Core Java

AJ

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1.Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

2.Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays

Java Primitive Data Types

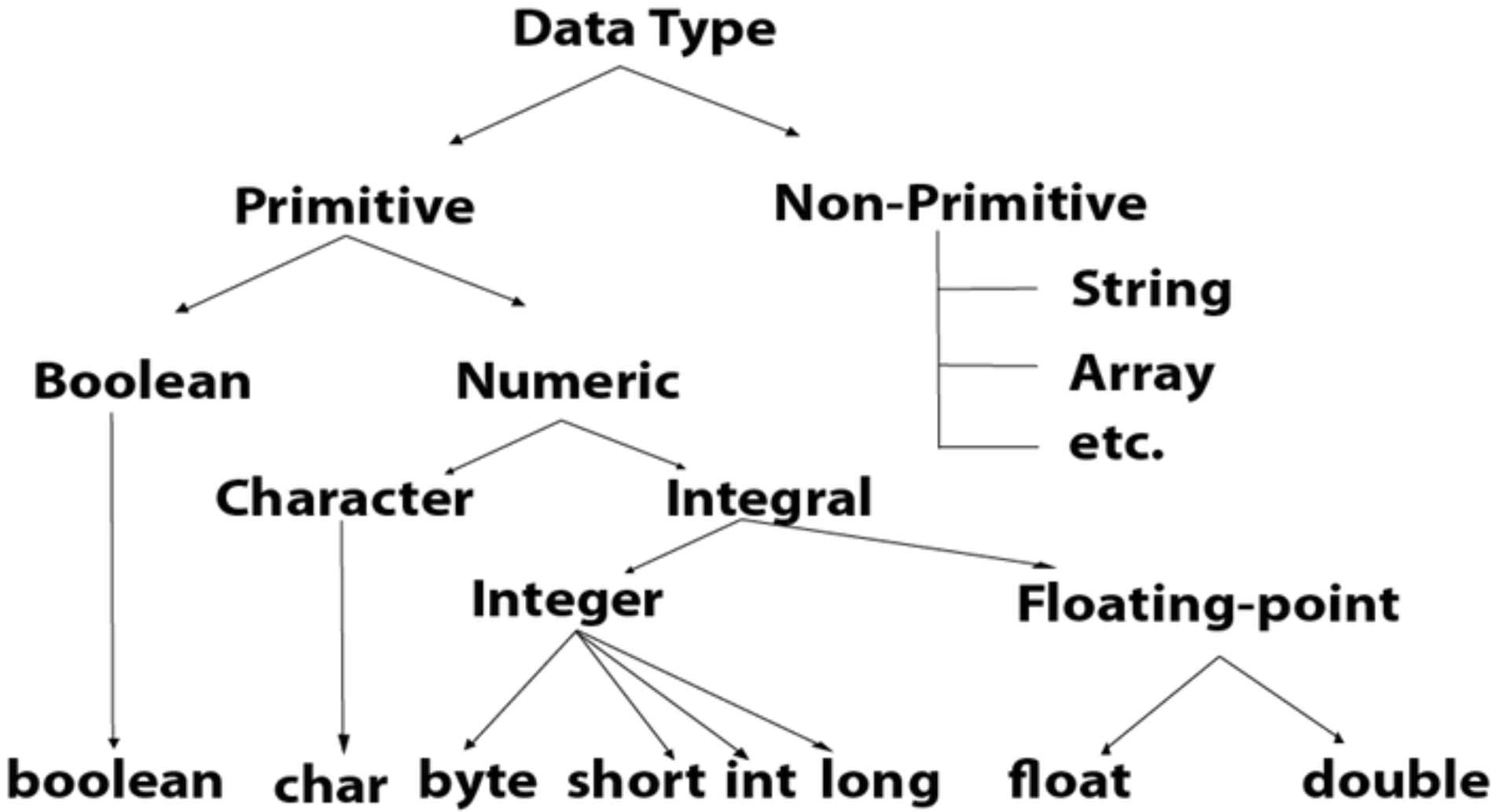
In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Data Types in Java



