CHAPTER 8-BASIC ITERATIONS AND ARRAYS IN JAVA

8.1 Objective

- Understand need of arrays in programming
- Understand procedure to declare, initialize and access arrays
- Understand and implement solutions involving arrays of primitive and custom types
- Understand iterations and loops
- Understand and implement solutions involving simple and nested iterations
- Understand finite, infinite loops
- Understand break and continue conditions
- Understand and implement solutions on arrays using iterations

8.2 Content

8.2.1 Arrays

We have seen how to declare variable in the previous section. int x; Here x is a variable of type int, so x will be capable of storing an integer value. At any point of time a variable can hold only a single value.

Few more eg.

```
int age=20;
int score=70;
```

Say, I need to store scores of last 5 matches. What kind of data type do I take? May be I will take 5 variables of type int.

```
int scoreMatch1;
int scoreMatch2;
int scoreMatch3;
int scoreMatch4;
int scoreMatch5;
```

Similarly, I need to record no. of kms I drove each day in last 5 days.

```
int kmsDay1;
int kmsDay2;
int kmsDay3;
int kmsDay4;
int kmsDay5;
```

Similarly, I need to record the amount spent for each day in last 5 days.

```
double expenseDay1;
double expenseDay2;
double expenseDay3;
double expenseDay4;
double expenseDay5;
```

The above approach will not be efficient if the requirement(of recording data for a specific period) changes from just last 5 days to last 30 days.

Array:

It is a single variable, capable of storing multiple values (of same data type). But an array does not store the multiple values directly. It is a collection of more than one variable, of same data type.

Declaring arrays:

The general syntax to declare an array is : data type variable name[]; The square brackets indicate that the variable is an array of the data type declared.

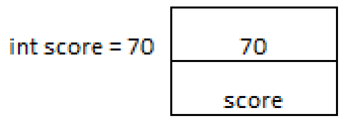
Example:

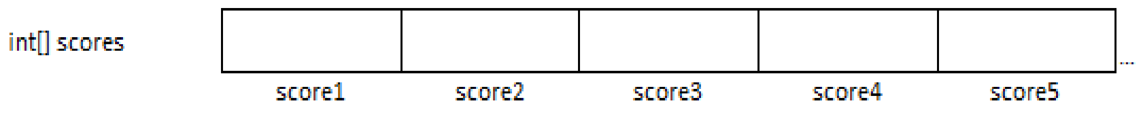
```
int scores [];
int kmsTravelled [];
double expenses [];
char grades[];
```

Also, all the above arrays are just declared, and have not yet been initialized(not defined the size).

Here scores is not just an int type variable. scores is an array of type int i.e. It is capable of storing more than one int value.

int score = 70; Here the variable score holds a value 70.



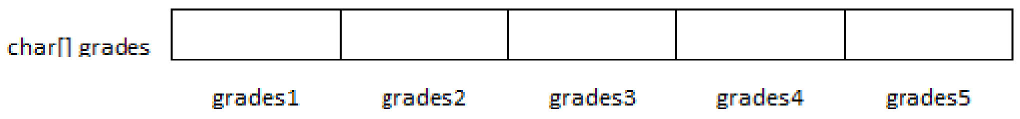


Here the array scores do not hold the values directly. It is just a reference (pointing to) to the 5 variables score1 … score5. Hence arrays are called references or objects in Java.

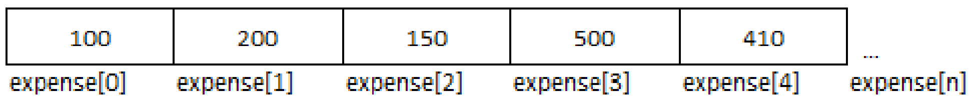
If we do not initialize an int variable, its default value is 0. In case of array (as it does not store value directly), if we do not initialize(do not say to which list of variables it refers to) it, the default value is null. i.e. it does not point to any list of variables.

Similarly, expenses is not just a double type variable. expenses is an array of type double i.e. it is capable of storing more than one double value.

Similarly, grades is not just a char type variable. grades is an array of type char i.e. it is capable of storing more than one char value.



Once an array is declared, you can think of multiple variables being generated automatically in a sequential order i.e. if an array double[] expense is declared, the variable that stores the first value will be expense[0], the variable that stores the second value will be expense[1] and so on.



Note: While talking about arrays, always remember two key things. One, about the array itself, whether it is initialized or not. Two, about the element inside the array – whether it is initialized or not.

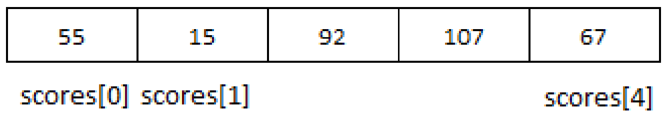
Initializing arrays – using {} initializer:

Arrays have a specific length(or size). The length of the array cannot be changed after it has been initialized. The elements in the array are put at index of the array. The first element in the array is at index 0, and the last element is at the index length-1.

To declare and initialize an array at the same time, refer to the below syntax:

Example 1:

```
int[] scores = {55,15, 92,107,67}
```



{} is called an initializer block. The length (or size) of an array is equal to the no. of values declared in the initializer block.

scores is an int array. Its size is 5. The value of first element is 55, value of second element is 15, value of third element is 92, value of fourth element is 107 and value of fifth element is 67.

Example 2:

```
int[] expenses = {1000,1500,50,200,0}
```

expenses is an int array. It's size is 5. The value of first element is 1000, value of second element is 1500, value of third element is 50, value of fourth element is 200 and value of fifth element is 0.

Example 3:

```
String[] softDrinks = {"Coca-Cola", "Pepsi", "Sprite", "Mountain Dew"};
```

softDrinks is a String array. It's size is 4. The value of first element is Coca-Cola, value of second element is Pepsi, value of third element is Sprite, value of fourth element is Mountain Dew.

All the above examples, declare an array, initialize the array and also insert elements into the array.

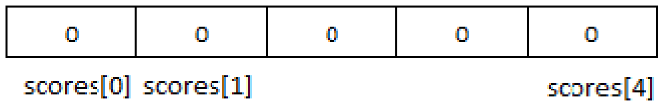
Initializing arrays – using new operator

```
int scores[];
```

The above statement just declares an array. It has not been initialized yet i.e. scores is null. Let us initialize the array.

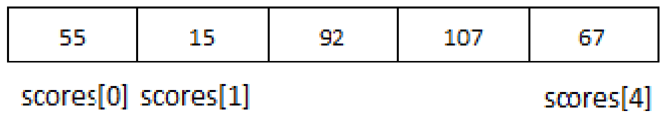
```
int scores[] = new int[5];
```

The above statement now initializes the array to size 5. But no elements are put into array. Hence the elements of the array will have their default value. As the array is of type int, all the 5 elements in the array have the value of 0.



You don't need to mention the length(or size) of an array in [] if you are using the initializer block.

```
int scores[] = new int[]{55,15, 92,107,67};
```



The above statement declares an array, initializes an array (of size 5) and also puts elements into the array. The value of first element is 55, the value of second element is 15, the value of third element is 92, the value of fourth element is 107 and the value of fifth element is 67.

Once the array is initialized, the length(or size) of an array can be known by the length property. It is used after the dot operator on the array variable. Refer the syntax below:

```
array.length;
```

eg.

```
int scores[] = new int[]{55,15, 92,107,67};
System.out.println(scores.length);
```

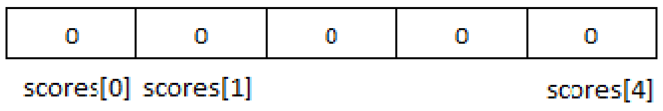
You could either store it in an int variable eg. int size = scores.length;

Putting elements into array:

Step 1:

```
int scores[] = new int[5];
```

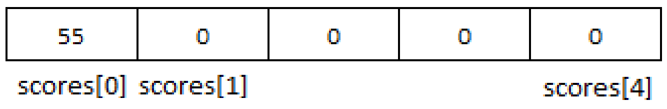
The above statement will declare an int array of length 5. The values in the array will be initialized to the default values as per their data type.



Step 2:

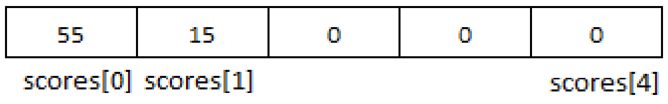
The first value in the array is accessed by the code : scores[0]. To assign values to the first element, use the assignment operator (=). The below statement assigns a value 55 to the first element in the array. The remaining elements still have the default value.

```
scores[0] = 55;
```



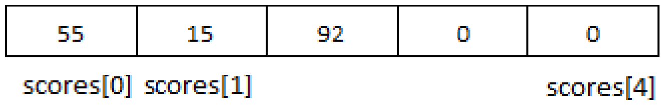
Similarly, to assign value 15 to the second element.

```
scores[1] = 15;
```



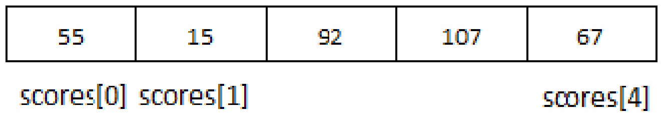
The next line assigns a value 92 to the third element.

```
scores[2] = 92;
```



Accessing elements from the array:

```
int scores[] = new int[]{55,15, 92,107,67};
```



Assuming that the above array is initialized with the given values. The elements of the array are accessed by their index. The first element of the array is stored at index 0 and the last element is stored at index which is length-1.

