Tutorial 2

SMITA SANKHE
Assistant Professor
Department of Computer Engineering

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya TRUST

Outline

- Read input from user:
  - Command line interpreter
  - Buffered reader Class
  - Scanner Class
- Static variable, method, class
- Operators in Java
- Control Statements

To read input Data :
1. command line interpreter

```
Class Sample
{
public static void main(String args[])
{
For(int i=0;i<args.length();i++)
System.out.println(args[i]);
}
}
```

To read input Data :
2. Using BufferedReader and InputStreamReader

```
import java.io.*;
class Sample{
public static void main(String args[])throws Exception
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter your name");
String name=br.readLine();
System.out.println("Welcome "+name);
System.out.println("Enter your number");
int n=Integer.parseInt(br.readLine());
System.out.println(“Number is "+n);
 }
}
```

Using BufferedReader and InputStreamReader (Contd)…

InputStreamReader class

InputStreamReader class can be used to read data from keyboard. It performs two tasks:

- connects to input stream of keyboard
- converts the byte-oriented stream into character-oriented stream

BufferedReader class

BufferedReader class can be used to read data line by line by readLine() method.

System.in

When you type a value in a program, to retrieve it, you can store in the **in** object of the **System class .**

To read input Data
3. Using Scanner class

```
import java.util.Scanner;
class Sample{
 public static void main(String args[]){
   Scanner sc=new Scanner(System.in);
   System.out.println("Enter your id");
   int id=sc.nextInt();
   System.out.println("Enter your name");
   String name=sc.nextLine();
   System.out.println("Enter your salary");
   double salary=sc.nextDouble();
   System.out.println(“Id:"+id+" name:"+name+" Salary:"+salary);
   sc.close();
 }
```

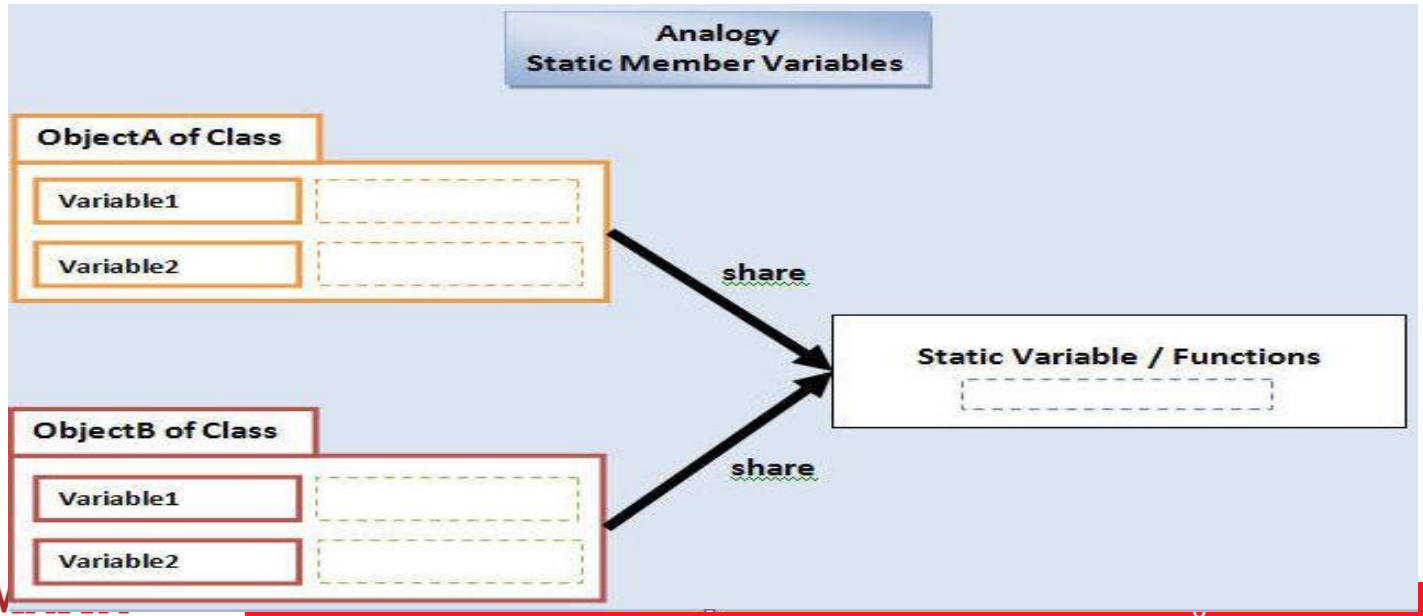
Using Scanner class(contd..)