Data Types in Java

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.
There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Unary Operator Example: ++ and --

Java Unary Operator Example: ~ and !

```
1.class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=-10;
5.boolean c=true;
6.boolean d=false;
7.System.out.println(~a);//-
11 (minus of total positive value which starts from 0)
8.System.out.println(~b);//9 (positive of total minus, positive st
arts from 0)
9.System.out.println(!c);//false (opposite of boolean value)
10.System.out.println(!d);//true
11.}}
```

Operators in java

Java Arithmetic Operators

Java arithmatic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

```
1.class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=5;
5.System.out.println(a+b);//15
6.System.out.println(a-b);//5
7.System.out.println(a*b);//50
8.System.out.println(a/b);//2
9.System.out.println(a%b);//0
10.}}
```

Operators in java

Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example

```
1.class OperatorExample{
2.public static void main(String args[]){
3.System.out.println(10<<2);//10*2^2=10*4=40
4.System.out.println(10<<3);//10*2^3=10*8=80
5.System.out.println(20<<2);//20*2^2=20*4=80
6.System.out.println(15<<4);//15*2^4=15*16=240
7.}}
```

Java Right Shift Operator

The Java right shift operator >> is used to move left operands value to right by the number of bits specified by the right operand.

Java Right Shift Operator Example

```
1.class OperatorExample{
2.public static void main(String args[]){
3.System.out.println(10>>2);//10/2^2=10/4=2
4.System.out.println(20>>2);//20/2^2=20/4=5
5.System.out.println(20>>3);//20/2^3=20/8=2
6.}}
```

Operators in java

Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
1.class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=5;
5.int c=20;
6.System.out.println(a<b&&a<c);//false && true = false
7.System.out.println(a<b&a<c);//false & true = false
8.}}
```

Java AND Operator Example: Logical && vs Bitwise &

```
1.class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=5;
5.int c=20;
6.System.out.println(a<b&&a++<c);//false && true = false
7.System.out.println(a);//10 because second condition is not ch
ecked
8.System.out.println(a<b&a++<c);//false && true = false
9.System.out.println(a);//11 because second condition is check
ed
10.}}
```

Operators in java

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
1.class OperatorExample{
2.public static void main(String args[]){
3.int a=10;
4.int b=5;
5.int c=20;
6.System.out.println(a>b||a<c);//true || true = true
7.System.out.println(a>b|a<c);//true | true = true
8.//|| vs |
9.System.out.println(a>b||a++<c);//true || true = true
10.System.out.println(a);//10 because second condition is not checked
11.System.out.println(a>b|a++<c);//true | true = true
12.System.out.println(a);//11 because second condition is checked
13.}}
```

Operators in java

Java Ternary Operator

Java Ternary operator is used as one liner replacement for if-then-else statement and used a lot in java programming. it is the only conditional operator which takes three operands.

Java Ternary Operator Example

```
1.class OperatorExample{
2.public static void main(String args[]){
3.int a=2;
4.int b=5;
5.int min=(a<b)?a:b;
6.System.out.println(min);
7.}}
```

Java Assignment Operator

Java assignment operator is one of the most common operator. It is used to assign the value on its right to the operand on its left.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Operators in java

Java Assignment Operator Example: Adding short

```
class OperatorExample{
public static void main(String args[]){
short a=10;
short b=10;
//a+=b;//a=a+b internally so fine
a=a+b;//Compile time error because 10+10=20 now int
System.out.println(a);
}}
```

Compile time error

After type cast:

```
1.class OperatorExample{
2.public static void main(String args[]){
3.short a=10;
4.short b=10;
5.a=(short)(a+b);//20 which is int now converted to short
6.System.out.println(a);
7.}}
```