To retrieve the value of the elements at index 0, use scores[0] which will return the value 55.

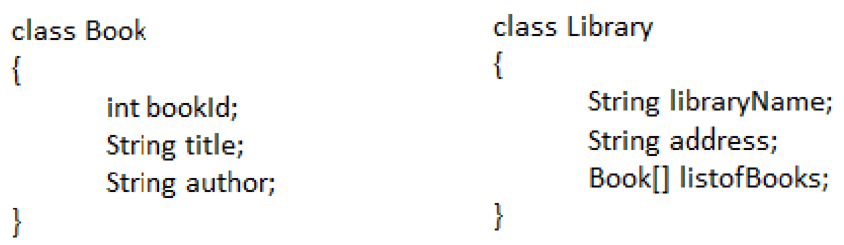
Similarly,

scores[1] will return the value 15
scores[2] will return the value 92
scores[3] will return the value 107
scores[4] will return the value 67

We can use a variable to store the value we retrieve from the array,
int x = scores[0]; as scores is an int array, scores[0] is going to return an int.

Creating array of custom types

Consider the case of library where it has list of books as inventory. So library will have attributes like library name, address. It will also have list of books as an attribute. We could model the Book class and Library class as follow.



```
class Book
{
        int bookId;
        String title;
        String author;
}
```

```
class Library
{
        String libraryName;
        String address;
        Book[] listofBooks;
}
```

The library class has an array of Book, i.e. the array will have multiple objects of Book representing the real books in the library.

- <u>Creating book objects:</u>

  Assuming the Book class has parameterized constructor.

  Book b1 = new Book(101, “Head First Java”, “Author1”);
  Book b2 = new Book(102, “Head First JSP”, “Author2”);
  Book b3 = new Book(103, “Java Complete Reference”, “Author3”);

- <u>Creating book array:</u>

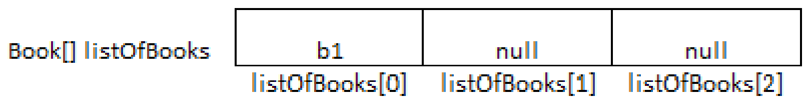
  Book[] listOfBooks;

  The above statement just declares the array variable. It is not initialized so the listOfBooks is still null. Let us initialize the array.
  listOfBooks = new Book[3];

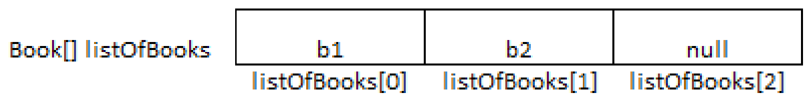
  The above statement just initialized the array itself, the elements have not yet been initialized. Let us initialize the elements of the array by assigning the book object value.

- <u>Putting book objects into the book array</u>

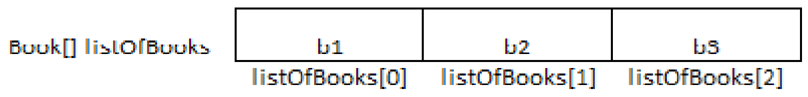
listOfBooks[0] = b1;



listOfBooks[1] = b2;



listOfBooks[2] = b3;



The above 3 statement, initializes all the 3 elements to the value of objects b1, b2 and b3 respectively.

- <u>Retrieving objects from the book array</u>

  listOfBooks[0] will return the objects b1
  listOfBooks[1] will return the objects b2
  listOfBooks[2] will return the objects b3

  We can use a variable to store the value we retrieve from the array.

  Book book = listOfBooks[0]; as listOfBooks is a Book array, listOfBooks[0] is going to return a Book object.

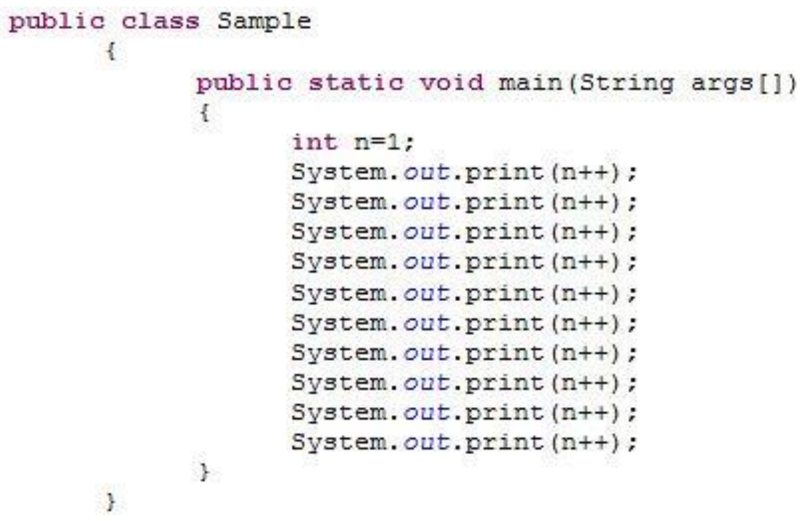
8.2.2 Loops and Iterations

Let us take a scenario where a teacher has to perform task of giving grades to the students according to their percentage. He/she checks the percentage of first student and enters the grade, then checks the percentage of second student and enters the grade, same procedure for the next student as well. Here, the teacher has to perform the same task again and again. Don't you think this is a laborious and error prone work? Yes it is.

One of the things computers are often used for is the automation of repetitive tasks. Repeating identical or similar tasks without making errors is something that computers do well and humans do poorly. In this chapter, we shall study in detail how the repeated tasks can be automated using Java programming.

Let us see a simple java program to print first 10 natural numbers.

Program:



```
public class Sample
      {
            public static void main(String args[])
            {
                  int n=1;
                  System.out.print(n++);
                  System.out.print(n++);
                  System.out.print(n++);
                  System.out.print(n++);
                  System.out.print(n++);
                  System.out.print(n++);
                  System.out.print(n++);
                  System.out.print(n++);
                  System.out.print(n++);
                  System.out.print(n++);
            }
      }
```

Output:

```
12345678910
```

In the above program, we wrote the same line of code 10 times to obtain output. This is nothing but code redundancy(repetition) which makes our program an inefficient one. To eliminate this we use loops and iterations in Java.

Loop is a block of code executed repeatedly until some condition is satisfied and one pass through(execution of) the loop is called **iteration**.

Here condition is a boolean expression. A boolean expression is an expression written using conditional operators(<, >, ==, <=,>= etc) and the result of it will always be either true or false.

Loops in Java are mainly of three types :-

1. 'while' loop
2. 'do while' loop
3. 'for' loop

While loop:

A while loop is a control structure that allows you to repeat a task certain number of times. In simple words, using while loop we can execute a set of statements several times based on a condition.

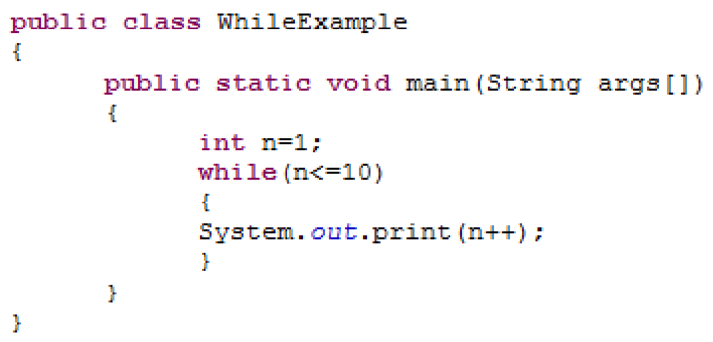
Syntax:

```
while(condition)
{
  //Statements
}
```

Here, “while” is a keyword and “condition” is a boolean expression and as long as this condition yields true, the statements within the braces({}) are executed. You can almost read a while statement as if it were English- “while condition is true, continue executing the Statements”.

Let us now write the same program(print first 10 natural numbers) using while loop.

Program:



```
public class WhileExample
{
	public static void main(String args[])
	{
		int n=1;
		while(n<=10)
		{
		System.out.print(n++);
		}
	}
}
```

Output:

```
12345678910
```

In the above code snippet, we eliminated those 9 repeated lines, clubbed them into a single line and obtained the output using while. Here n is also called **counter variable** and it is used to iterate through the loop.