- The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values**.**

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

Static variable

- When a variable is declared with the keyword “static”, its called a “**class variable**”.
- All instances share the same copy of the variable. A class variable can be accessed directly with the class, without the need to create a instance. It makes your program **memory efficient.**



Example

```
class Counter{
int count=0;//will get memory when instance is
created

Counter(){
count++;
System.out.println(count);
}
public static void main(String args[]){

Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
 }
}  Output:1 1 1
```

Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

```
class Counter2{
static int count=0;//will get memory only once and
retain its value

Counter2(){
count++;
System.out.println(count);
}
public static void main(String args[]){

Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();
 }
}  Output:1 2 3
```

static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value. So its value is incremented

Static method

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.
- There are two main restrictions for the static method. They are:
  - The static method can not use non static data member or call non-static method directly.
  - this and super cannot be used in static context

Example

```
class Difference {
  public static void main(String[] args)
 {
display(); //calling without object
 Difference t = new Difference();
t.show(); //calling using object
}
 static void display() { System.out.println("Programming is amazing."); }
void show(){
System.out.println("Java is awesome.");
}
 }
```

If you wish to call static method of another class then you have to write class name while calling static method

Static block

- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.
- So this is one of the way to **execute a program without main() method**

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}  OUTPUT: static block is invoked
                Hello main
```

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
  - Assignment Operator

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|

Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Control STATEMENTS

- **Selection Statements**
  - Using `if` and `if...else`
  - Nested `if` Statements
  - Using `switch` Statements
  - Conditional Operator

- **Repetition Statements**
  - Looping: `while`, `do-while`, and `for`
  - Nested loops
  - Using `break` and `continue`

Selection Statements

- `if` Statements
- `switch` Statements

`if` Statements

```
if (booleanExpression)
{
   statement(s);
}
```

Example:

```
if ((i > 0) && (i < 10))
  {
   System.out.println("i is an " +
     "integer between 0 and 10");
  }
```

Note:

Adding a semicolon at the end of an if clause is a common mistake.

```
if (radius >= 0);
{
  area = radius*radius*PI;
  System.out.println(
    "The area for the circle of radius " +
    radius + " is " + area);
}
```

Wrong

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

The `if...else` Statement

```
if (booleanExpression)
{
   statement(s)-for-the-true-case;
}
else
{
   statement(s)-for-the-false-case;
}
```

if...else Example

```
if (radius >= 0)
{
   area = radius*radius*PI;

  System.out.println("The area for the "
     + "circle of radius " + radius +
     " is " + area);
}
else
{
   System.out.println("Negative input");
}
```

Multiple Alternative if Statements

```
if (score >= 90)
  grade = 'A';
else
  if (score >= 80)
    grade = 'B';
  else
    if (score >= 70)
      grade = 'C';
    else
      if (score >=
  60)
        grade = 'D';
      else
        grade = 'F';
```

```
if (score >= 90)
  grade = 'A';
else if (score >= 80)
  grade = 'B';
else if (score >= 70)
  grade = 'C';
else if (score >= 60)
  grade = 'D';
else
  grade = 'F';
```

Note: (cont.)

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces:

