switch

- switch (variable/expression)
- {
- case value1: // statements of case1
- break;
- case value2: // statements of case2
- break; .. .. … .. .. …
- default: // default statements
- }

Facts about switch

- Expression can only be **char, byte, short, int, Character, Byte, Short, Integer, String, or an enum type** other wise a **compile-time error** occurs.
- According to the specification followings are also must be true:
- No two of the case constant expressions associated with a switch statement may have the same value.
- No switch label is null.
- switch expression can't be float, double or boolean.
- boolean true false are meaningful using with if-else, e.g., if(true) then do.
- Floating point numbers (float, double) are not a good candiadtes for switch as exact comparison is often broken by rounding errors. e.g. 0.11 - 0.1 == 0.01 is false.
-

Loop is

- Execution of a set of instructions/functions repeatedly ,while some condition evaluates to true.
- There are three types of loops in Java.
- for loop
- while
- do while

Introduction	The Java for loop is a control flow statement that iterates a part of the programs multiple times.	The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Syntax	`for(init;condition;incr/decr){`
	`// code to be executed`
	`}`
		`while(condition){`
		`//code to be executed`
		`}`
			`do{`
			`//code to be executed`
			`}while(condition);`

Example	`//for loop`
	`for(int i=1;i<=10;i++){`
	`System.out.println(i);`
	`}`
		`//while loop`
		`int i=1;`
		`while(i<=10){`
		`System.out.println(i);`
		`i++;`
		`}`
			`//do-while loop`
			`int i=1;`
			`do{`
			`System.out.println(i);`
			`i++;`
			`}while(i<=10);`

Syntax for infinitive loop	`for(;;){`
	`//code to be executed`
	`}`
		`while(true){`
		`//code to be executed`
		`}`
			`do{`
			`//code to be executed`
			`}while(true);`

while

- **while**(condition)
- {
- //code to be executed
- }

==========

```
1.  public class Example
2.  {
3.  public static void main(String[] args)
4.  {
5.      int i=1;
6.      while(i<=10)
7.     {
8.      System.out.println(i);
9.      i++;
10.     }
11. }
12. }
```

do while

- do
- {
- statements..;
- } while (condition);

```
1. class Demo
2.  {
3.     public static void main(String args[])
4.     {
5.       int x = 21;
6.       do
7.       {
8.       System.out.println("Value of x:" + x);
9.       x--;
10.      }
11.        while (x < 20);
12.    }
13.}
```

- **for**(initialization;condition;incr/decr)
- {
- //statement or code to be executed
- }
-

```
1. public class ForExample
2.  {
3. public static void main(String[] args)
4. {
5.     //Code of Java for loop
6.     for(int i=1;i<=10;i++)
7.    {
8.        System.out.println(i);
9.     }
10.}
11.}
```

Table using different loops

- 2*1 = 2
- 2*2 = 4
- 2*3 = 6
- 2*4 = 8
- 2*5 = 10
- 2*6 = 12
- 2*7 = 14
- 2*8 = 16
- 2*9 = 18
- 2*10 = 20

While table

```
public class table
{
 public static void main(String[] args)
 {
 int n, i=1;
 n=2;
while (i<=10)
 {
 System.out.println(   n+ "*“  +i+  "=“  + n*i);
 i++;
 }
}
}
```

Do while table

```
public class table
{
 public static void main(String[] args)
 {
 int n, i=1;
 n=2;
do
{
System.out.println(n+ "*“ +i+ "=“ + n*i);
i++;
 }
while(i<=10);
}
}
```

For table

```
public class table
{
 public static void main(String[] args)
 {
 int n, i;
 n=2;
for(i=1;i<=10;i++)
{
 System.out.println(n+ "*“ +i+ "=“ + n*i);
 }
}
}
```

- For()//rows
- {
- For()//cols
- {
- }
- }
- R=1,c=1,2,3
- R=2, c=1,2,3

R=3,c=1,2,3

For table

```
public class table
{
 public static void main(String[] args)
 {
 int n, i;
 n=2;
for(i=1;i<=10;i++)
{
 System.out.println(n+ "*“ +i+ "=“ + n*i);
 }
}
}
```