Flow of control(while loop):

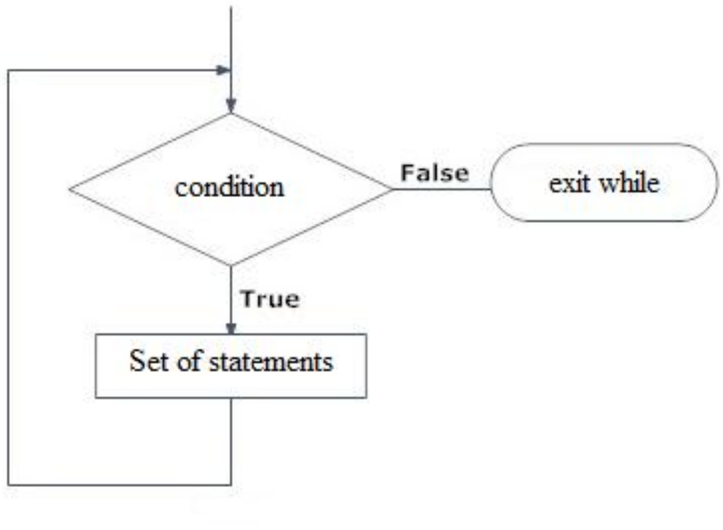


Fig 1:Flowchart of while loop

The flow of execution for a while statement is as follows:

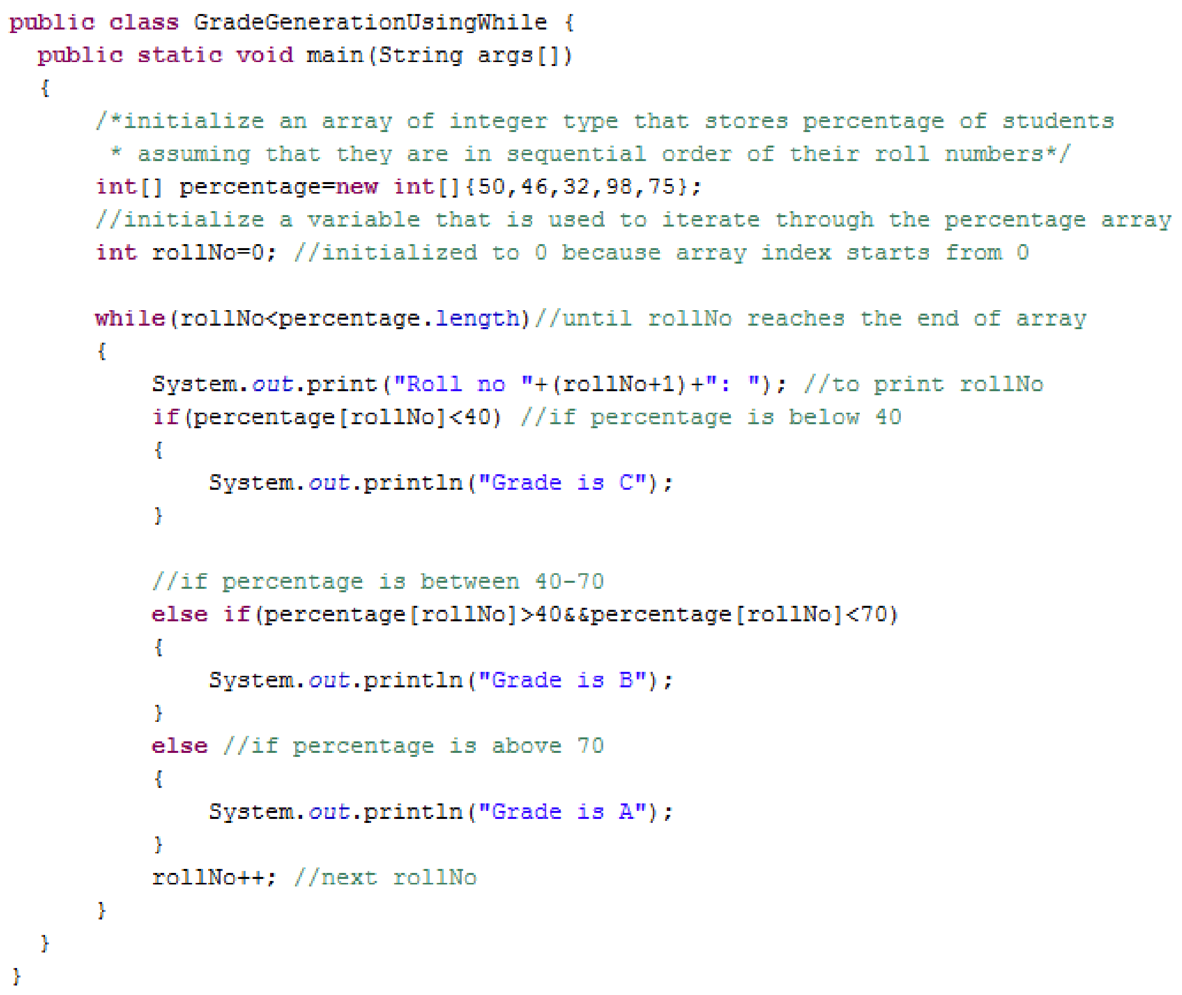
1. Evaluate the condition, whether the result is true or false.

2. If the result is false, exit the while statement and and execute the first statement after the while loop.

3.If the result is true, execute the set of statements within the curly braces, and then go back to step 1.

Example:

/*Program to automate the grade generation of five students based on their percentage using while loop */



```java
public class GradeGenerationUsingWhile {
  public static void main(String args[])
  {
      /*initialize an array of integer type that stores percentage of students
       * assuming that they are in sequential order of their roll numbers*/
      int[] percentage=new int[]{50,46,32,98,75};
      //initialize a variable that is used to iterate through the percentage array
      int rollNo=0; //initialized to 0 because array index starts from 0

      while(rollNo<percentage.length)//until rollNo reaches the end of array
      {
          System.out.print("Roll no "+(rollNo+1)+": "); //to print rollNo
          if(percentage[rollNo]<40) //if percentage is below 40
          {
              System.out.println("Grade is C");
          }

          //if percentage is between 40-70
          else if(percentage[rollNo]>40&&percentage[rollNo]<70)
          {
              System.out.println("Grade is B");
          }
          else //if percentage is above 70
          {
              System.out.println("Grade is A");
          }
          rollNo++; //next rollNo
      }
  }
}
```

Output:

```
Roll no 1: Grade is B
Roll no 2: Grade is B
Roll no 3: Grade is C
Roll no 4: Grade is A
Roll no 5: Grade is A
```

<u>Flow of the above program:</u>

1. Initialize an integer array of percentage.
2. Initialize an integer variable rollNo=0; which is used to iterate through the array.

3. Since we need to generate grades for only those students whose percentage is given, we are specifying the condition in while as rollNo<percentage.length. We have learned in the previous topic that .length property gives the size of array which is 4 in the above program.

rollNo points to the index of the array. So before entering while loop, the value of rollNo is checked.

4. If the condition is satisfied(if rollNo<percentage.length) go to step 5 else go to step 6.
5. Now control is sent to the body of loop
    - In the body of while loop, if-else conditions are specified to check the percentage of student marks. Based on those conditions respective grade is printed.
    - Increment rollNo.
    - Go to Step 4.
6. End.

Do... While loop:

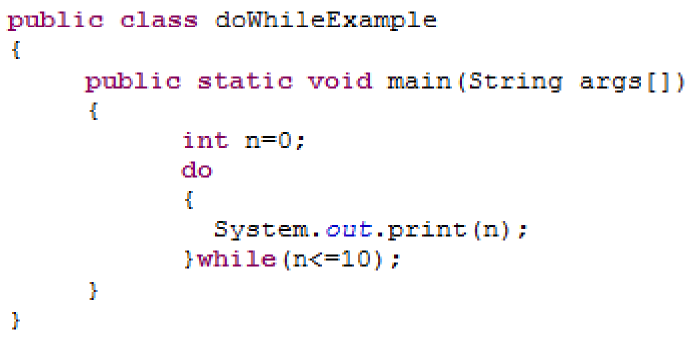
It is very similar to the 'while' loop shown above. The only difference being, in a 'while' loop, the condition is checked beforehand, but in a 'do... while' loop, the condition is checked after one execution of the loop. Hence, the body of do... while loop is executed at least once.

Syntax:

```
do
{
  //statements
}while(condition);
```

Here, "while" and "do" are keywords and "condition" is a boolean expression. Each iteration of the do-while loop first executes the statements and then evaluates the condition. If the result of this condition is true, the loop will repeat. Otherwise, the loop terminates. Notice, there is semicolon at the end of while(); in do... while loop.
Let us now write the same program(print first 10 natural numbers) using do... while loop.

Program:



```
public class doWhileExample
{
     public static void main(String args[])
     {
            int n=0;
            do
            {
              System.out.print(n);
            }while(n<=10);
     }
}
```

Output:

```
12345678910
```

Now you may get a question as why “do.. while” when we already have “while” and when both are giving the same output? We shall study about it in detail.

Why do... while and how it differs from while?

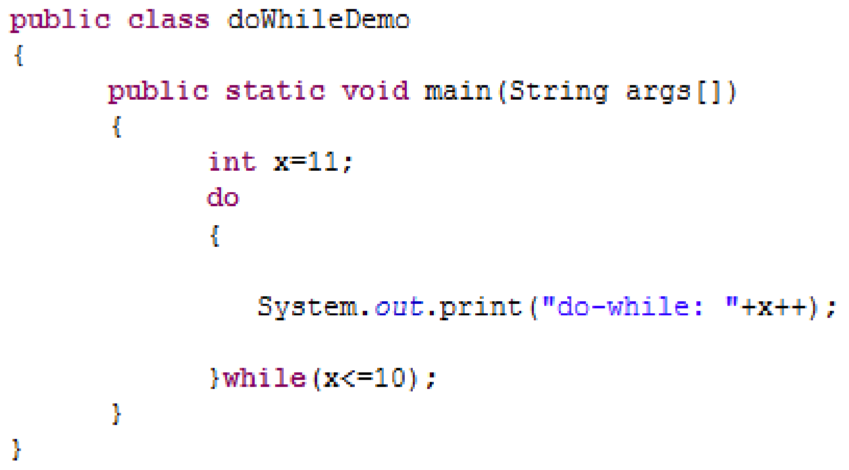
If the condition controlling a while loop is initially false, then the body of the loop will not be executed at all. However, sometimes it is desirable to execute the body of a while loop at least once, even if the conditional expression is false to begin with. In other words, there are times when you would like to test the condition at the end of the loop rather than at the beginning.