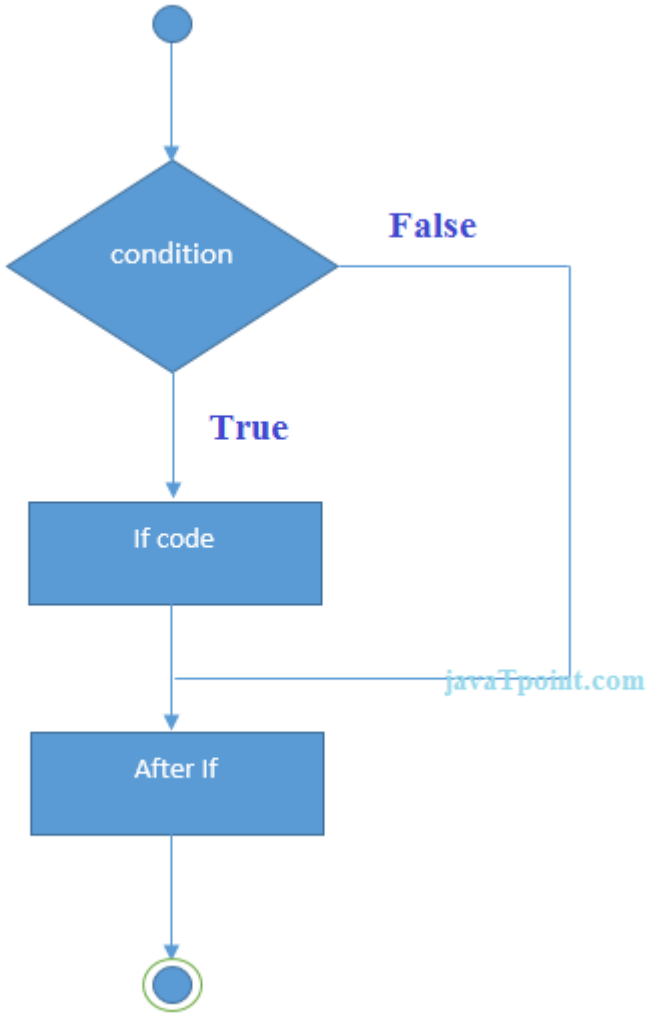
Output:

20

Java If-else Statement

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in java.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement




```
//Java Program to demonstate the use of if statement.
public class IfExample {
public static void main(String[] args) {
   //defining an 'age' variable
   int age=20;
   //checking the age
   if(age>18){
      System.out.print("Age is greater than 18");
   }
}
}
```