Nested for

```
public class NestedForExample
{
public static void main(String[] args) {
//loop of i
for(int i=1;i<=3;i++){
//loop of j
for(int j=1;j<=3;j++){
        System.out.println(i+" "+j);
}//end of i
}//end of j
}
}
```

for-each Loop Sytnax

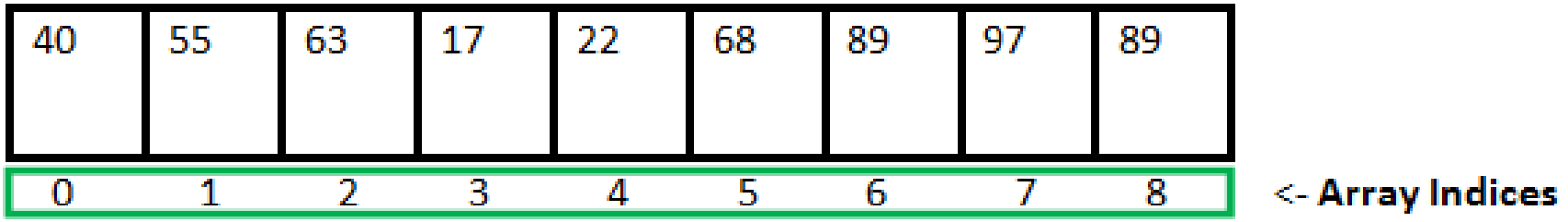
- The syntax of the Java **for-each** loop is:
- for(dataType item : array)
- { ... }

- Here,
- **array** - an array or a collection
- **item** - each item of array/collection is assigned to this variable
- **dataType** - the data type of the array/collection
-

Array

- Def, syntax,memrepresent
- Declare, initialize,disp
- Accept , disp
- Collection of elements but similar datatypes
- Int a[]=

Array Representation



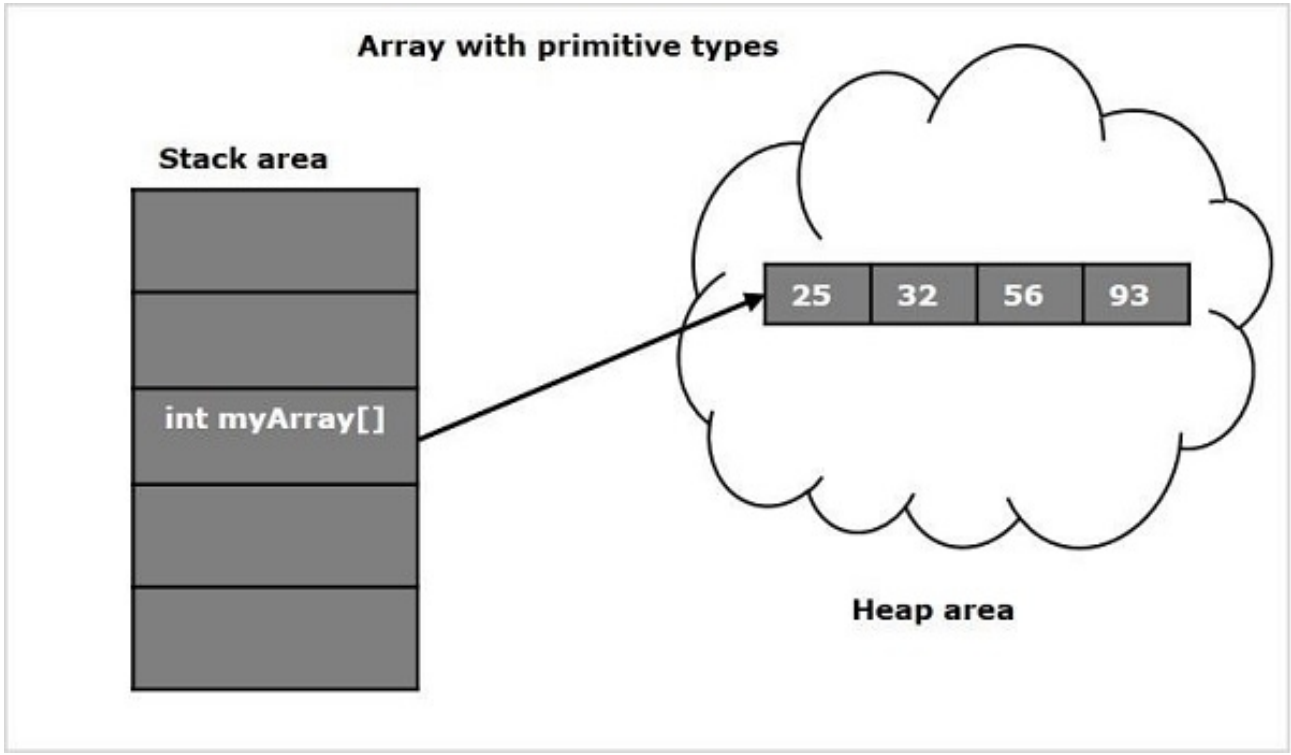
Array Length = 9
First Index = 0
Last Index = 8