Fortunately, Java supplies a loop that does just that: the do-while. The do-while loop always executes its body at least once, because its condition is at the bottom of the loop. Therefore, while is called entry-called loop whereas do...while is called exit-controlled loop.

The difference is best illustrated using the programs below:

/*Program to demonstrate difference between while and do..while*/

Do-while example:

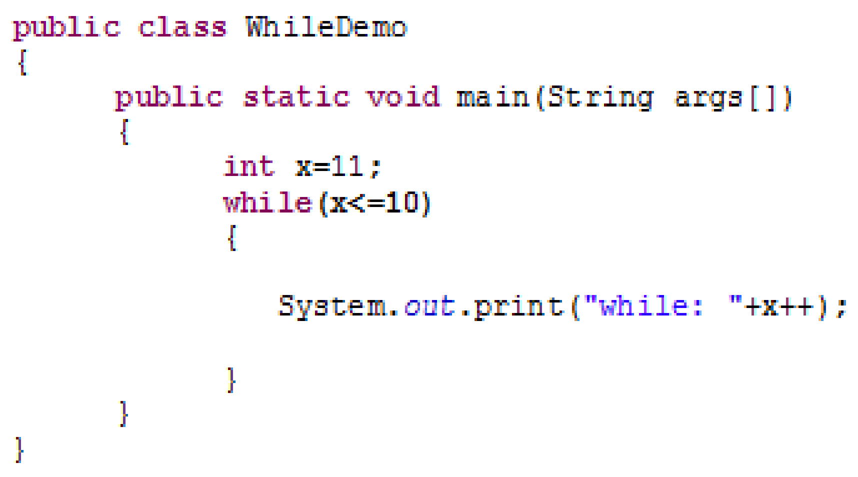


```java
public class doWhileDemo
{
	public static void main(String args[])
	{
		int x=11;
		do
		{

		   System.out.print("do-while: "+x++);

		}while(x<=10);
	}
}
```

Output:

```
do-while: 11
```

While example:



```
public class WhileDemo
{
	public static void main(String args[])
	{
		int x=11;
		while(x<=10)
		{

		   System.out.print("while: "+x++);

		}
	}
}
```

Output:(no output)

Flow of control(do...while loop):

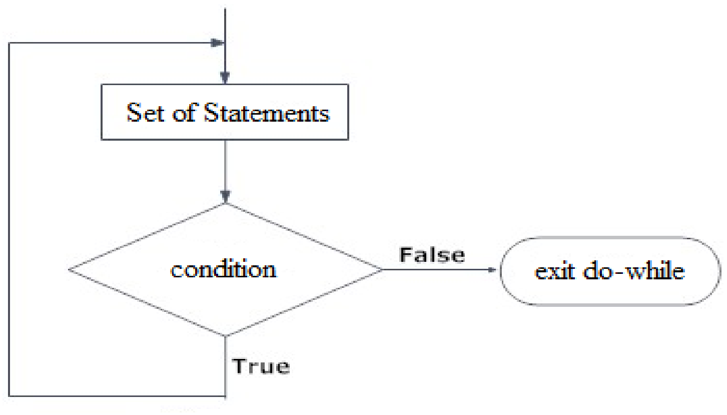


Fig 2:Flowchart of do... while loop

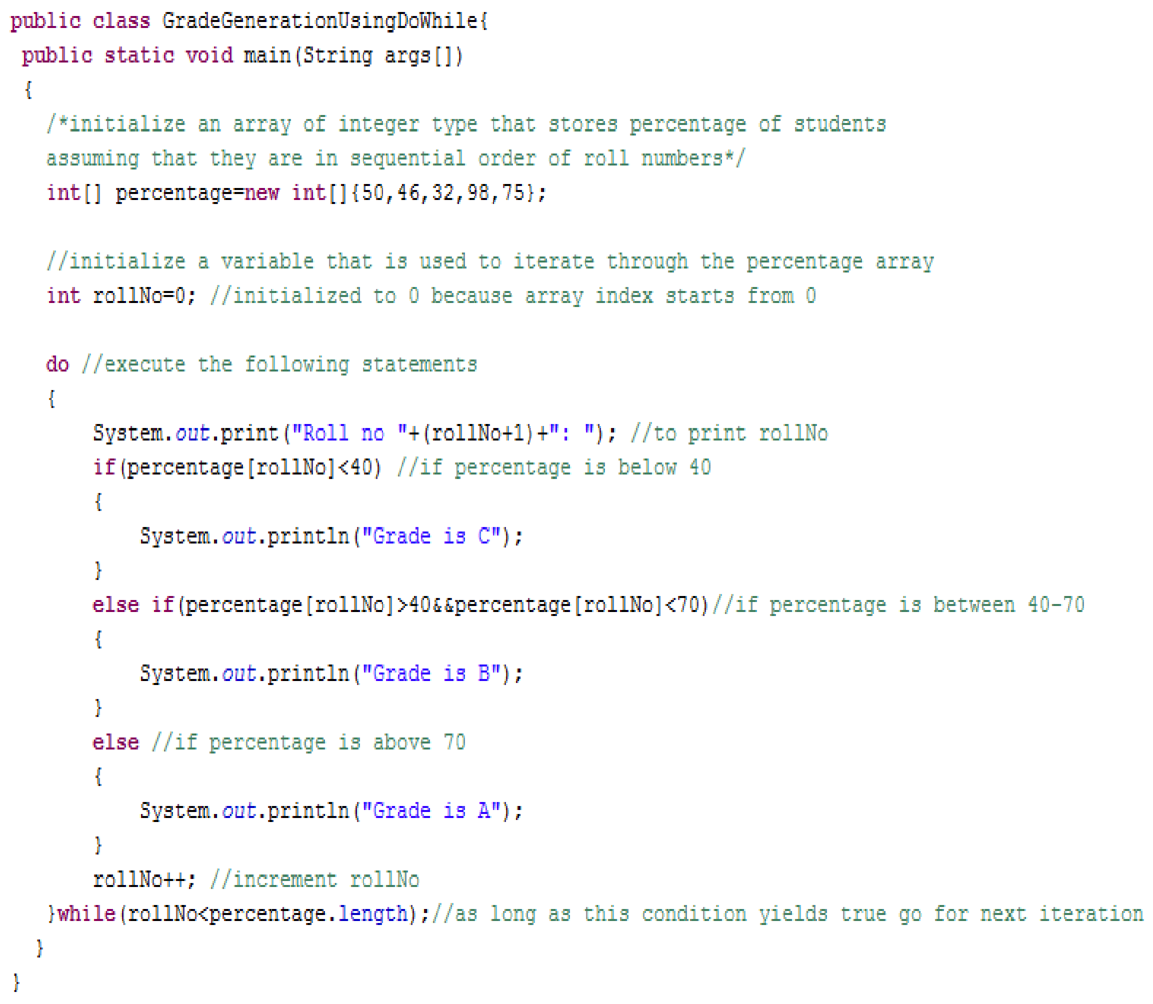
The flow of execution for a do... while statement is as follows:

1. Execute the set of statements.
2. Evaluate the condition, whether the result is true or false.
3. If the result is false, exit the while statement and execute the first statement after the do... while loop.

4. If the result is true, go back to step 1.

Example:

/*Program to generate the grade for five students of a class based on their percentage using do... while loop */



```
public class GradeGenerationUsingDoWhile{
 public static void main(String args[])
 {
   /*initialize an array of integer type that stores percentage of students
   assuming that they are in sequential order of roll numbers*/
   int[] percentage=new int[]{50,46,32,98,75};

   //initialize a variable that is used to iterate through the percentage array
   int rollNo=0; //initialized to 0 because array index starts from 0

   do //execute the following statements
   {
       System.out.print("Roll no "+(rollNo+1)+": "); //to print rollNo
       if(percentage[rollNo]<40) //if percentage is below 40
       {
           System.out.println("Grade is C");
       }
       else if(percentage[rollNo]>40&&percentage[rollNo]<70)//if percentage is between 40-70
       {
           System.out.println("Grade is B");
       }
       else //if percentage is above 70
       {
           System.out.println("Grade is A");
       }
       rollNo++; //increment rollNo
   }while(rollNo<percentage.length);//as long as this condition yields true go for next iteration
  }
}
```

Output:

```
Roll no 1: Grade is B
Roll no 2: Grade is B
Roll no 3: Grade is C
Roll no 4: Grade is A
Roll no 5: Grade is A
```

Flow of the above program:

1. Initialize an integer array of percentage.
2. Initialize an integer variable rollNo=0; which is used to iterate through the array
3. Execute step 4.

