For loop

Consider the following code which was written to find the minimum element in an array of numeric type and of any length:

```
n1 = v[0];
for (int i = 1; i < v.length; i++)
   if (v[i] < n1)
      n1 = v[i];
```

What is wrong with this code? You can assume any necessary declarations and initialisations have occurred.

```
public class Test{
   public static void main(String[ ] args){
      int[ ]v={1,3,4,-9,8,-5,-2,0};
      int n1;
      n1=v[0];
      for (int i=1; i<v.length; i++)
         if (v[i]<n1)
            n1=v[i];
      System.out.println(n1);
   }
}
```

output: -9

Select one:

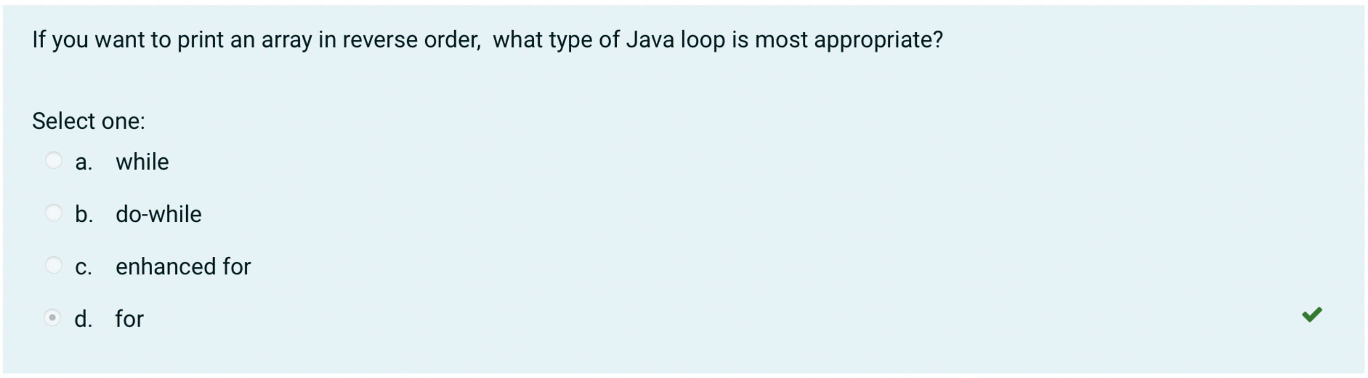
- a. It will not compile
- b. n1 = v[0]; should be n1 = 0;
- c. int i = 1 should be int i = 0
- d. n1 = v[i]; should be v[i] = n1
- e. The **if** condition should be n1 < v[i]
- [✓ f. If you think there is nothing wrong with the code, select this option] ✓

The following piece of code is developed to find the maximum value in an array of integers, but, it throws a **syntax error.** Which line does the error occur on?

```
        int ar[]={4,3,6,7};
        int max=ar[0];
        int index=0;
        for(int i=0;i<ar.length;i++){
            if (ar[i]>max)
                max=ar[i];
                index=max;
            }
        System.out.println("The max is ="+max+" in cell with index="+i);
        }
```

Select one:

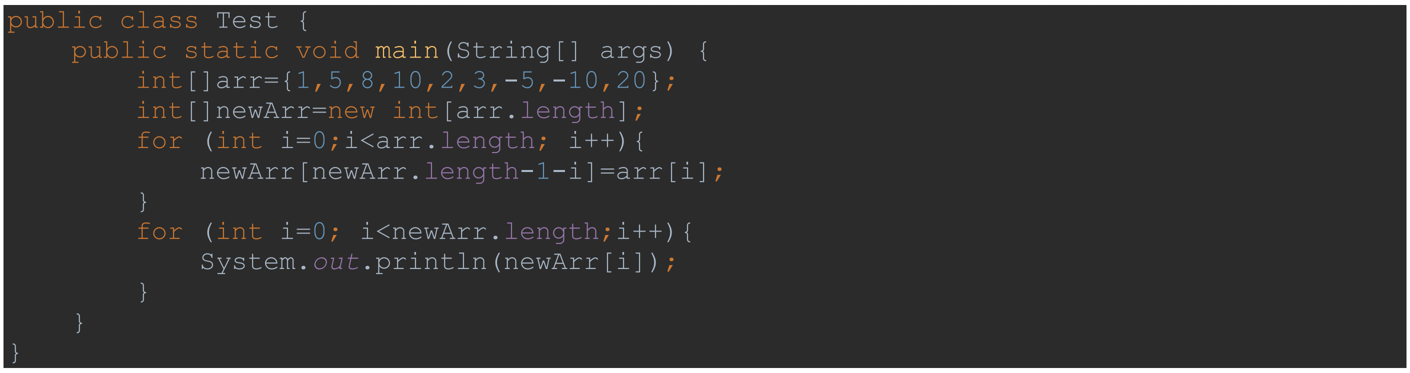
- [✓ a. 9] scope of i: within for loop ✓
- b. 7
- c. 2
- d. 5
- e. 4
- f. 6