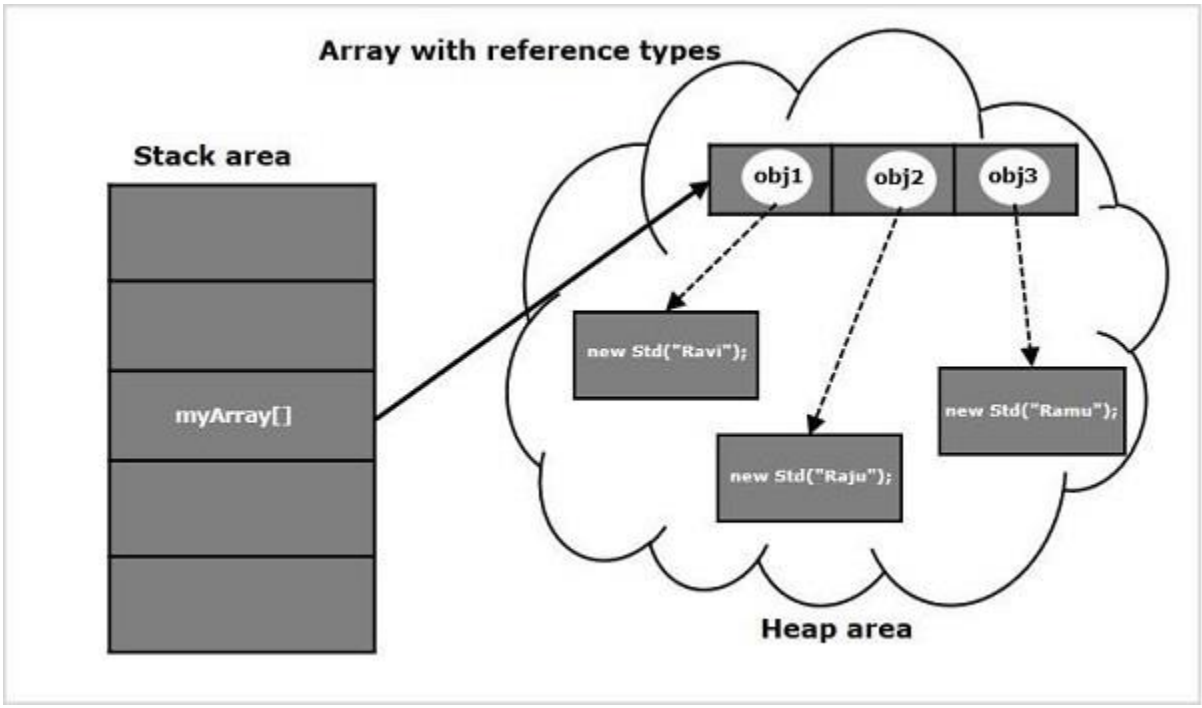
- System.out.println(a[0]);
- For(i=0;i<=4;i++)
- {
- System.out.println(a[i]);

- }

JVM memory locations

- JVM has five memory locations namely −
- **Heap** − Runtime storage allocation for objects (reference types).
- **Stack** − Storage for local variables and partial results. A stack contains frames and allocates one for each thread. Once a thread gets completed, this frame also gets destroyed. It also plays roles in method invocation and returns.
- **PC Registers** − Program Counter Registers contains the address of an instruction that JVM is currently executing.
- **Execution Engine** − It has a virtual processor, interpreter to interpret bytecode instructions one by one and a JIT, just in time compiler.
- **Native method stacks** − It contains all the native methods used by the application.
-


Array with primitive types
Stack area
int myArray[]
25
32
56
93
Heap area


Array with reference types
Stack area
myArray[]
obj1
obj2
obj3
new Std("Ravi");
new Std("Raju");
new Std("Ramu");
Heap area

- String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
- int[] myNum = {10, 20, 30, 40};
- String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
- System.out.println(cars[0]);

- **Declaration**
- dataType[] arr;

- dataType []arr;
-
- dataType arr[];

- **Instantiation of an Array in Java**

- var=**new** datatype[size];

Initialization

```
class Testarray
{
public static void main(String args[])
{
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length();i++)
{
 System.out.println(a[i]);
}
}
```

```
class Testarray1
{
public static void main(String args[])
{
int a[]={33,3,4,5};

for(int i=0;i<a.length;i++)
{
System.out.println(a[i]);
}
}
```

Var=arrayname.length

length & length()

- length attribute is applicable to array
- the length() method is applicable for string objects but not for arrays.
- **Length** can be used for int[], double[], String[]
- int a[]=new int[5];
- a.length(); doesn’t work

```
public class Test
 {
public static void main(String[] args)
    {
      int[] array = new int[4];
   System.out.println(“Array size  " + array.length);

     String str = “Hello Edac";
     System.out.println("String size " + str.length());
    }
}
```

String doesn’t support length

- String[] str = { “ABC", "FOR” ,”POM" };
- System.out.println(str[0].length);
- }
- }
- Output:

- error: cannot find symbol
- symbol: method length()
- location: variable str of type String[]
- Explanation: Here the str is an array of type string and that’s why str.length() CANNOT be used to find its length.

ArrayIndexOutOfBoundsException

- The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException
- if length of the array in negative,
- equal to the array size or greater than the array size while traversing the array.
-