4. Now control is sent to the body of loop
   1. In the body of do... while loop, if-else conditions are specified to check the percentage of student marks. Based on those conditions respective grade is printed.
   2. Increment rollNo.
5. Since we need to generate grades for only those students whose percentage is given, we are specifying the condition in while as rollNo<percentage.length. We have learned in the previous topic that .length property gives the size of array which is 4 in the above program. So before proceeding with the next iteration, the value of rollNo is checked.
6. If the condition is satisfied(if rollNo<percentage.length) go to step 3 else go to step 7.
7. End

For loop:

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. A for loop is useful when you know how many times a task is to be repeated.

Syntax:

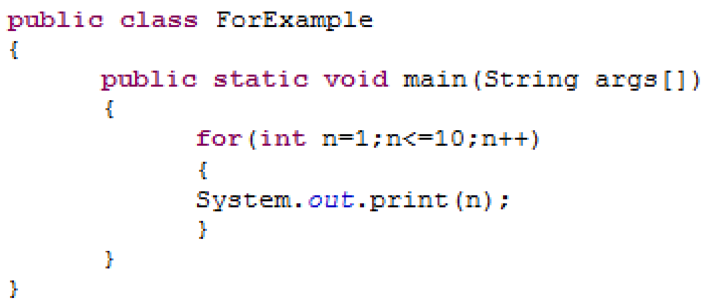
```
for(initialization; condition; increment/decrement)
{
  //Statements
}
```

Here,the initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. Next, the condition is evaluated. If it is true, the statements are executed, else control comes out of for loop and the next statement after the loop is executed.

Increment/decrement is an update statement which gets executed after the execution of body of loop.

Let us now write the same program(print first 10 natural numbers) using for loop.

Program:



```
public class ForExample
{
        public static void main(String args[])
        {
                for(int n=1;n<=10;n++)
                {
                System.out.print(n);
                }
        }
}
```

Output:

```
12345678910
```

Flow of control(for loop):

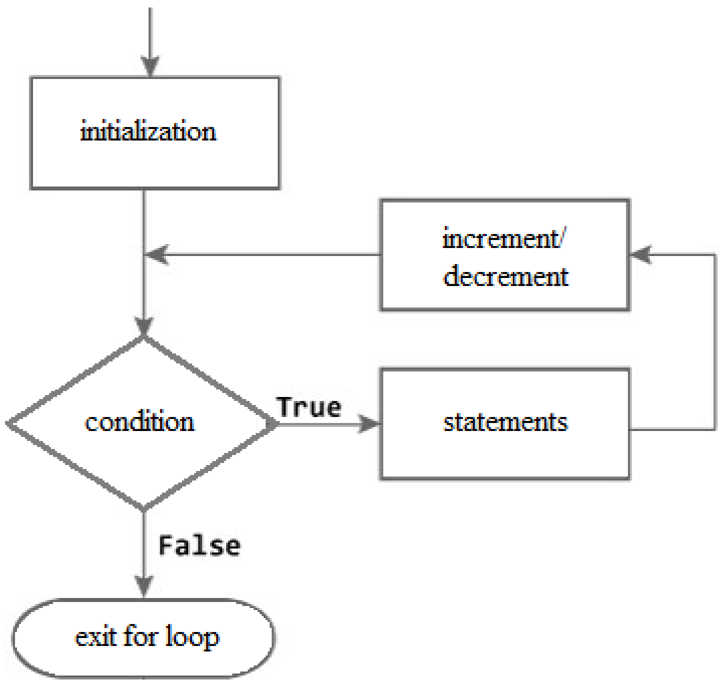


Fig 3:Flowchart of for loop

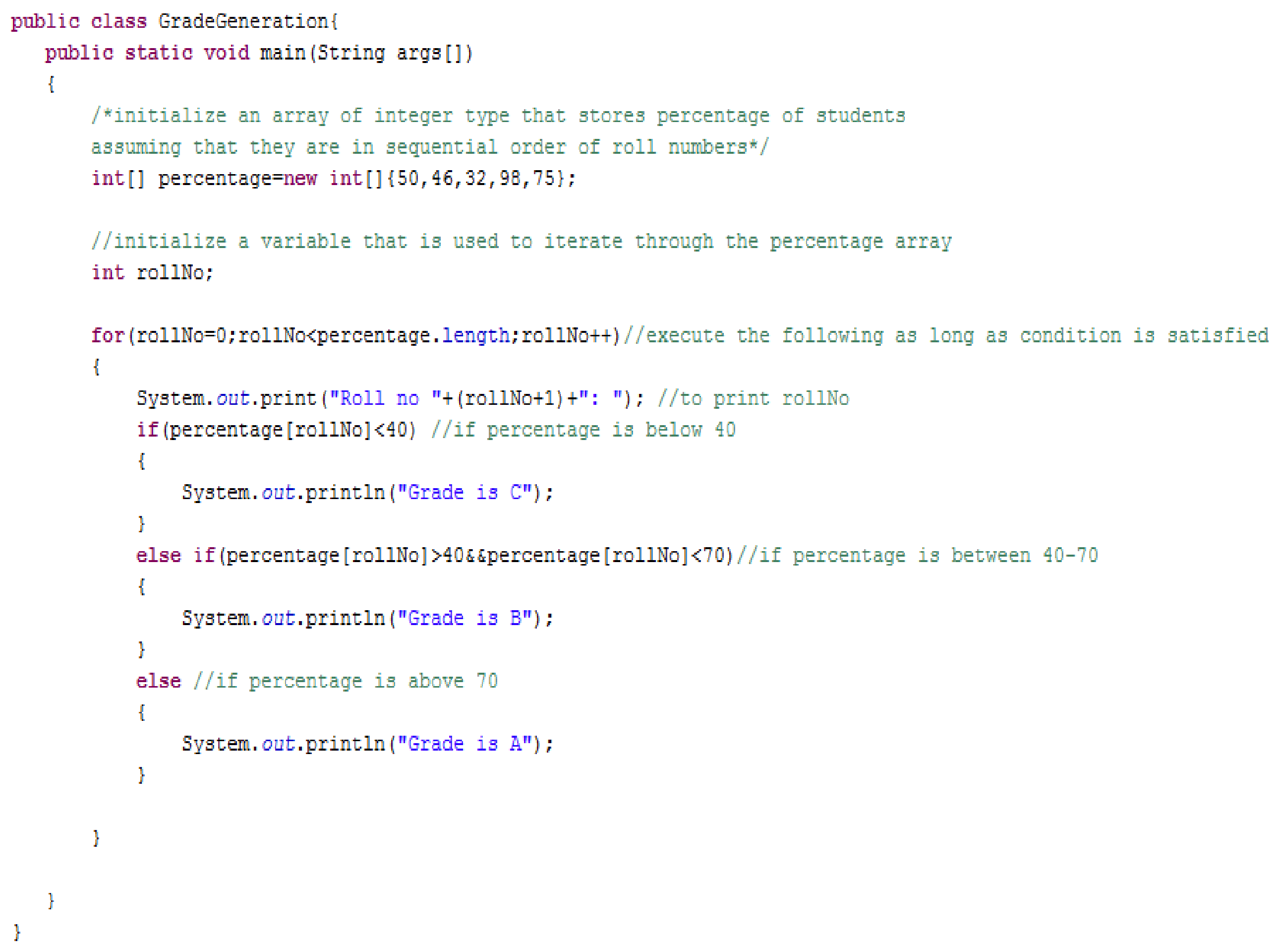
The flow of execution for a while statement is as follows:

1. Initialize the variable. Initialization step is executed first, and only once.
2. Evaluate the condition, whether the result is true or false.
3. If the result is false, exit the for loop and and execute the first statement after the for loop.

4. If the result is true, execute the set of statements within the curly braces, and then go to increment/decrement statement.
5. Go to step 2.

Example:

/*Program to generate the grades for five students based on their percentage using for loop */



```
public class GradeGeneration{
   public static void main(String args[])
   {
       /*initialize an array of integer type that stores percentage of students
       assuming that they are in sequential order of roll numbers*/
       int[] percentage=new int[]{50,46,32,98,75};

       //initialize a variable that is used to iterate through the percentage array
       int rollNo;

       for(rollNo=0;rollNo<percentage.length;rollNo++)//execute the following as long as condition is satisfied
       {
           System.out.print("Roll no "+(rollNo+1)+": "); //to print rollNo
           if(percentage[rollNo]<40) //if percentage is below 40
           {
               System.out.println("Grade is C");
           }
           else if(percentage[rollNo]>40&&percentage[rollNo]<70)//if percentage is between 40-70
           {
               System.out.println("Grade is B");
           }
           else //if percentage is above 70
           {
               System.out.println("Grade is A");
           }

       }

   }
}
```

Output:

```
Roll no 1: Grade is B
Roll no 2: Grade is B
Roll no 3: Grade is C
Roll no 4: Grade is A
Roll no 5: Grade is A
```

Flow of the above program:

1. Initialize an integer array of percentage.
2. Declare an integer variable rollNo; which is used to iterate through the array
3. Initial value of rollNo is 0 and this statement is executed only once in the entire cycle of for loop.
4. Since we need to generate grades for only those students whose percentage is given, we are specifying the condition as rollNo<percentage.length. We have learned in the previous topic that .length property gives the size of array which is 4 in the above program. So before entering into the for loop, the value of rollNo is checked.
5. If the condition is satisfied(if rollNo<percentage.length) go to step 6 else go to step 7.
6. Now control is sent to the body of loop
   i. In the body of while loop, if-else conditions are specified to check the percentage of numbers. Based on those conditions respective grade is printed.
   ii. Increment rollNo.
   iii. Go to Step 5.
7. End.