Java If-else Statement

Java if-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

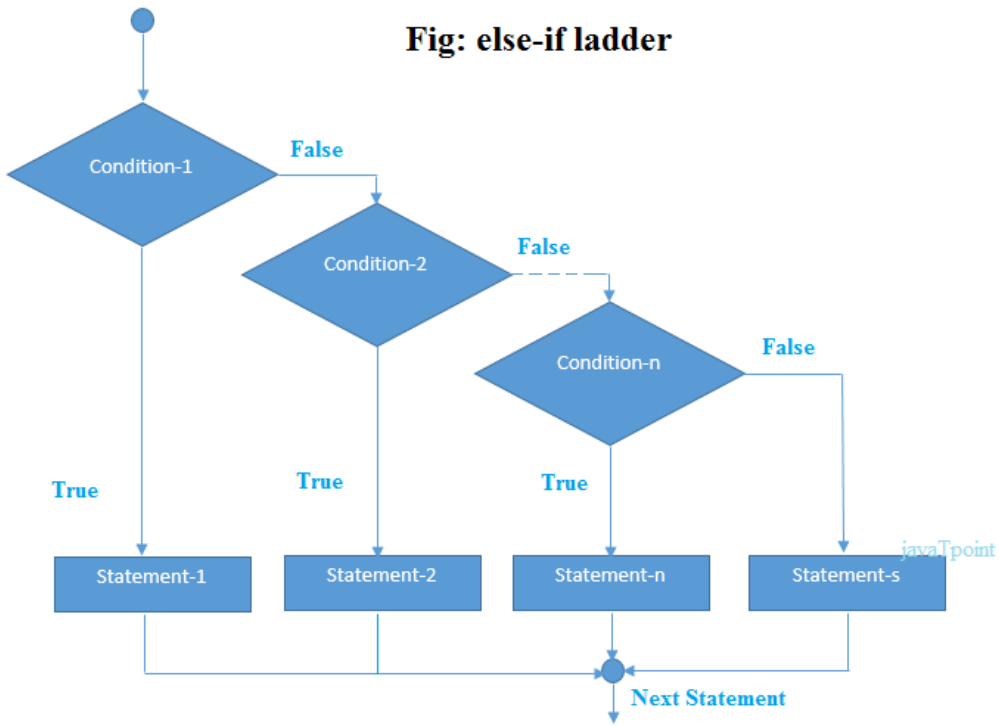


Fig: else-if ladder

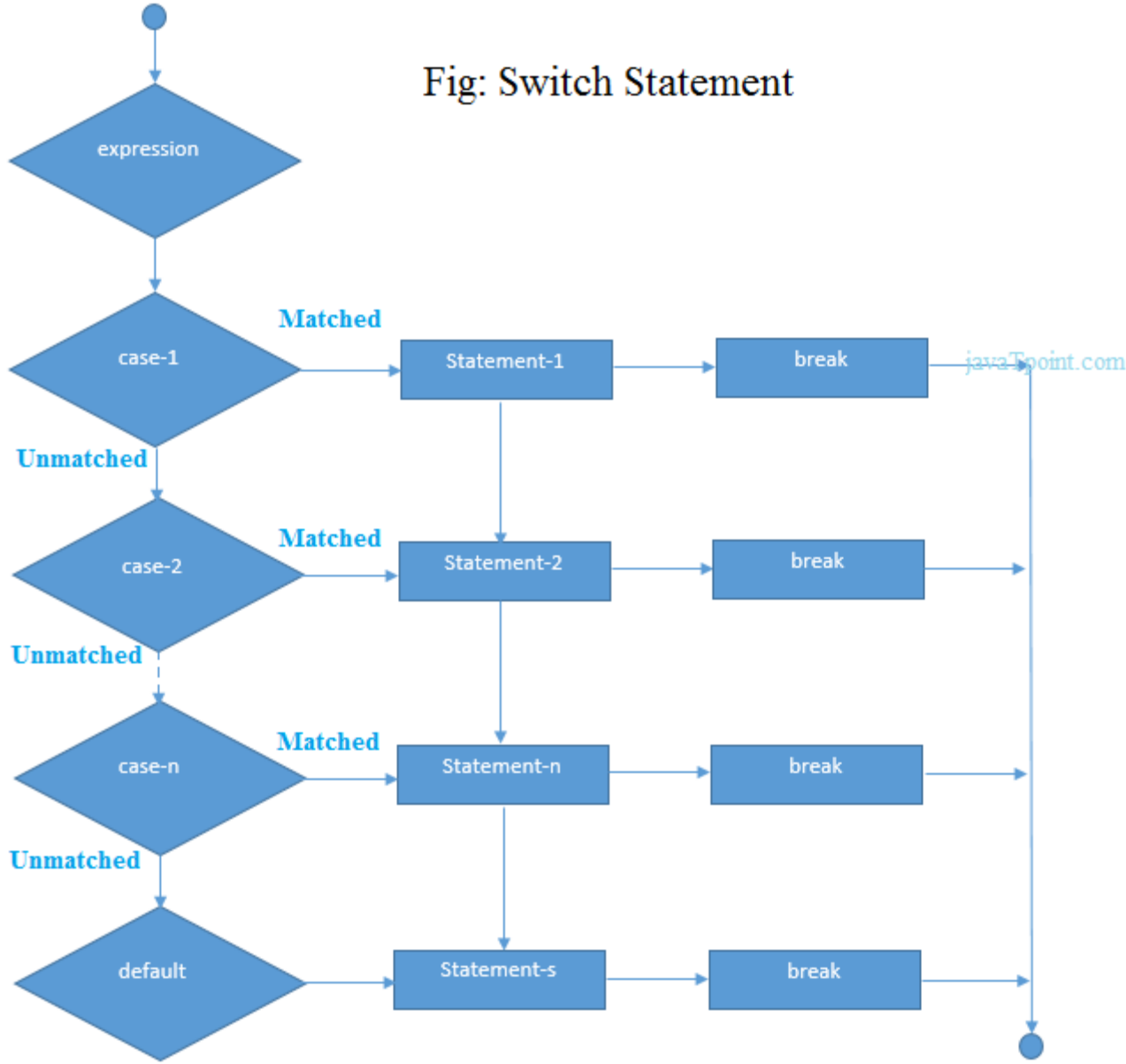
Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.
In other words, the switch statement tests the equality of a variable against multiple values.

