L17

Monday, January 24, 2022 2:41 PM

Loops–syntax and applications
Saurav Hathi

https://www.youtube.com/channel/UCp6MFWao5vWRnyRCxBsKnfw

for - syntax and application

Q1. 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:

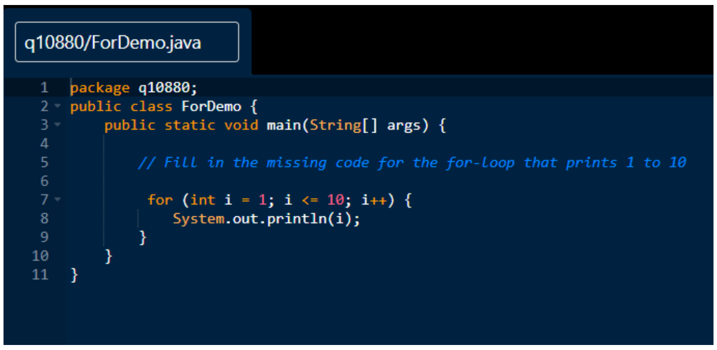
```
for (int i = 0; i < 10; i++) {
  System.out.println(i);
}
```

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.
4. It is a good practice to always keep the loop body which contains the code to be executed within an opening-brace { and a closing-brace }.

In the below program, the class ForDemo should print numbers from 1 to 10 on separate lines.
The code in the main method should use a for loop to initialize a loop counter i to 1, then print the value in i and increment i.

The for loop should iterate until i's value is **less than or equal to** 10.

Note: Pay attention to the **less than or equal to** condition in the above statement. Make sure to use println(...) and not print(...).


q10880/ForDemo.java

```
package q10880;
public class ForDemo {
    public static void main(String[] args) {

        // Fill in the missing code for the for-loop that prints 1 to 10

         for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

Q2. 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:

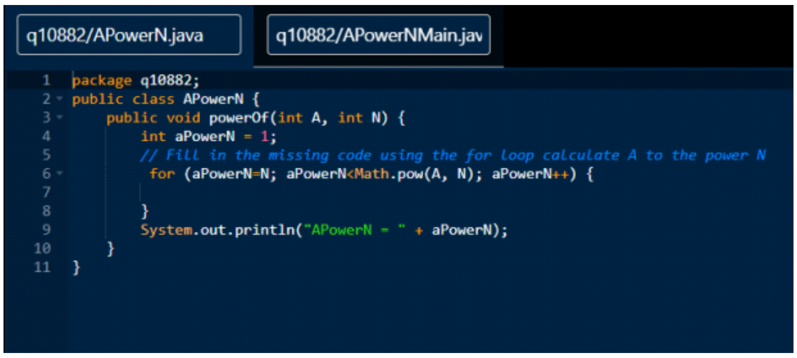
1. A and N are two positive and non-zero numbers

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.

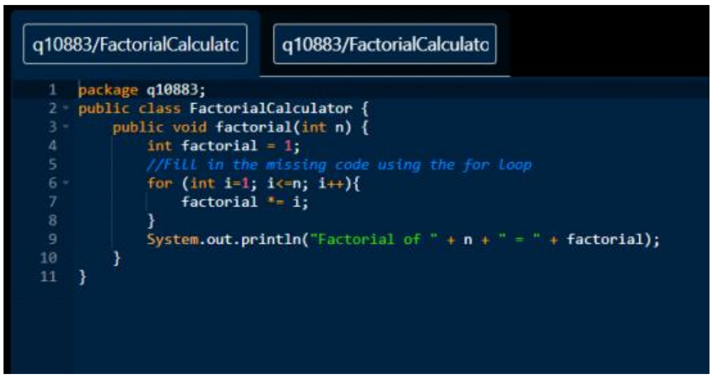


```java
package q10882;
public class APowerN {
    public void powerOf(int A, int N) {
        int aPowerN = 1;
        // Fill in the missing code using the for loop calculate A to the power N
         for (aPowerN=N; aPowerN<Math.pow(A, N); aPowerN++) {

        }
        System.out.println("APowerN = " + aPowerN);
    }
}
```

Q3. 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.

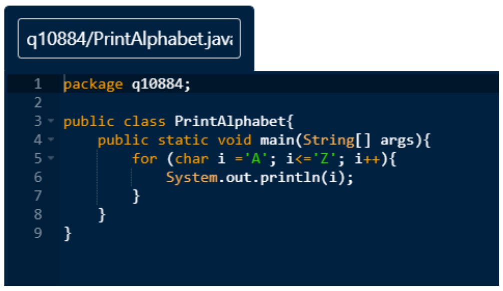