If you want to print an array in reverse order, what type of Java loop is most appropriate?

Select one:

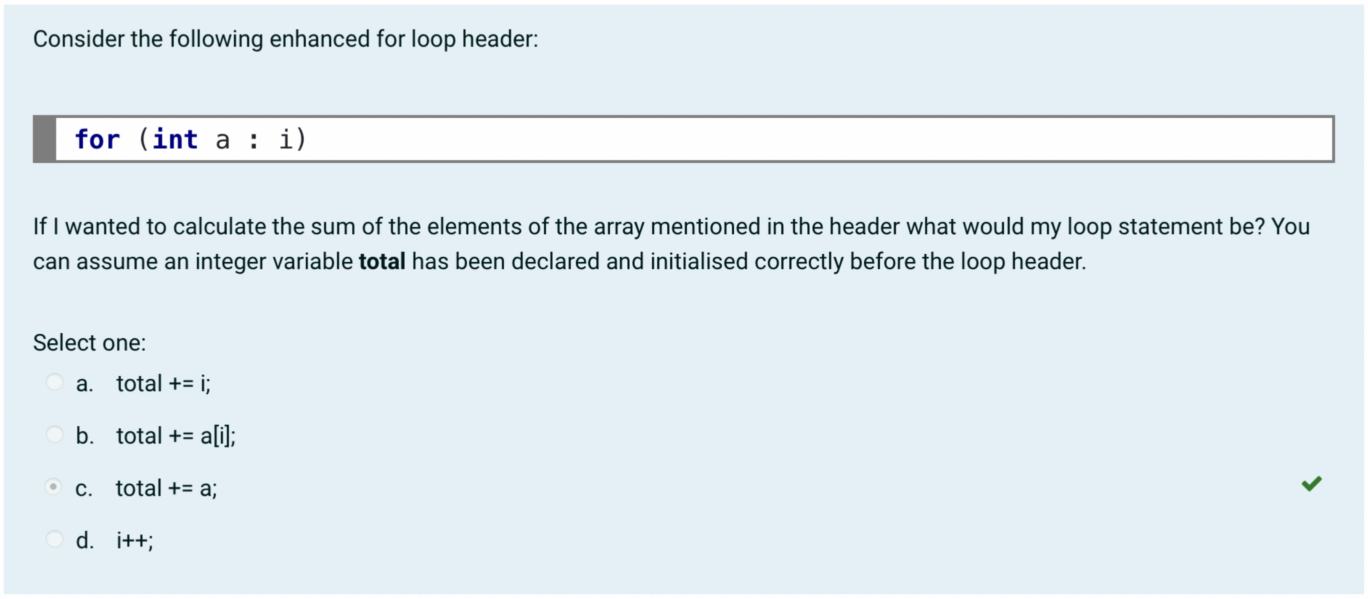
- a. while
- b. do-while
- c. enhanced for
- [✓ d. for] ✓



```java
public class Test {
    public static void main(String[] args) {
        int[]arr={1,5,8,10,2,3,-5,-10,20};
        int[]newArr=new int[arr.length];
        for (int i=0;i<arr.length; i++){
            newArr[newArr.length-1-i]=arr[i];
        }
        for (int i=0; i<newArr.length;i++){
            System.out.println(newArr[i]);
        }
    }
}
```

output: 20 -10 -5 3 2 10 8 5 1

Enhanced for loop



Consider the following enhanced for loop header:

```
for (int a : i)
```

If I wanted to calculate the sum of the elements of the array mentioned in the header what would my loop statement be? You can assume an integer variable **total** has been declared and initialised correctly before the loop header.