```
int i = 1;
int j = 2;
int k = 3;
  if (i > j)
  {
    if (i > k)
      System.out.println("A");
  }
  else
    System.out.println("B");
```

Output: B.

Switch Statement

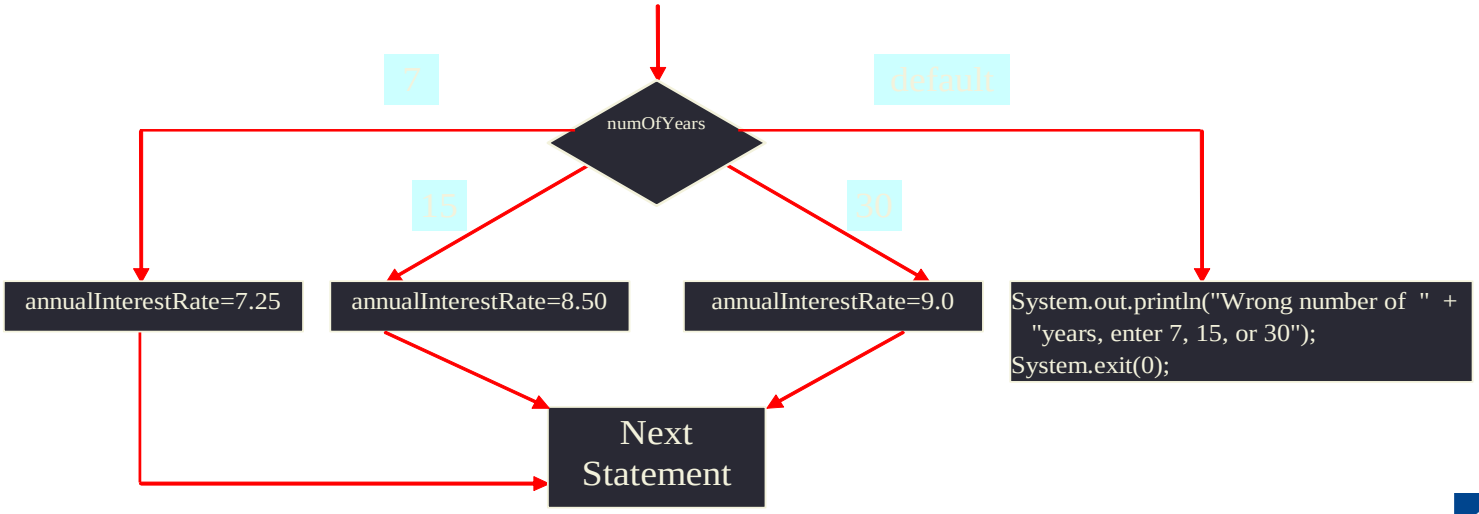
- The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.
- The switch statement works with byte, short, int, long, enum types, String

Example:

This program reads in number of years and loan amount and computes the monthly payment and total payment. The interest rate is determined by number of years.

switch Statements