```java
package q10883;
public class FactorialCalculator {
    public void factorial(int n) {
        int factorial = 1;
        //Fill in the missing code using the for loop
        for (int i=1; i<=n; i++){
            factorial *= i;
        }
        System.out.println("Factorial of " + n + " = " + factorial);
    }
}
```

Q4. 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'.

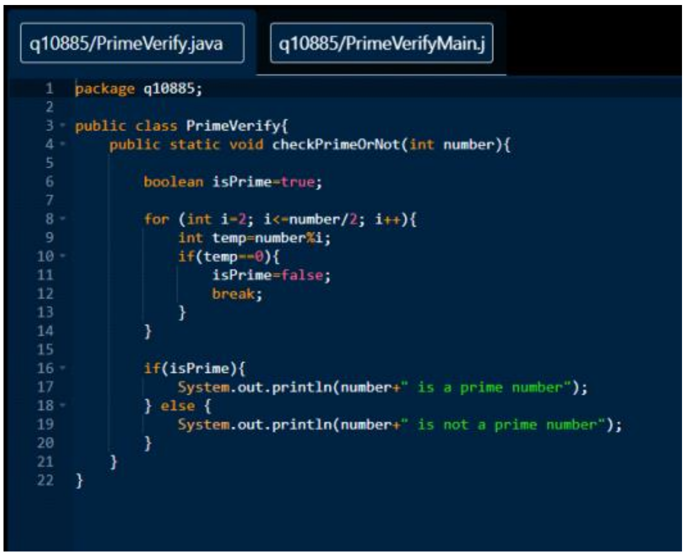
You can declare and initialize a loop counter char i and initialize it to 'A' (eg: char i = 'A';). The condition can similarly be i <= 'Z';.
And the update statement can be i++ then print the value of i.



```java
package q10884;

public class PrintAlphabet{
    public static void main(String[] args){
        for (char i ='A'; i<='Z'; i++){
            System.out.println(i);
        }
    }
}
```

Q5. 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.

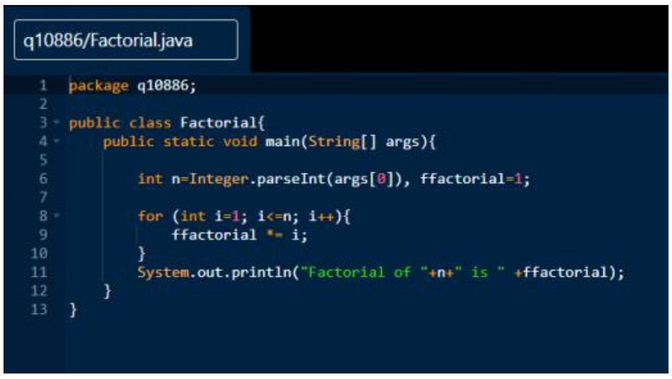
For example:
Cmd Args : 13
13 is a prime number



```java
package q10885;

public class PrimeVerify{
    public static void checkPrimeOrNot(int number){

        boolean isPrime=true;

        for (int i=2; i<=number/2; i++){
            int temp=number%i;
            if(temp==0){
                isPrime=false;
                break;
            }
        }

        if(isPrime){
            System.out.println(number+" is a prime number");
        } else {
            System.out.println(number+" is not a prime number");
        }
    }
}
```

Q6. 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



```java
package q10886;

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

        int n=Integer.parseInt(args[0]), ffactorial=1;

        for (int i=1; i<=n; i++){
            ffactorial *= i;
        }
        System.out.println("Factorial of "+n+" is " +ffactorial);
    }
}
```

while - syntax and application

Q1. 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).

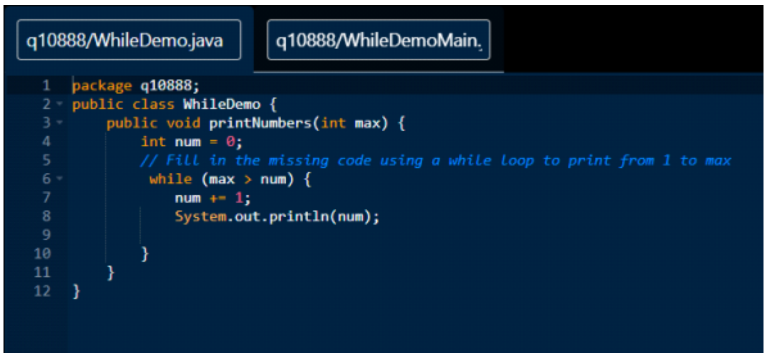
- If it evaluates to true, the body containing one or more code statements is executed.
- If the expression evaluates to false, the body containing code statements is not executed.

For example:

```
int value = 4, sum = 0;
while (value > 0) { // start of while loop bodysum = 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.

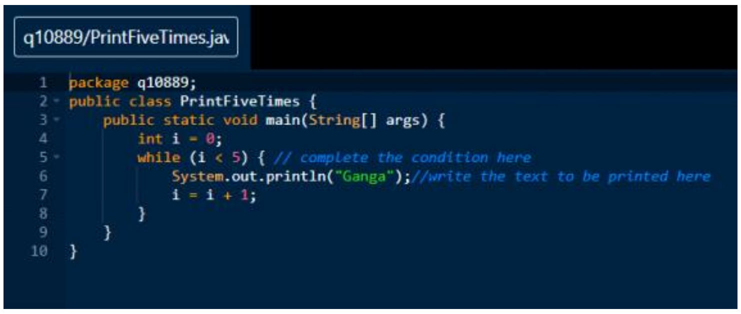
In the below program with class WhileDemo, fill in the missing code inside the printNumbers(...) method using a while loop to print numbers from 1 to max (including max).



```java
package q10888;
public class WhileDemo {
    public void printNumbers(int max) {
        int num = 0;
        // Fill in the missing code using a while loop to print from 1 to max
         while (max > num) {
            num += 1;
            System.out.println(num);

        }
    }
}
```

Q2. Write a class PrintFiveTimes with a **main** method. The program should print **Ganga** five times.

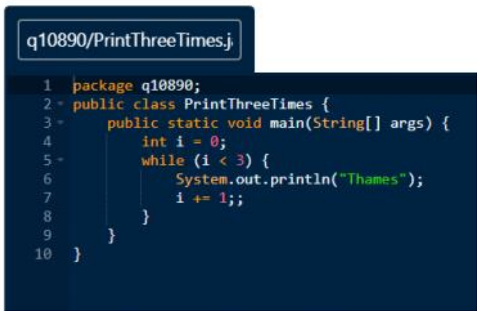
Fill the missing code in the below program so that it produces the desired output.



```java
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;
        }
    }
}
```

Q3. 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.



```java
package q10890;
public class PrintThreeTimes {
    public static void main(String[] args) {
        int i = 0;
        while (i < 3) {
            System.out.println("Thames");
            i += 1;;
        }
    }
}
```

Q4. 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

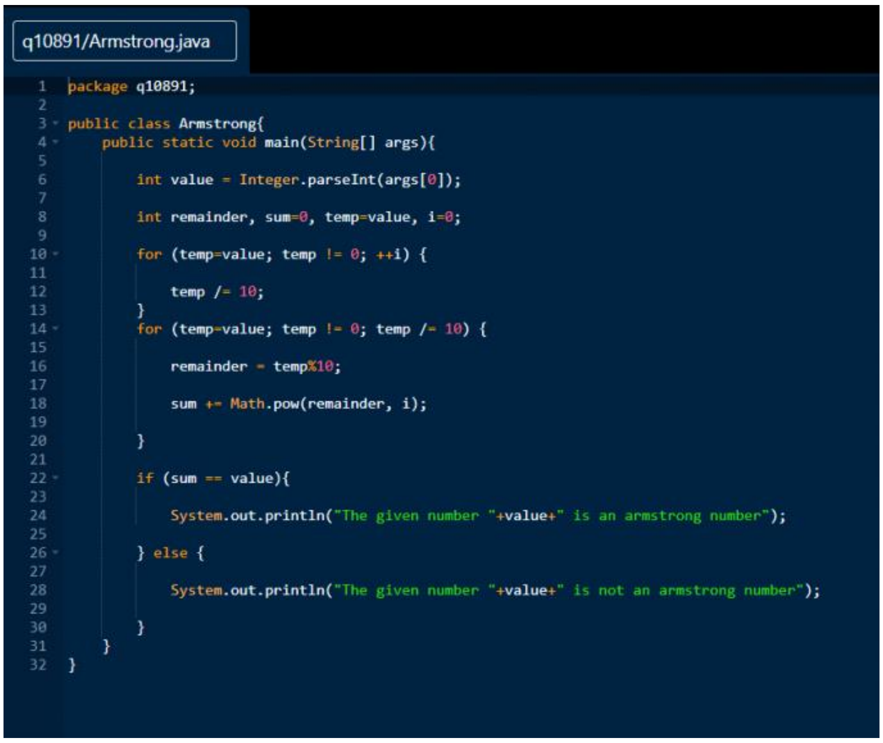
$9 = 9_1 = 9$
$371 = 3_3 + 7_3 + 1_3 = 27 + 343 + 1 = 371$
$8208 = 8_4 + 2_4 + 0_4 + 8_4 = 4096 + 16 + 0 + 4096 = 8028$