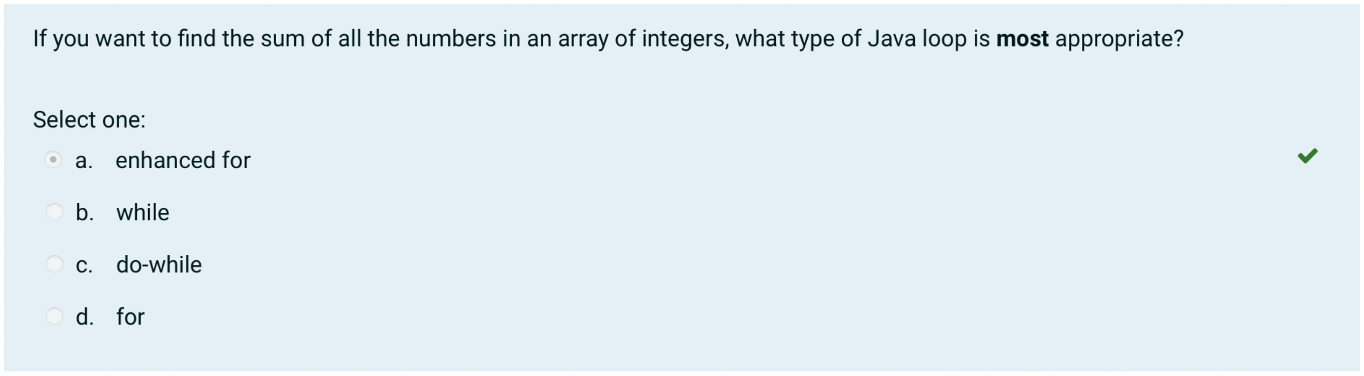
Select one:

a. total += i;

b. total += a[i];

[✓ c. total += a;] ✓

d. i++;



If you want to find the sum of all the numbers in an array of integers, what type of Java loop is **most** appropriate?

Select one:

[✓ a. enhanced for] ✓

b. while

c. do-while

d. for

While loop

Consider the following code OUTLINE:

```
int i, j, k = 0;
i = j = 1;
while (i < 5){
   //:
   while (j < 10) {
      //some statements    then  k++;
   }
   //:
}
```

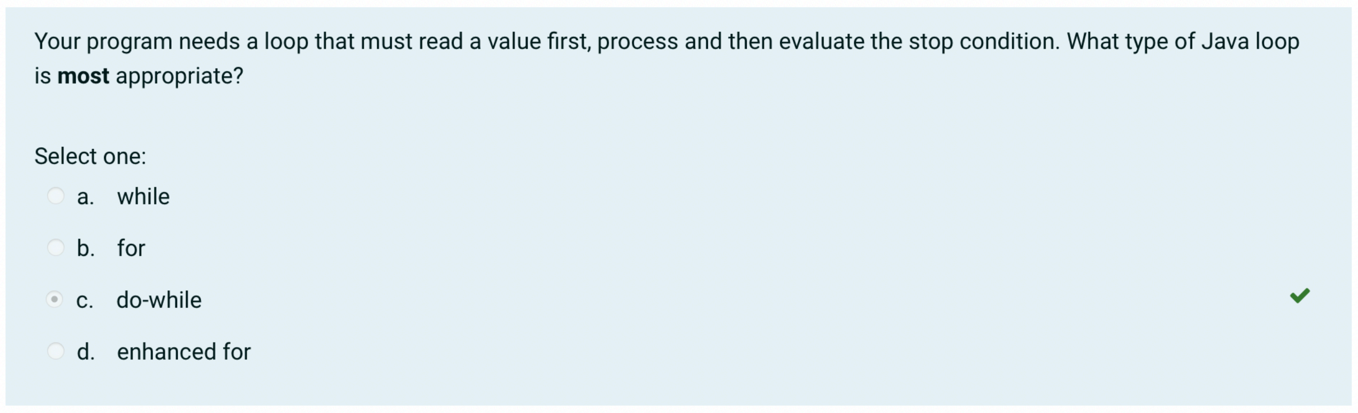
You can assume the outer loop is meant to repeat 4 times and the inner loop is meant to repeat 9 times.