```
switch (year) {
  case 7:  annualInterestRate = 7.25;
           break;
  case 15: annualInterestRate = 8.50;
           break;
  case 30: annualInterestRate = 9.0;
           break;
  default: System.out.println(
    "Wrong number of years, enter 7, 15, or 30");
}
```



switch Statement Rules

- The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.
- The value1, ..., and valueN must have the same data type as the value of the switch-expression.
- The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. (The case statements are executed in sequential order.)
- The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement.
  If the break statement is not present, the next case statement will be executed.

`switch` Statement Rules, cont.

- The default case, which is optional, can be used to perform actions when none of the specified cases is true.

- The order of the cases (including the default case) does not matter. However, it is a good programming style to follow the logical sequence of the cases and place the default case at the end.

Note:

Do not forget to use a break statement when one is needed. For example, the following code always displays Wrong number of years regardless of what numOfYears is.

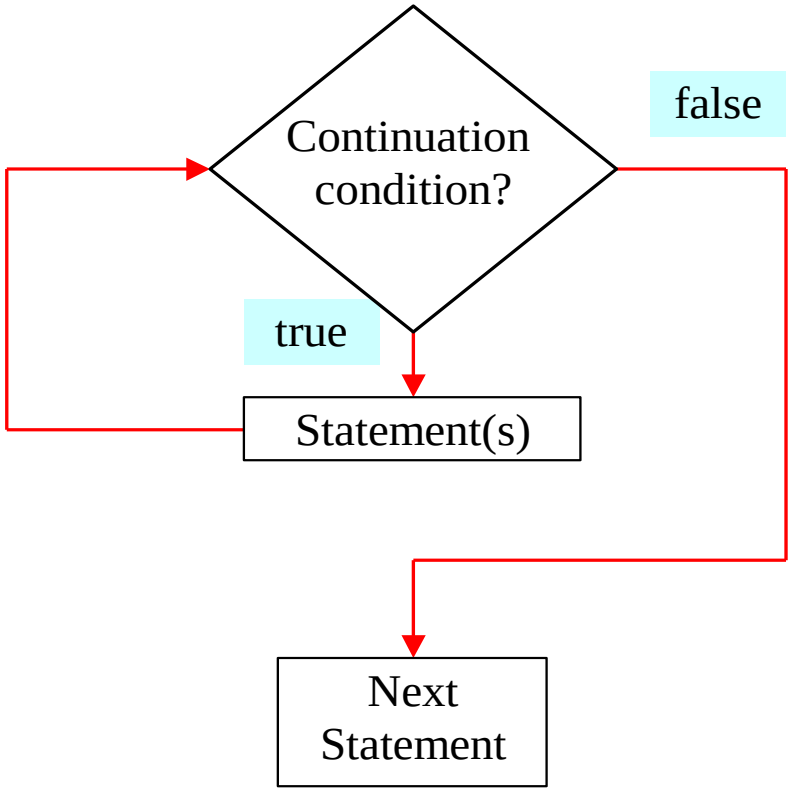
```
switch (numOfYears) {
  case 7:  annualInterestRate = 7.25;
  case 15: annualInterestRate = 8.50;
  case 30: annualInterestRate = 9.0;
  default: System.out.println("Wrong number of
years");
}
```

Repetitions

- `while` Loops
- `do-while` Loops
- `for` Loops
- `for` each loops
- `break` and `continue`

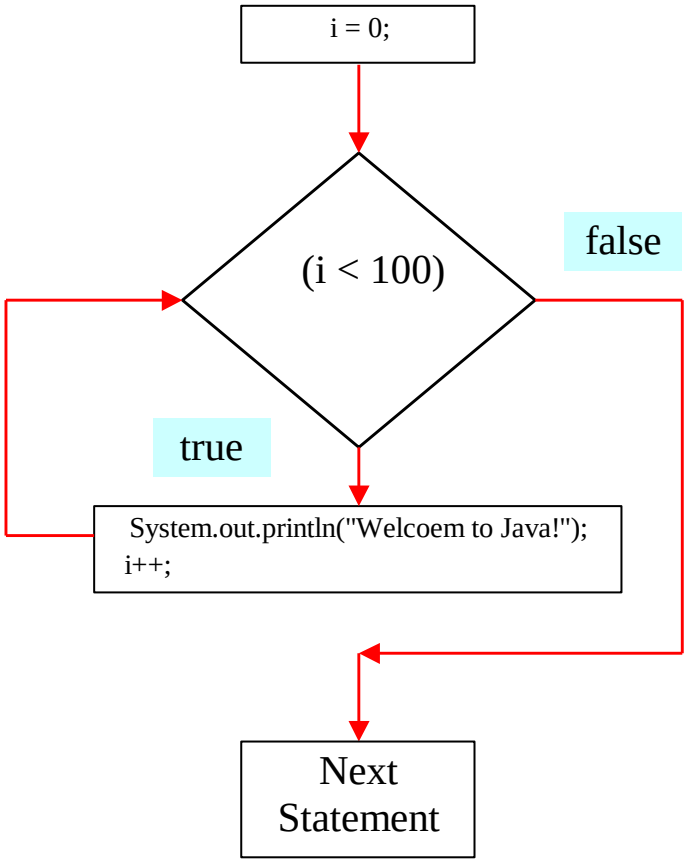
`while` Loop Flow Chart

```
while (continuation-condition)
{
  // loop-body;
}
```



Example:

```
int i = 0;
while (i < 100) {
   System.out.println(
     "Welcome to Java!");
   i++;
}
```



Note:

Don’t use floating-point values for equality checking in a loop control.

Since floating-point values are approximations, using them could result in imprecise counter values and inaccurate results.

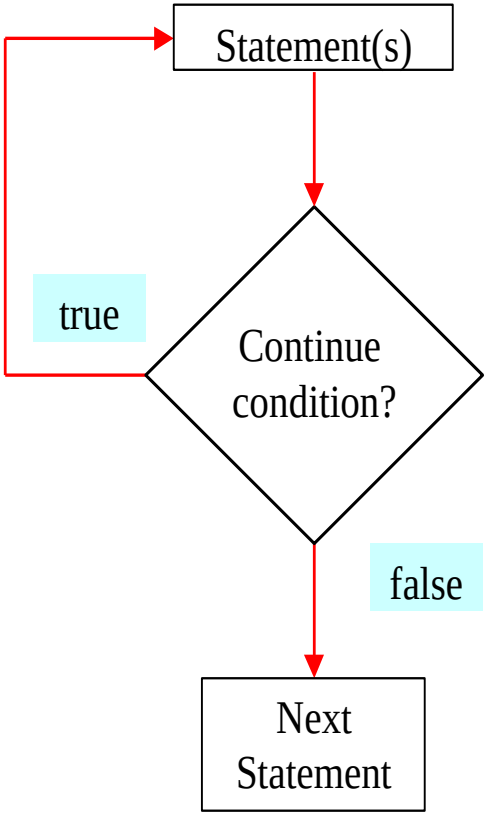
This example uses int value for data. If a floating-point type value is used for data, (data != 0) may be true even though data is 0.