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
```

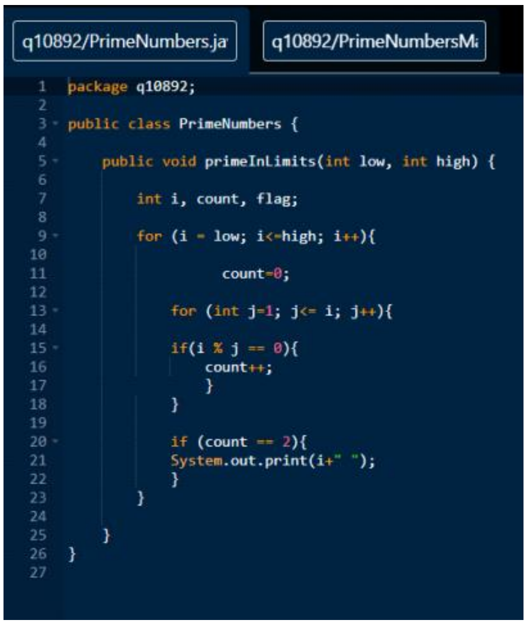

q10891/Armstrong.java

```java
package q10891;

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

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

        int remainder, sum=0, temp=value, i=0;

        for (temp=value; temp != 0; ++i) {

            temp /= 10;
        }
        for (temp=value; temp != 0; temp /= 10) {

            remainder = temp%10;

            sum += Math.pow(remainder, i);

        }

        if (sum == value){

            System.out.println("The given number "+value+" is an armstrong number");

        } else {

            System.out.println("The given number "+value+" is not an armstrong number");

        }
    }
}
```

Q5. 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
```

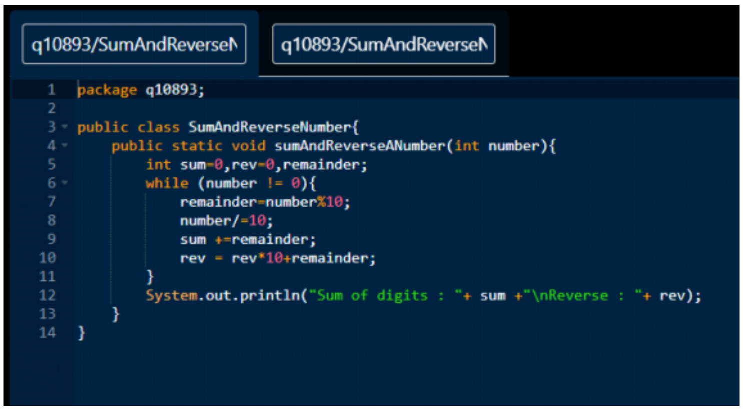

q10892/PrimeNumbers.ja q10892/PrimeNumbersM

```java
package q10892;

public class PrimeNumbers {

    public void primeInLimits(int low, int high) {

        int i, count, flag;

        for (i = low; i<=high; i++){

               count=0;

            for (int j=1; j<= i; j++){

            if(i % j == 0){
                count++;
                }
            }

            if (count == 2){
            System.out.print(i+" ");
            }
        }

    }
}
```

Q6. 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:

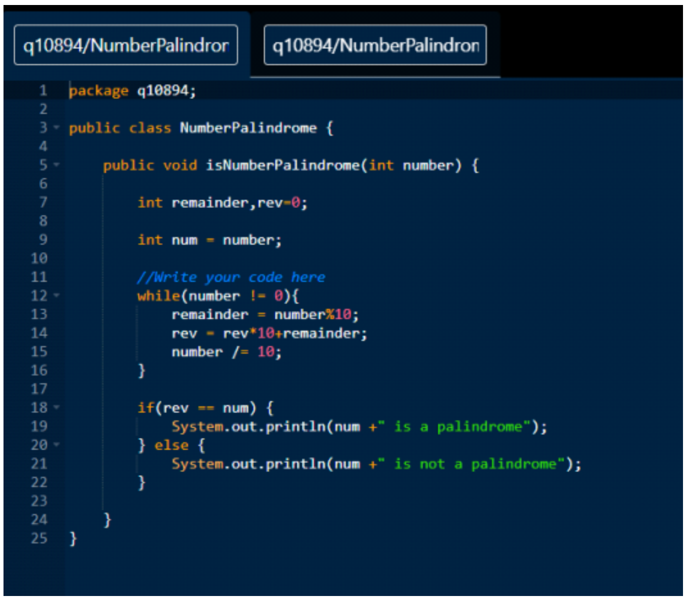
```
Sum of digits : 18
Reverse : 963
```



```java
package q10893;

public class SumAndReverseNumber{
    public static void sumAndReverseANumber(int number){
        int sum=0,rev=0,remainder;
        while (number != 0){
            remainder=number%10;
            number/=10;
            sum +=remainder;
            rev = rev*10+remainder;
        }
        System.out.println("Sum of digits : "+ sum +"\nReverse : "+ rev);
    }
}
```

Q7. 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



```java
package q10894;

public class NumberPalindrome {

    public void isNumberPalindrome(int number) {

        int remainder,rev=0;

        int num = number;

        //Write your code here
        while(number != 0){
            remainder = number%10;
            rev = rev*10+remainder;
            number /= 10;
        }

        if(rev == num) {
            System.out.println(num +" is a palindrome");
        } else {
            System.out.println(num +" is not a palindrome");
        }

    }
}
```

do-while - syntax and application

Q1. 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.

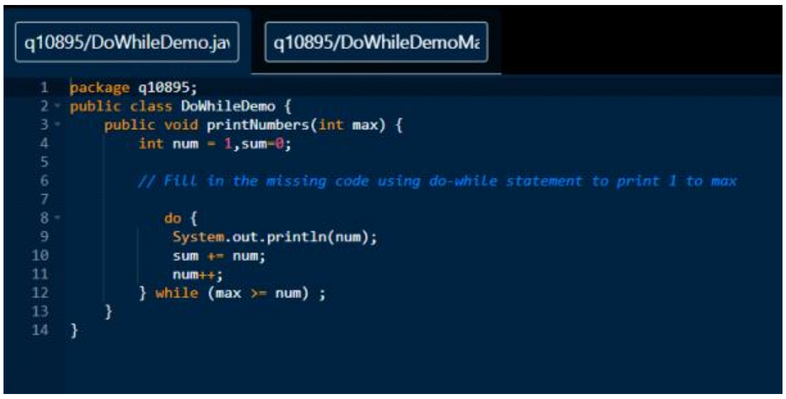
- If it evaluates to true, the body containing one or more code statements is executed.
- If the expression evaluates to false, the body containing code statements is not executed.

For example:

```
        int value = 4, sum = 0;
        do { // start of do-while loop bodysum = 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.
4. A do-while loop's body can contain two branching statements, break; and continue;, which we will learn later.

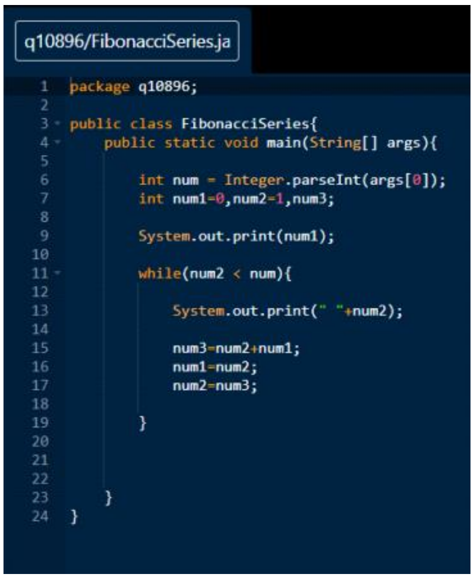
In the below program with class DoWhileDemo, fill in the missing code inside the printNumbers(...) method using a do-while loop to print numbers from 1 to max (including max).