Advanced for loop:

This is an enhanced for loop which mainly used for arrays. Advanced for loop is also called 'for each' loop.

Syntax:

```
for(declaration : expression)
{
  //Statements
}
```

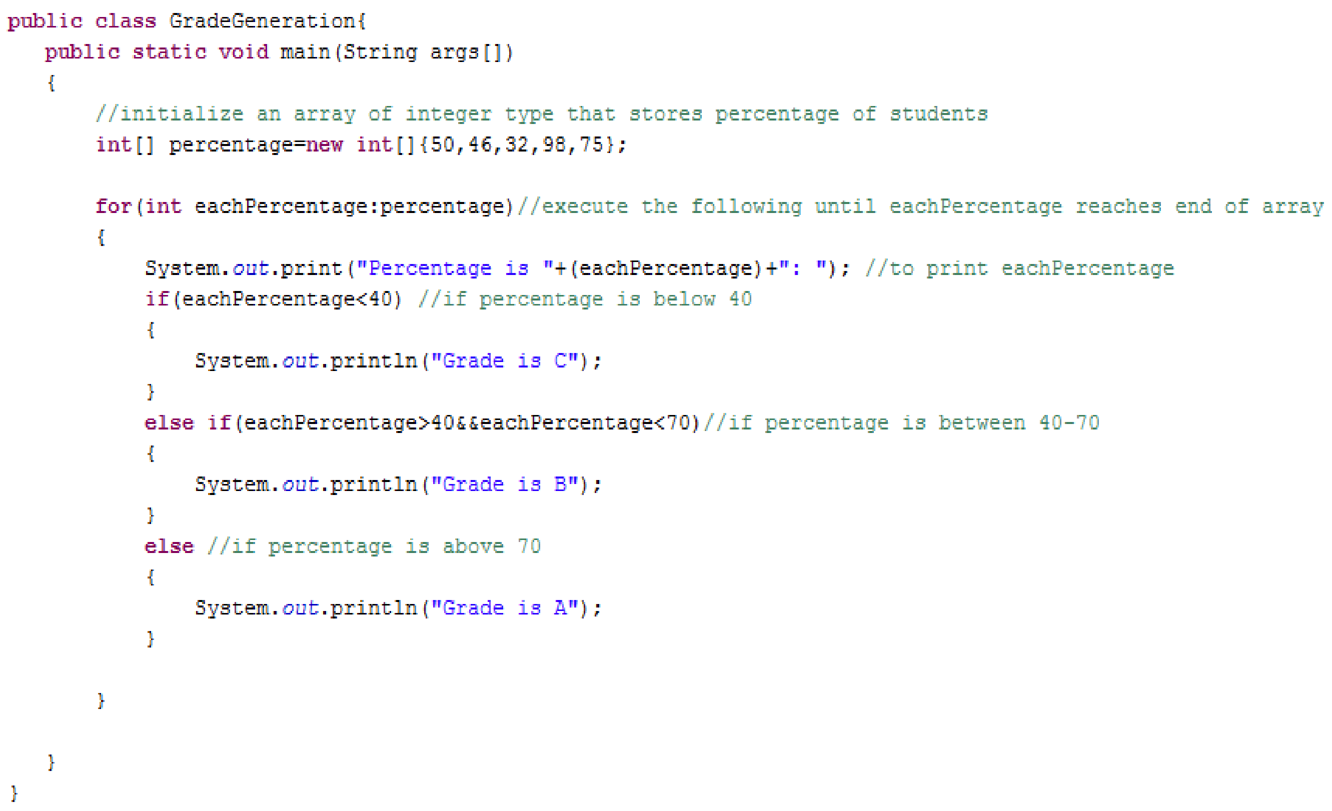
Declaration: The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.

Expression: This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Let us take the same example of grade generation and see how it works in advanced for loop.

Example:

/*Program to generate the grades of 5 students based on their percentage using advanced for loop */



```
public class GradeGeneration{
   public static void main(String args[])
   {
       //initialize an array of integer type that stores percentage of students
       int[] percentage=new int[]{50,46,32,98,75};

       for(int eachPercentage:percentage)//execute the following until eachPercentage reaches end of array
       {
           System.out.print("Percentage is "+(eachPercentage)+": "); //to print eachPercentage
           if(eachPercentage<40) //if percentage is below 40
           {
               System.out.println("Grade is C");
           }
           else if(eachPercentage>40&&eachPercentage<70)//if percentage is between 40-70
           {
               System.out.println("Grade is B");
           }
           else //if percentage is above 70
           {
               System.out.println("Grade is A");
           }

       }

   }
}
```

Output:

```
Percentage is 50: Grade is B
Percentage is 46: Grade is B
Percentage is 32: Grade is C
Percentage is 98: Grade is A
Percentage is 75: Grade is A
```

In the above program, eachPercentage is a variable that is of the same data type as array of percentage. In the previous program rollNo takes the index value whereas here eachPercentage takes value at index. Therefore, initially the value of eachPercentage is percentage[0] which is 50 and in the second iteration eachPercentage will take the value percentage[1] which is 46 and so on.

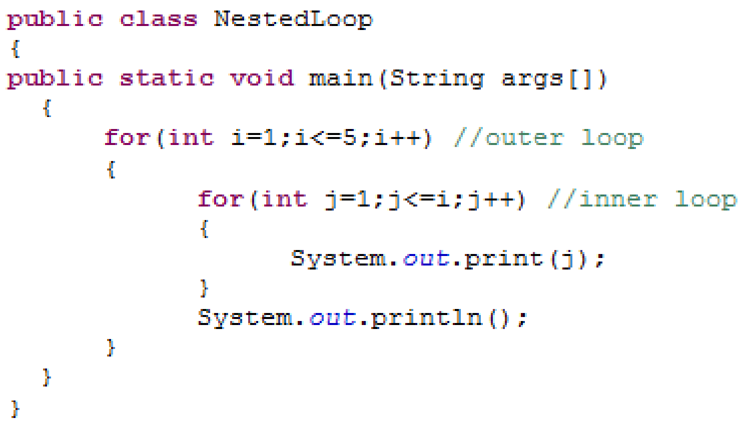
Nested loops:

We have seen the advantages of using various methods of iteration, or looping. Now let's take a look at what happens when we combine looping procedures. The placing of one loop inside the body of another loop is called nesting.

When you "nest" two loops, the outer loop takes control of the number of complete repetitions of the inner loop. While all types of loops may be nested, the most commonly nested loops are for loops.

Let us look at an example on nested loops:

```
/*Program to print the following pattern
  1
  12
  123
  1234
  12345
*/
```



```
public class NestedLoop
{
public static void main(String args[])
  {
      for(int i=1;i<=5;i++) //outer loop
      {
            for(int j=1;j<=i;j++) //inner loop
            {
                  System.out.print(j);
            }
            System.out.println();
      }
  }
}
```

Output:

```
      1
      12
      123
      1234
      12345
```

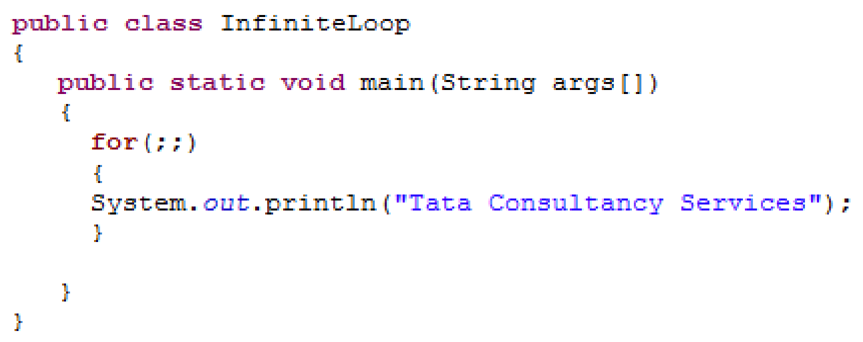
In the above program, outer loop is used to determine the no.of digits to be printed in a line whereas the inner loop is used to print the digits. We have used print() within the body of inner loop because we want the digits to be displayed in the same line and after the inner loop is executed, we go to the next line using println() statement.

8.2.3 Finite and Infinite loops:

A finite loop ends itself i.e control comes out of the loop at certain point of time. For example, the for loop in the previous program that displays multiples of 3 is a finite loop because the loop exits if n>10.

In other case,a loop becomes an infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.

Example for infinite loop:



```
public class InfiniteLoop
{
   public static void main(String args[])
   {
     for(;;)
     {
     System.out.println("Tata Consultancy Services");
     }

   }
}
```

Output:

```
      Tata Consultancy Services
      Tata Consultancy Services
      Tata Consultancy Services
      .
      .
      .
      .
```

This goes on printing until we forcefully stop the execution. The above program is an example of implementing infinite loop using for loop. Similarly, you can implement an infinite loop using while and do... while as follows.