Points to Remember

•There can be *one or N number of case values* for a switch expression.
•The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
•The case values must be *unique*. In case of duplicate value, it renders compile-time error.
•The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.
•Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
•The case value can have a *default label* which is optional.

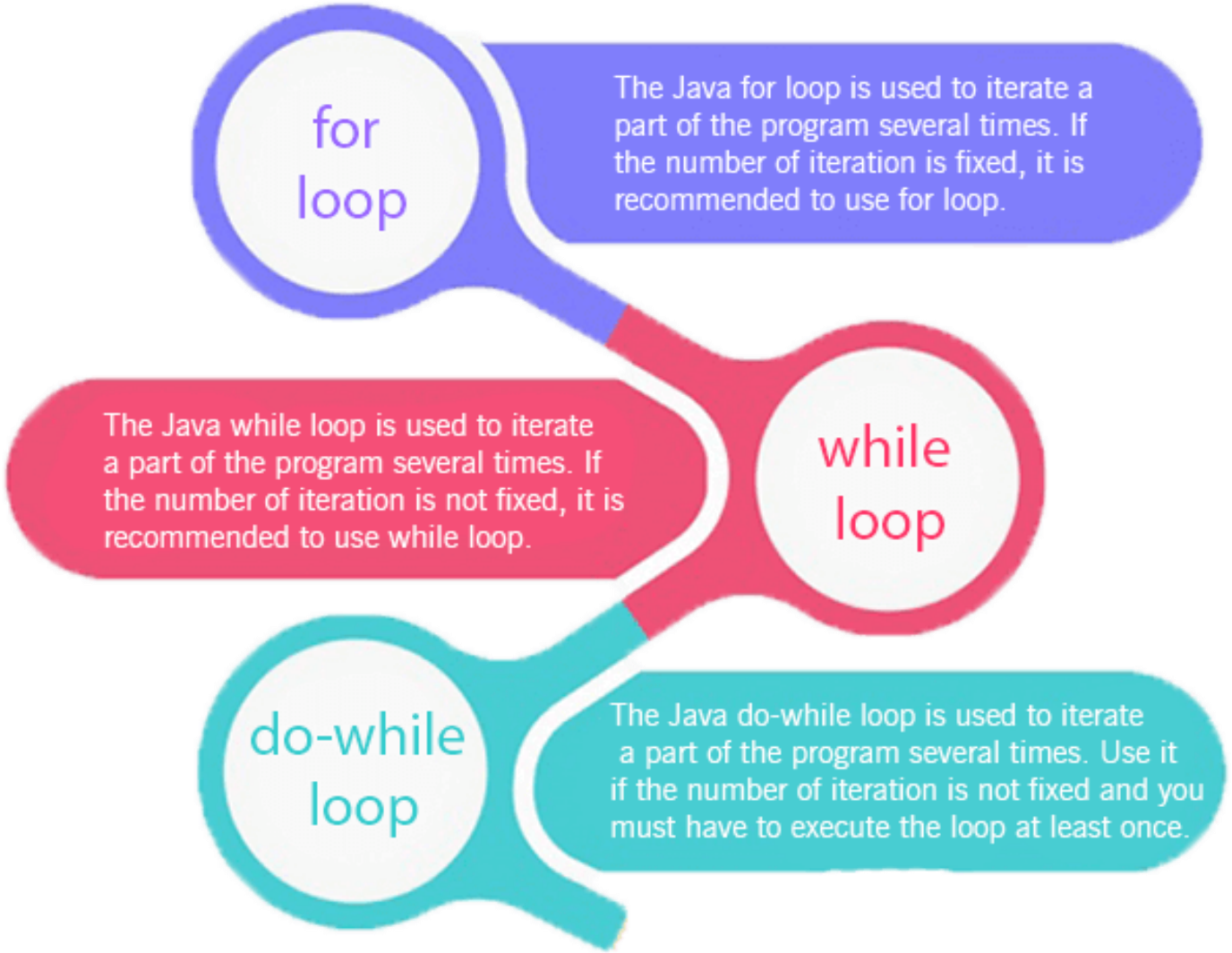
Java Switch Statement



Loops in Java

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in java.

- for loop
- while loop
- do-while loop



Loops in Java

Java For Loop vs While Loop vs Do While Loop

Comparison	for loop	while loop	do while loop
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);`

Loops in Java

Java For Loop vs While Loop vs Do While Loop

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);`

Loops in Java

- Simple For Loop
- For-each or Enhanced For Loop
- Labeled For Loop

Java Simple For Loop

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1.Initialization: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.

2.Condition: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.

3.Statement: The statement of the loop is executed each time until the second condition is false.

4.Increment/Decrement: It increments or decrements the variable value. It is an optional condition.

Loops in Java

Java for-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

Syntax:

```
for(Type var:array){
//code to be executed
}
```

```
//Java For-each loop example which prints the
  //elements of the array
  public class ForEachExample {
  public static void main(String[] args) {
     //Declaring an array
     int arr[]={12,23,44,56,78};
     //Printing array using for-each loop
     for(int i:arr){
        System.out.println(i);
     }
  }
  }
```

Loops in Java

Java Labeled For Loop

We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.

Usually, break and continue keywords breaks/continues the innermost for loop only.

Syntax:

1.labelname:
2.for(initialization;condition;incr/decr){
3.//code to be executed
4.}

```