```java
package q10895;
public class DoWhileDemo {
    public void printNumbers(int max) {
        int num = 1,sum=0;

        // Fill in the missing code using do-while statement to print 1 to max

          do {
           System.out.println(num);
           sum += num;
           num++;
        } while (max >= num) ;
    }
}
```

Q2. 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



```java
package q10896;

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

        int num = Integer.parseInt(args[0]);
        int num1=0,num2=1,num3;

        System.out.print(num1);

        while(num2 < num){

            System.out.print(" "+num2);

            num3=num2+num1;
            num1=num2;
            num2=num3;

        }



    }
}
```

Branching Statements

Q1. 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;

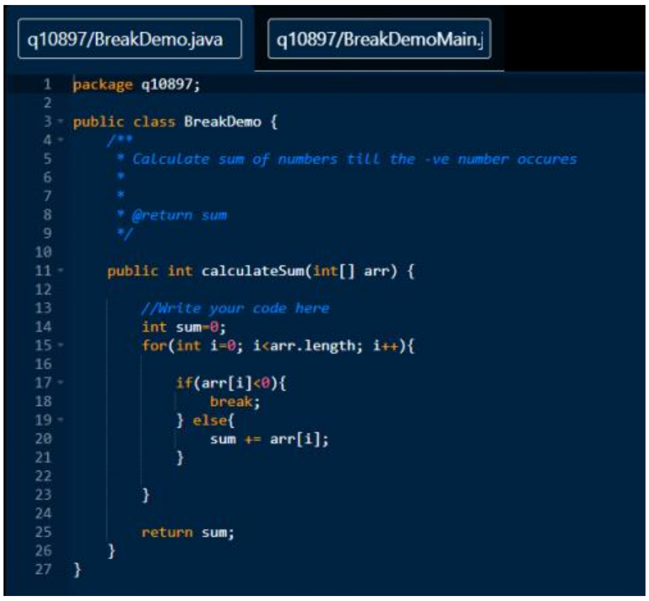
```
for (int i = 1; i < 10; i++) {
        if (i % 5 == 0) {
                break; // breaks from for-loop, if i is divisible by 5.}
        System.out.println(i);
}
```

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.



```
package q10897;

public class BreakDemo {
    /**
     * Calculate sum of numbers till the -ve number occures
     *
     *
     * @return sum
     */

    public int calculateSum(int[] arr) {

        //Write your code here
        int sum=0;
        for(int i=0; i<arr.length; i++){

            if(arr[i]<0){
                break;
            } else{
                sum += arr[i];
            }

        }

        return sum;
    }
}
```

Q2. 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:

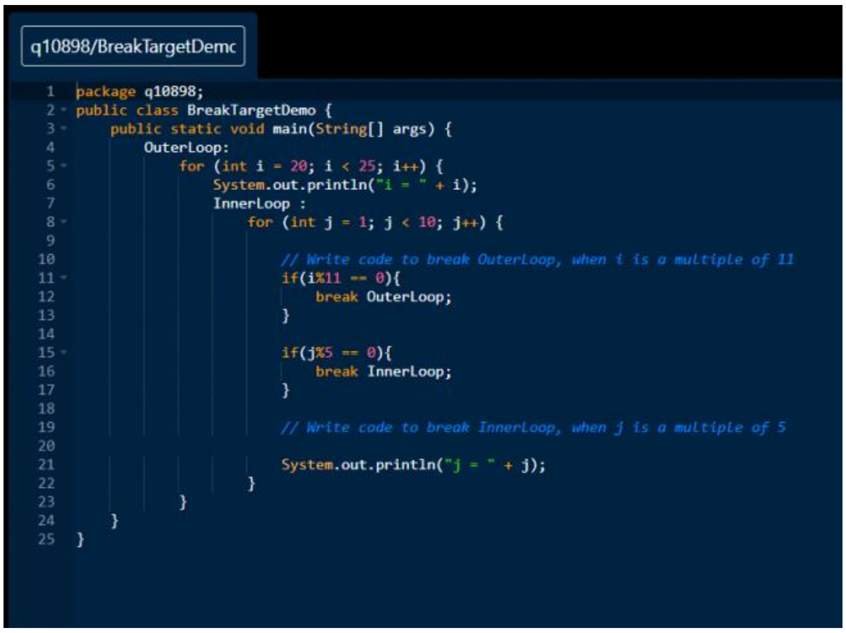
```
for (int i = 1; i < 10; i++) {
        System.out.println("Step : 1");
        SkipMe : {
                if (i % 5 == 0) {
                        break SkipMe; // breaks to the SkipMe : label}
                System.out.println(i);
        }
        System.out.println("Step : 2");
}
```

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...

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

In the below program with class BreakTargetDemo, write the appropriate conditions inside the for loop labeled InnerLoop such that:

1. when the value of i is a multiple of 11 (meaning i divided by 11 will leave a reminder 0), it should call break OuterLoop;
2. when the value of j is a multiple of 5 (meaning j divided by 5 will leave a reminder 0), it should call break InnerLoop;.


q10898/BreakTargetDemc