```
// data should be zero
double data = Math.pow(Math.sqrt(2), 2) - 2;

if (data == 0)
  System.out.println("data is zero");
else
  System.out.println("data is not zero");
```

do-while Loop

```
do {
   // Loop body;
} while (continue-condition);
```



for Loops

```
for (initial-action; loop-continuation-
  condition; action-after-each-iteration) {
   //loop body;
}
```

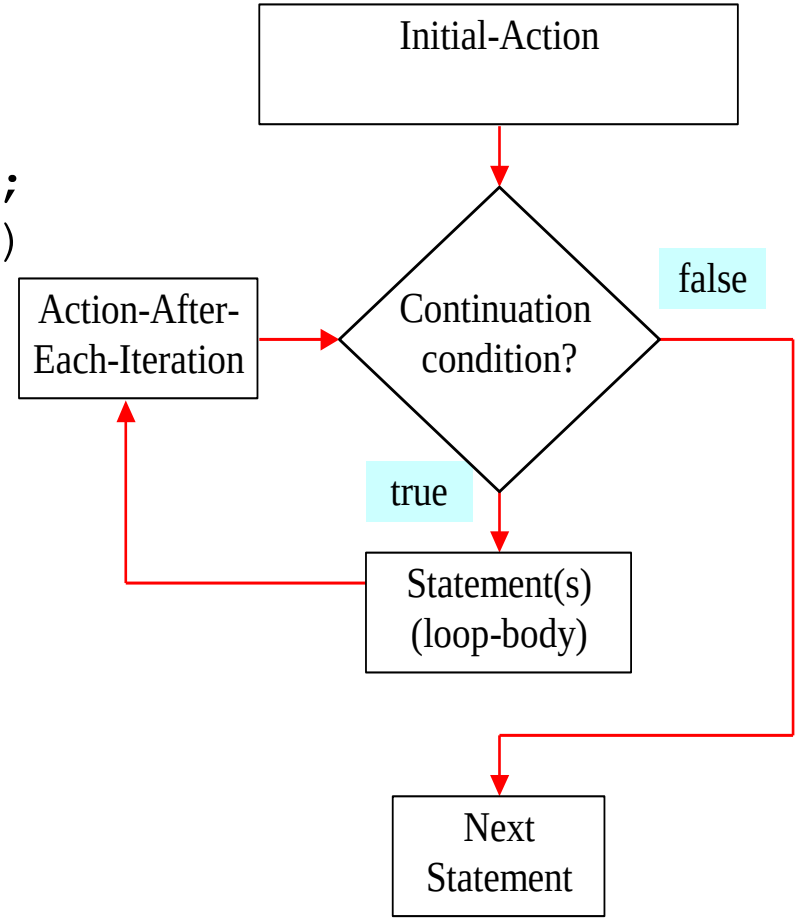
```
int i = 0;
while (i < 100)
{
  System.out.println("Welcome to Java! ” + i);
  i++;
}
```

Example:

```
int i;
for (i = 0; i < 100; i++)
 {
  System.out.println("Welcome to Java! ” + i);
}
```

for Loop Flow Chart

```
for (initial-action;
   loop-continuation-condition;
   action-after-each-iteration)
{
    //loop body;
}
```



Note

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

Wrong

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

Note: (cont…)

Similarly, the following loop is also wrong:

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

Wrong

In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
   System.out.println("i is " + i);
   i++;
} while (i<10);
```

Correct

for each loop

- In Java, there is another form of for loop (in addition to standard for loop) to work with arrays and collection, the enhanced for loop.