Infinite loop using while:

```
      while(true)
```

```
{
   // your code goes here
}
```

Infinite loop using do... while:

```
do{
   // your code goes here
} while(true);
```

8.2.4 Break Keyword:

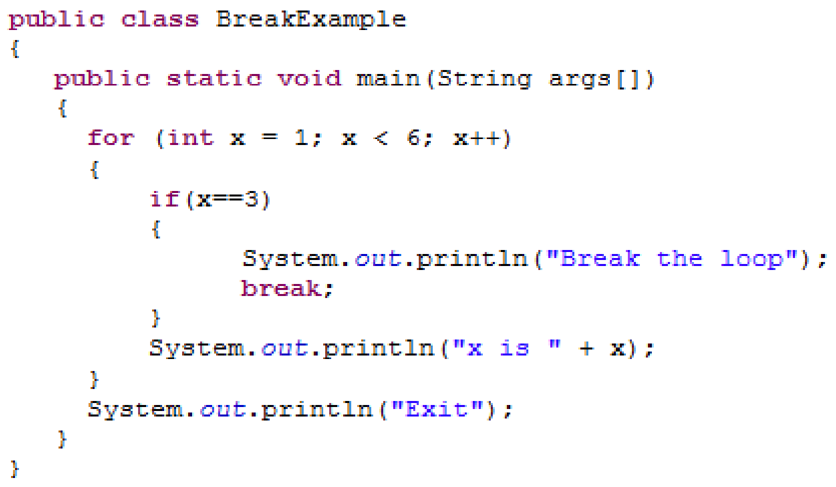
The break keyword is used to stop the entire loop. Using break we can leave a loop even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

Syntax:

```
break;
```

Example: Write a program to print a list of numbers until 3 is encountered

Program:



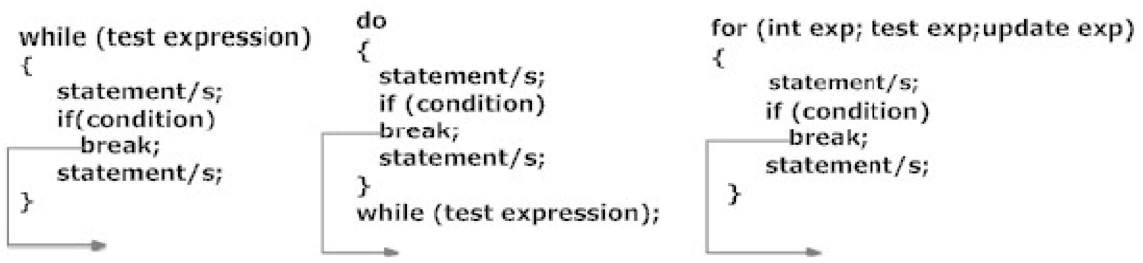
```
public class BreakExample
{
   public static void main(String args[])
   {
     for (int x = 1; x < 6; x++)
     {
         if(x==3)
         {
               System.out.println("Break the loop");
               break;
         }
         System.out.println("x is " + x);
     }
     System.out.println("Exit");
   }
}
```

Output:

```
x is 1
x is 2
Break the loop
Exit
```

In the above example if the value of x is 3 then the control enters into the loop and when break statement is encountered, it comes out of the for loop and executes the first statement after the for loop. The break statement can be used in terminating all three loops for, while and do...while.

Working of break statement:



```
while (test expression)
{
   statement/s;
   if(condition)
     break;
   statement/s;
}
```

```
do
{
  statement/s;
  if (condition)
  break;
  statement/s;
}
while (test expression);
```

```
for (int exp; test exp;update exp)
{
    statement/s;
    if (condition)
      break;
    statement/s;
 }
```

Fig 4: Working of break statement in different loops.

8.2.5 Continue Keyword:

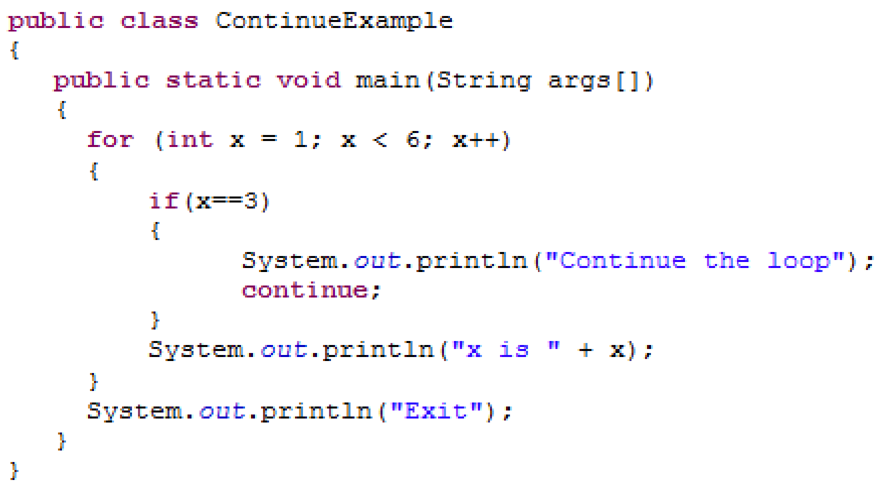
The continue keyword can be used in any of the loop control structures. Continue statement skips the current iteration of a for, while or do while loop. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes flow of control to immediately jump to the increment/decrement statement.
- In a while loop or do/while loop, flow of control immediately jumps to the condition.

Syntax:

continue;

Example: Write a program to print numbers between 1 to 5 except 3.



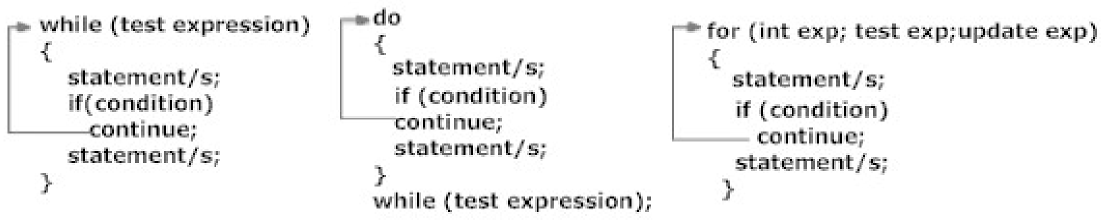
```
public class ContinueExample
{
   public static void main(String args[])
   {
     for (int x = 1; x < 6; x++)
     {
         if(x==3)
         {
               System.out.println("Continue the loop");
               continue;
         }
         System.out.println("x is " + x);
     }
     System.out.println("Exit");
   }
}
```

Output:

```
x is 1
x is 2
Continue the loop
x is 4
x is 5
Exit
```

In the above example if the value of x is 3 then the control enters into the loop and when continue statement is encountered, it jumps to the next iteration without executing the rest of the statements within the loop.

Working of continue statement:



```
while (test expression)
{
    statement/s;
    if(condition)
      continue;
    statement/s;
}
```

```
do
{
   statement/s;
   if (condition)
   continue;
   statement/s;
}
while (test expression);
```

```
for (int exp; test exp;update exp)
{
   statement/s;
   if (condition)
     continue;
   statement/s;
 }
```

Fig 5: Working of continue statement in different loops.

8.2.6 Practice Problems on arrays using iterations:

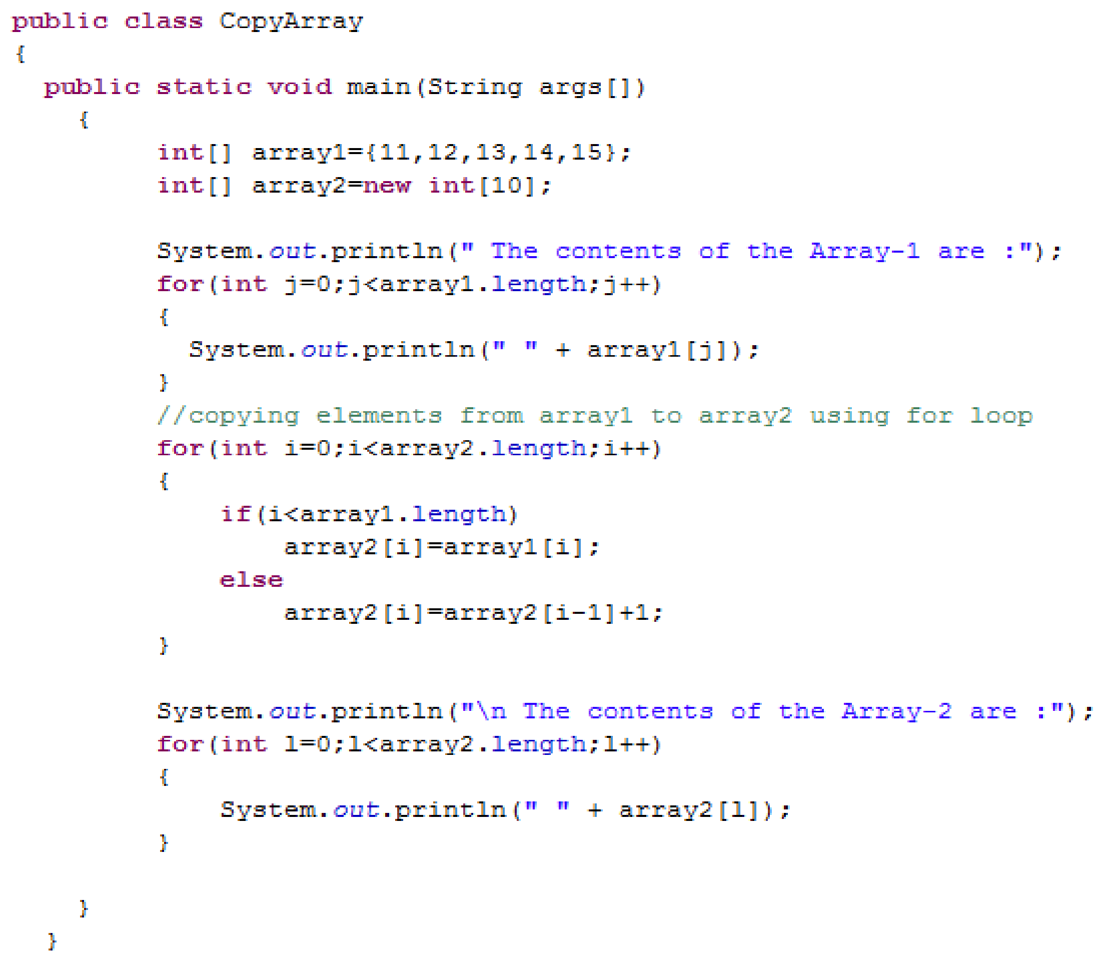
1./*Program to search for an element in the given array*/

In the program below we are searching for 90.