//A Java program to demonstrate the use of labeled for loop
public class LabeledForExample {
public static void main(String[] args) {
   //Using Label for outer and for loop
   aa:
      for(int i=1;i<=3;i++){
         bb:
            for(int j=1;j<=3;j++){
               if(i==2&&j==2){
                  break aa;
               }
               System.out.println(i+" "+j);
            }
      }
}
}
```

Output:

```
1 1
1 2
1 3
2 1
```

Loops in Java

If you use break bb;, it will break inner loop only which is the default behavior of any loop.

```
public class LabeledForExample2 {
public static void main(String[] args) {
   aa:
      for(int i=1;i<=3;i++){
         bb:
            for(int j=1;j<=3;j++){
               if(i==2&&j==2){
                  break bb;
               }
               System.out.println(i+" "+j);
            }
      }
}
}
```

Java Continue Statement

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.
The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.
We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Java Continue Statement Example
Example:

```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
   //for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //using continue statement
         continue;//it will skip the rest statement
      }
      System.out.println(i);
   }
}
}
```

Java Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.
We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Java Break Statement with Labeled For Loop

We can use break statement with a label. This feature is introduced since JDK 1.5. So, we can break any loop in Java now whether it is outer loop or inner.

Example:

```
//Java Program to illustrate the use of continue statement
//with label inside an inner loop to break outer loop
public class BreakExample3 {
public static void main(String[] args) {
        aa:
        for(int i=1;i<=3;i++){
            bb:
            for(int j=1;j<=3;j++){
               if(i==2&&j==2){
                  //using break statement with label
                  break aa;
               }
               System.out.println(i+" "+j);
            }
        }
}
}
```