```java
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++) {

                        // Write code to break OuterLoop, when i is a multiple of 11
                        if(i%11 == 0){
                            break OuterLoop;
                        }

                        if(j%5 == 0){
                            break InnerLoop;
                        }

                        // Write code to break InnerLoop, when j is a multiple of 5

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

Q3. 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.

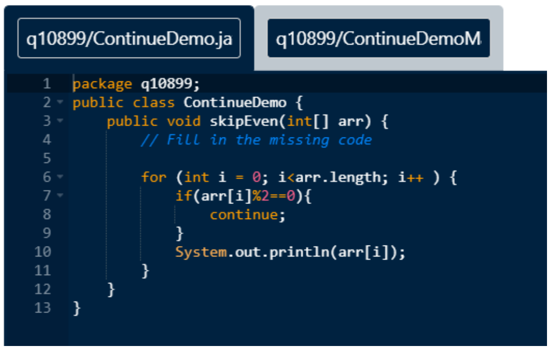
Similar to a break statement, a continue statement can be written simply as continue; or with a label as continue targetLabelName;

```
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.

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 printing the even numbers.


q10899/ContinueDemo.ja q10899/ContinueDemoM

```java
package q10899;
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]);
        }
    }
}
```

Q4. 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.

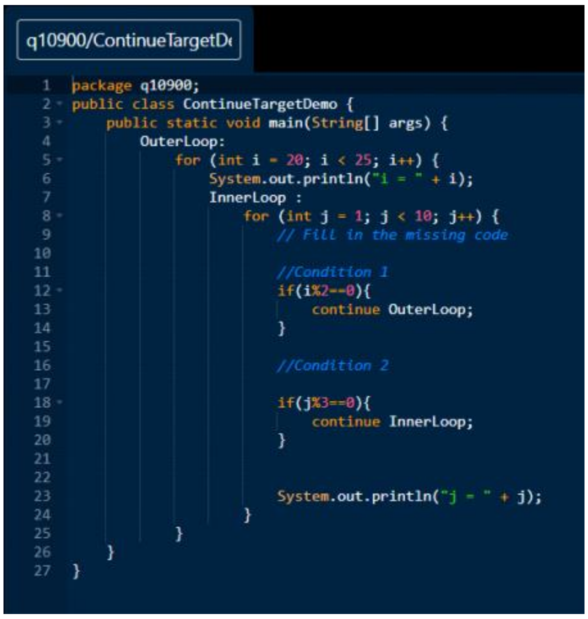
This allows one to directly transfer control (using continue) to a labelled loop block. For example in the below code:

```
OuterLoop :
while (condition1) {
        //do something...//do something...InnerLoop :
        while (condition2) {
                //do something...//do something...if (condition3) {
                        continue OuterLoop;
                }
        }//end of inner-while}//end of outer-while
```

The statement continue OuterLoop; will transfer the control directly to the outer-most while which works on condition1.

In the below program under the comment *// Fill in the missing code* write code for the below two conditions:

1. If the value of i is a multiple of 2 continue to the **OuterLoop**
2. If the value of j is a multiple of 3 continue to the **InnerLoop**


q10900/ContinueTargetDe

```java
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++) {
                        // Fill in the missing code

                        //Condition 1
                        if(i%2==0){
                            continue OuterLoop;
                        }

                        //Condition 2

                        if(j%3==0){
                            continue InnerLoop;
                        }


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

When to use which loop

Q1. 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:

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