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

  int numbers[] = new int[]{55,15,92,107,67}; //initializing an array
  boolean isFound=false;
  int i=0;


  for(i=0;i<numbers.length;i++) //iterating through array
  {
    if(numbers[i]==90) //if number is present do the following
    {
      isFound=true;
      break;                //break the loop
    }
  }//end for


  if(isFound==true)
  {
    System.out.println("90 is found at position: "+(i));
  }
  else
  {
    System.out.println("Element is not present in the given set of numbers");
  }

 }
}
```

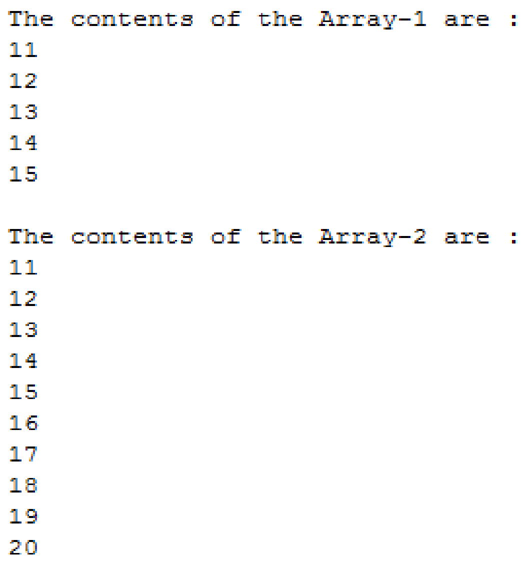
Output: Element is not present in the given set of numbers.

2./*Program to copy elements from one array to the other using for loop*/



```
public class CopyArray
{
  public static void main(String args[])
    {
         int[] array1={11,12,13,14,15};
         int[] array2=new int[10];

         System.out.println(" The contents of the Array-1 are :");
         for(int j=0;j<array1.length;j++)
         {
           System.out.println(" " + array1[j]);
         }
         //copying elements from array1 to array2 using for loop
         for(int i=0;i<array2.length;i++)
         {
             if(i<array1.length)
                 array2[i]=array1[i];
             else
                 array2[i]=array2[i-1]+1;
         }

         System.out.println("\n The contents of the Array-2 are :");
         for(int l=0;l<array2.length;l++)
         {
             System.out.println(" " + array2[l]);
         }

    }
  }
```

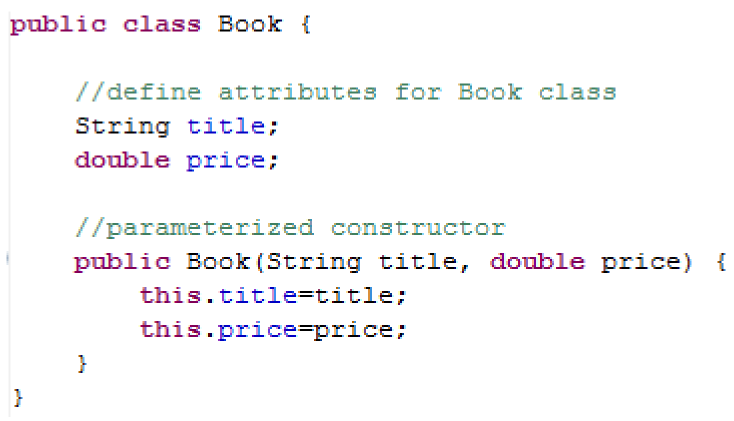
Output:



```
The contents of the Array-1 are :
11
12
13
14
15

The contents of the Array-2 are :
11
12
13
14
15
16
17
18
19
20
```

3. Practice problems on arrays using iterations:

```
/*Program to find the costliest book in a library*/

 public class Library {

  //defining the attributes of Library
  String libraryName;
  String address;
  Book[] listOfBooks;

  //parameterized constructor
  public Library(String libraryName, String address, Book[] listOfBooks) {

   this.libraryName = libraryName;
   this.address = address;
   this.listOfBooks = listOfBooks;
  }

  //method to find the costliest among all the books present in library
  public Book getCostliestBook()
  {
    Book costliestBook=listOfBooks[0];//assuming that first book is costliest book
    for(int i=0;i<listOfBooks.length;i++)//iterating through array of Books
    {
       //comparing the price of one book with that of the other
      if(listOfBooks[i].price>costliestBook.price)// checking if the book is
                                                  costlier than the previous one
      {
        costliestBook=listOfBooks[i];//costliestBook holds a new value
      }
    }
   return costliestBook; //return costliest book
  }

  public static void main(String args[])
  {
   Book[] listOfBooks=new Book[3];//declaring array of Books
   //creating book objects
   Book b1 = new Book("Head First Java",550.0);
   Book b2 = new Book("Head First JSP", 980.0);
   Book b3 = new Book("Java Complete Reference", 500.0);
   //putting book objects into the book array
   listOfBooks[0]=b1;
   listOfBooks[1]=b2;
   listOfBooks[2]=b3;
   //creating library object
   Library library=new Library("Universal","Gandhinagar",listOfBooks);
  //calling getCostliestBook() method which returns an object of type Book
   Book costliestBook=library.getCostliestBook();
   //print the details of costliest book
   System.out.println("Costliest Book in the library is: "+costliestBook .title+"
                with cost of rupees "+costliestBook .price);
  }
 }
```



```
public class Book {

    //define attributes for Book class
    String title;
    double price;

    //parameterized constructor
    public Book(String title, double price) {
        this.title=title;
        this.price=price;
    }
}
```

Output: `Costliest Book in the Library is: Head First JSP with cost of rupees 980.0`

8.2.7 Points to remember:

8.2.7.1 Arrays

- ✓ Arrays are objects(not primitive) in Java.
- ✓ You create an object of array either by using the initializer block {} or by using the new operator.
- ✓ Arrays have a constant length, i.e. the length cannot increase dynamically at run time.
- ✓ The first index of an array is 0 and the last index is length-1. eg. if the array declared is of size 4, the values are stored at index 0 to 3.
- ✓ The property length is used to find the length(or size) of an array.
- ✓ If the elements in an array are not assigned any value, the value will be the default value depending on the type of array. eg. int[] will have value 0 at index where the value is not initialized, String[] will have value null at index where the value is not initialized.

8.2.7.2 Iterations

- ✓ Iterations are used to execute repetitive tasks.
- ✓ Condition in the while,do... while and for should give result as false after several iterations to terminate the loop.

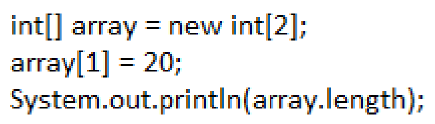
- ✓ A loop control variable used to count the iterations is called counter variable.
- ✓ Do not forget to update the counter variable. Ex:n++,i-- etc.
- ✓ In do while loop, there is a semicolon after while statement.
- ✓ In for each loop, declared variable and array should be of same data type.
- ✓ The **break** statement would stop the execution.
- ✓ The **continue** statement skips the current iteration.

8.3 Test Your Knowledge

1. Which of the following statement declares an int array of size 10?
    [a] int[10] array
    [b] int array[10]
    [c] int[] array = new int[10]
    [d] int[10] array = new int[10]
    [e] int[] array = new int[10]{}

2. Is following declaration valid: int[] array = new int[2]{10,20}

    [a] Yes
    [b] No, size should be defined on the left side
    [c] No, Cannot define dimension expressions when an array initializer is provided



```
int[] array = new int[2];
array[1] = 20;
System.out.println(array.length);
```

3. What will be output of following code?
    [a] 2
    [b] 1
    [c] 0

4. A continue statement causes execution to skip to
    [a] the end of the program
    [b] the first statement after the loop

[c] the statement following the continue statement

[d] the next iteration of the loop

5. In a group of nested loops, which loop is executed the most number of times?

   [a] the outermost loop

   [b] the innermost loop

   [c] all loops are executed the same number of times

   [d] cannot be determined without knowing the size of the loops

6. A break statement causes execution to skip to

   [a] the end of the program

   [b] the statement following the break statement

   [c] the first statement after the loop

   [d] the next iteration of the loop

Answers:

1. [c]
2. [c]
3. [a]
4. [d]
5. [b]
6. [c]