k should count the total number of times "some statements" has executed. Which of the following is the **best** code location to increment **k**?

Select one:

- a. Immediately after the outer while
- b. Immediately before the outer while after declarations and initialisations
- [✓ c.] In the inner while immediately after the "inner loop statements" ✓
- d. In the outer while immediately after the inner while
- e. In the inner while immediately before the "inner loop statements"
- f. In the outer while immediately before the inner while

Do-while loop


Your program needs a loop that must read a value first, process and then evaluate the stop condition. What type of Java loop is **most** appropriate?

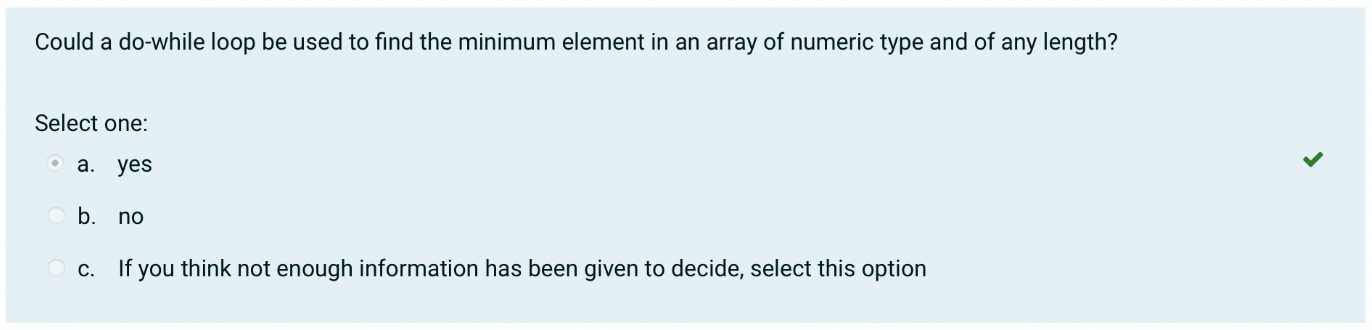
Select one:

a. while

b. for

[✓ c. do-while] ✓

d. enhanced for

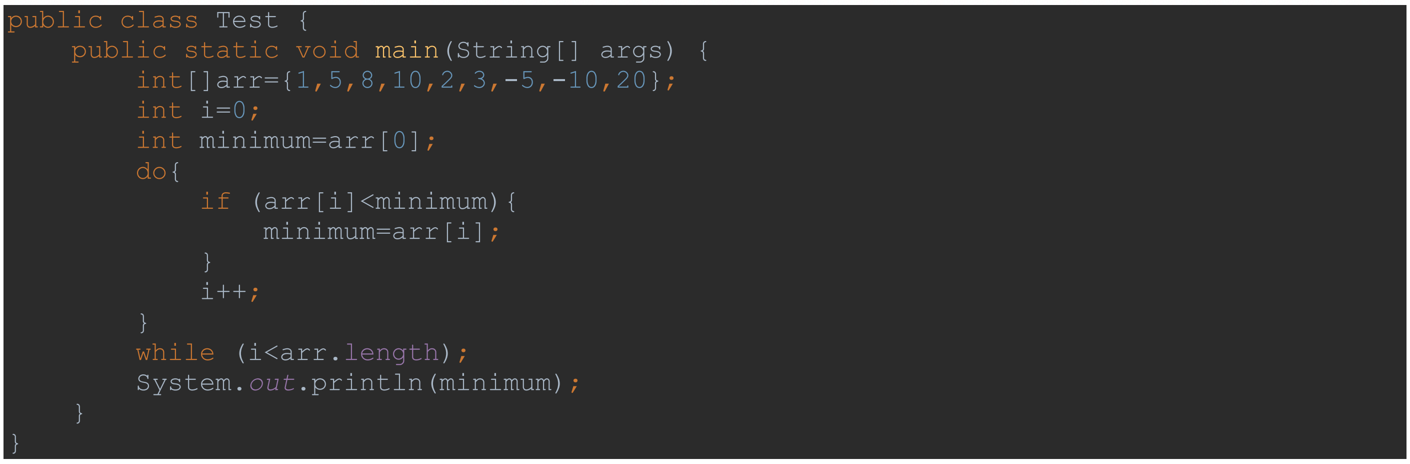

Could a do-while loop be used to find the minimum element in an array of numeric type and of any length?

Select one:

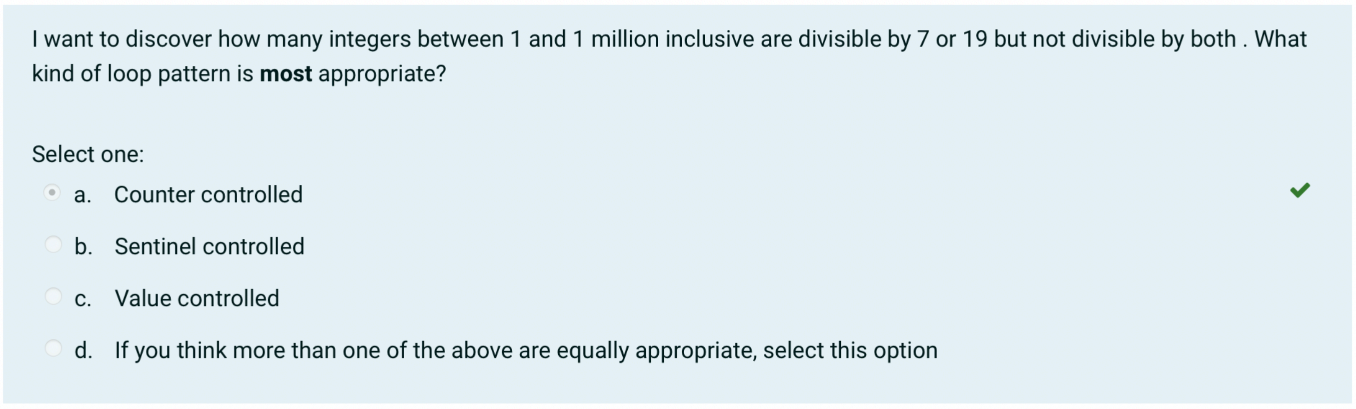
[✓ a. yes] ✓

b. no

c. If you think not enough information has been given to decide, select this option


```java
public class Test {
    public static void main(String[] args) {
        int[]arr={1,5,8,10,2,3,-5,-10,20};
        int i=0;
        int minimum=arr[0];
        do{
            if (arr[i]<minimum){
                minimum=arr[i];
            }
            i++;
        }
        while (i<arr.length);
        System.out.println(minimum);
    }
}
```
output: -10

Counter-controlled loop



I want to discover how many integers between 1 and 1 million inclusive are divisible by 7 or 19 but not divisible by both . What kind of loop pattern is **most** appropriate?

