/*calculate the result of raising
		base to the power of exponent using
		Math.pow(base,exponent)*/

		/*calculate the square root
		of the base value using
		Math.sqrt(base)*/
	}
}
```

Terminal Test cases

35.3.1. Java code conventions for naming

05:24

Code should be written in such a way that fellow programmers can understand easily. Most of the time the code written by one author is maintained by someone else. In such a scenario code would be easy to read and maintain if all authors followed some common rules or conventions..

Java language designers have provided guidelines called Java code conventions for writing Java code.

Java naming rules for classes, interfaces, methods and variables employ camel-case convention. In camel-case convention, when two or more words are joined together the starting letter of each word joined will be written in uppercase.

For example: SimpleDateFormat, getAge, maxSpeed.

Below are the rules for naming:

1. **Class** - A class name should be a noun and not a verb. For example `NumberPrinter` is preferred over `PrintNumbers`. It should follow camel-case and the starting letter should also be in **uppercase**. Eg: `String`, `StringBuilder`, `ArrayList`
2. **Interface** - The same rules followed for a class name.
3. **Method** - A method name should be a verb and not a noun. For example `getAge()` is preferred over `age()`. It should follow camel-case and the starting letter should be in **lowercase**. Eg: `clear()`, `toString()`, `resetURLConnection()`
4. **Variable** - A variable name should be a meaningful and clearly indicate what it is going to store. It should be a noun. For example `maxAge` is preferred over `x`. It should follow camel-case and the starting letter of the first word should be in **lowercase**. Eg: `speed`, `maxSpeed`
   Single-letter variables like `x`, `y`, `z` - should never be used.
   Single-letter variables like `i`, `j`, `k` etc - can be used as loop-counters in for loops.
5. **Constant** - A constant variable is one which is declared as `static` and `final`. Its name should be a meaningful and clear like a simple variable, but it does not follow camel-case. It is written completely in uppercase letters with '_' as a word separator. Eg:
   `public static final int MAX_WIDTH = 100;`

According to Java naming conventions, select all the correct statements in the below sample code :

- [ ] The class name `ParseText` is correct in Statement 1.
- [ ] In Statement 1, the class name does not follow camel-case convention. It should have been `parseText`.
- [ ] The method name `GET_LENGTH` in Statement 2 is correct.
- [x] In Statements 2 and 3, the variable name `textcontent` should have been written as `textContent`.

Shivanshojha

35.3.2. Java code conventions for spaces and blank lines

00:23

According to Java code conventions, a **blank line** is recommended to be used between two sections in a class. For example, between two methods.

Below are the rules for using a space ( ):

1. Between **keyword** which is followed by **parenthesis**:

```
while (condition) {
    ...
}
for (..; ...; ...) {
    ...
}
```

Note : There **should not** be a space between method name and the parenthesis that follows it.

2. A space should appear after commas. For example:

```
public int sum(int num1, int num2) {
    ...
}
String[] namesArr = {"Aries", "Taurus", "Gemini", "Cancer"};
```

3. A space should be used between all **binary operators** and their **operands** except dot (.). **Unary operators** (-, -- and ++) **must never be separated** from their operands by a space. For example:

```
int total = num1 + num2;
for (int i = 0; i < namesArr.length; i--) {
    ...
}
```

4. A space should be used between expressions of a for statement. For example:

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

- In statement 1, there should not be a space between the method name and the opening parenthesis. isPrimeNumber (
- In statement 2, there should be a space between `for` and the opening parenthesis that follows it.
- In statement 2, there should be a space on either side of `=` (assignment operator) in the initialization expression `int i=2;`.
- In statement 2, there should be a space on either side of `/` and `<=` operators in the condition expression `i<=(number/2);`.
- In statement 2, there should be a space after the semicolon `;` of the condition expression `i<=(number/2);`.
- In statement 3, there should be a space on either side of `%` and `==` operators in the if's condition expression `(number%i)==0`.

Shivanshojha

35.4.1. Understanding Autoboxing and Unboxing

00:19

Java compiler automatically converts primitive types to their corresponding wrapper classes when needed. This is called **Autoboxing**.

Similarly, when the conversion is from a wrapper class to its corresponding primitive type, it is called **Unboxing**.

For example, in the below code, in the expression where `total` is calculated, `num2` is converted (unboxed) from `Integer` to `int` and the summation is again converted (autoboxed) from `int` to `Integer` before being assigned to `total`:

```
int num1 = 20;
Integer num2 = new Integer(10);
Integer total = num1 + num2;
```

Note: Autoboxing and Unboxing is a concept employed by compiler. The programmer only needs to understand how the compiler employs it.

Understanding the above concept, select all the correct statements for the below code:

```
public class AutoBoxingExample {
    public static void main(String[] args) {
        int num1 = 3; //statement 1
        System.out.println("Square of " + num1 + " is : " + calculateSquare(num1)); //statement 2
    }

    public static Integer calculateSquare(Integer number) { //statement 3
        return number * number;  //statement 4
    }
}
```

- [ ] In statement 1, `num1` is being autoboxed.
- [ ] In statement 2, in the method invocation `calculateSquare(num1)`, `num1` is being unboxed.
- [ ] In statement 4, there is neither autoboxing or unboxing.
- [x] In statement 4, `number` is first unboxed and later the product is autoboxed gain before being returned.

Shivanshojha

35.4.2. Choose the correct result

00:20

Given :

```
public static void main(String[] args) {
    Integer i = new Integer(1) + new Integer(2);
     switch(i) {
        case 3:
            System.out.println("three");
            break;
        default:
            System.out.println("other");
            break;
    }
}
```

What is the result?

- [x] three
- [ ] other
- [ ] An exception is thrown at runtime.
- [ ] Compilation fails because of an error on `Integer i = new Integer(1) + new Integer(2);`
- [ ] Compilation fails because of an error on `switch(i) {`
- [ ] Compilation fails because of an error on`default: System.out.println("other"); break;`

Shivanshojha