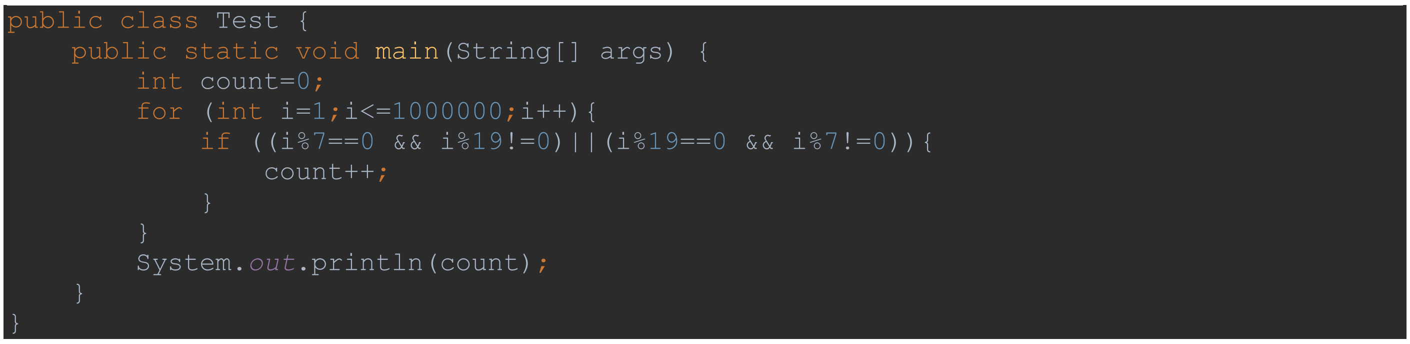
Select one:

[✓ a. Counter controlled]

b. Sentinel controlled

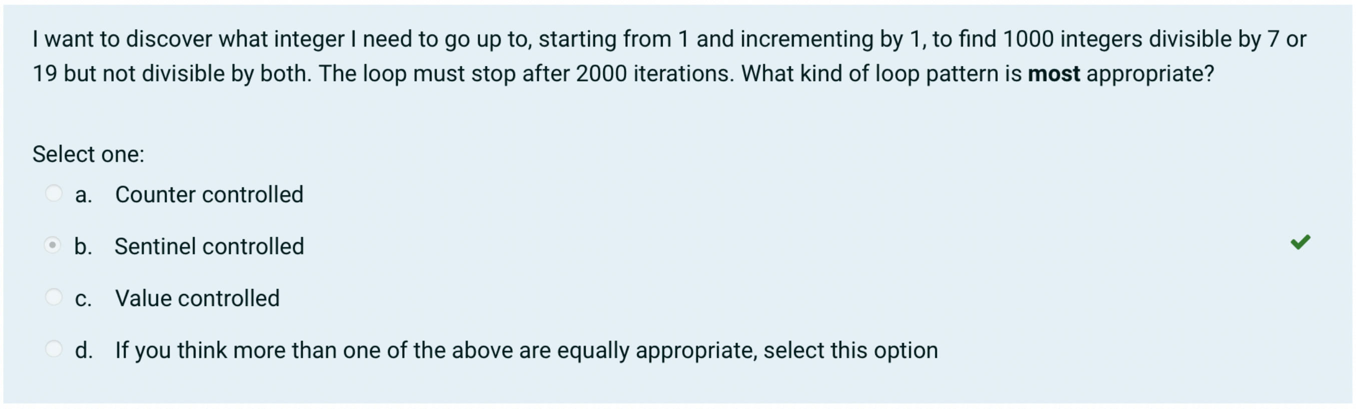
c. Value controlled

d. If you think more than one of the above are equally appropriate, select this option



```java
public class Test {
    public static void main(String[] args) {
        int count=0;
        for (int i=1;i<=1000000;i++){
            if ((i%7==0 && i%19!=0)||(i%19==0 && i%7!=0)){
                count++;
            }
        }
        System.out.println(count);
    }
}
```

Sentinel-controlled loop



I want to discover what integer I need to go up to, starting from 1 and incrementing by 1, to find 1000 integers divisible by 7 or 19 but not divisible by both. The loop must stop after 2000 iterations. What kind of loop pattern is **most** appropriate?

Select one:

a. Counter controlled

[✓ b. Sentinel controlled] ✓

c. Value controlled

d. If you think more than one of the above are equally appropriate, select this option

```
public class Test {
    public static void main(String[] args) {
        int totalRepeat = 0;
        int number = 1;
        int count = 0;
        while (totalRepeat < 2000 && count <= 1000) {
            if ((number %7==0&&number %19!= 0)||(number%19==0&&number%7!=0)){
                count++;
            }
            totalRepeat++;
            number++;
        }
        System.out.println(totalRepeat);
        System.out.println(count);
    }
}
```

output:

```
2000
360
```