- Syntax of for-each loop

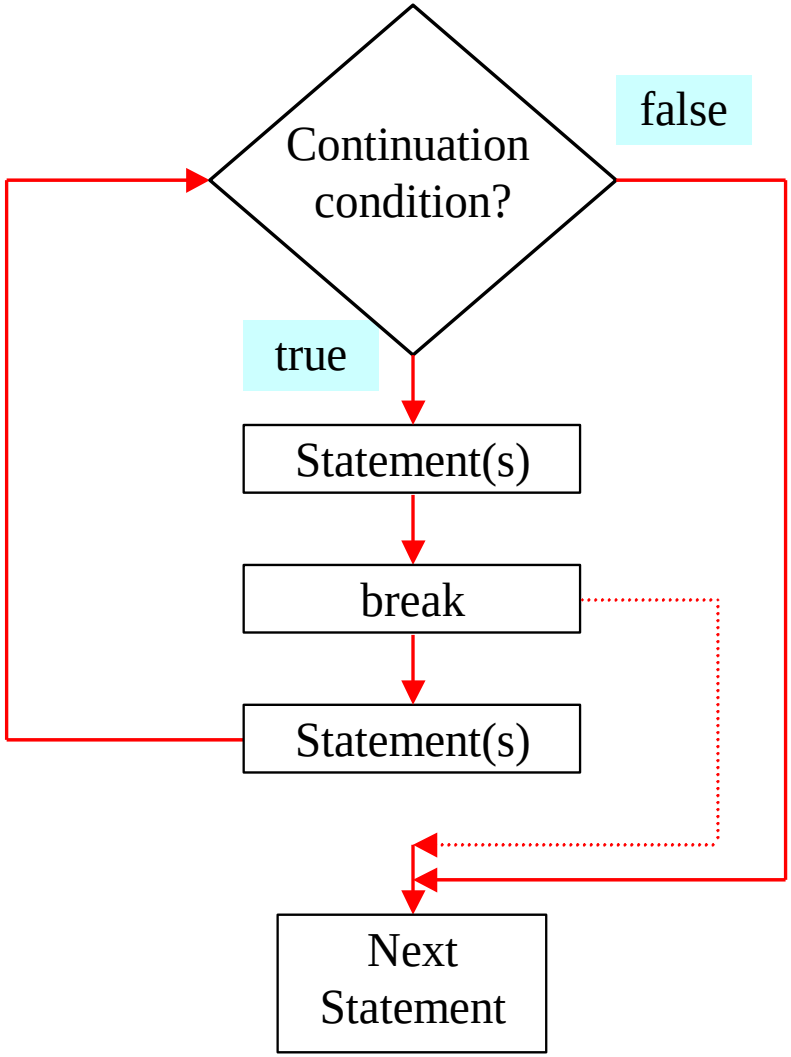
```
for(data_type item : collection)
 { ... }
```

Example:

```
class ForLoop
{
public static void main(String[] args)
 {
char[] vowels = {'a', 'e', 'i', 'o', 'u'};
for (int i = 0; i < vowels.length; ++ i)
{
System.out.println(vowels[i]);
}
}
}
```

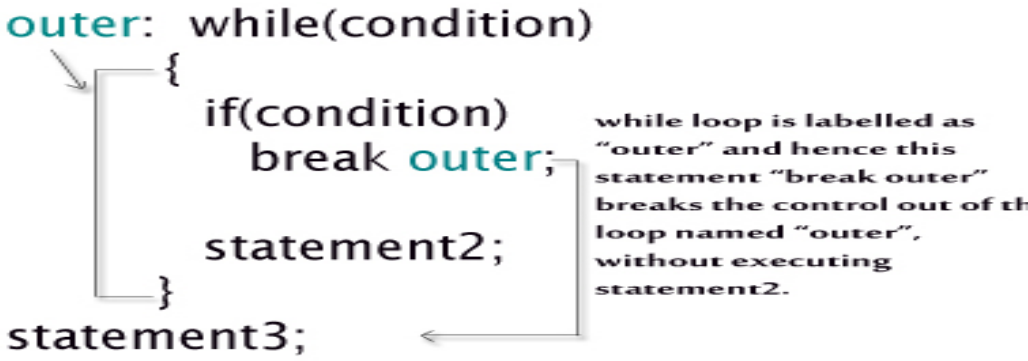
```
class foreachLoop
 {
public static void main(String[] args) {
char[] vowels = {'a', 'e', 'i', 'o', 'u'};
// foreach loop
for (char item: vowels) {
System.out.println(item);
}
}
}
```

The `break` Keyword



Break with label

- In **Labelled Break Statement**, we give a *label/name* to a loop.
- When this **break statement** is encountered with the *label/name of the loop*, it *skips* the execution any statement after it and takes the control right out of this labelled loop.
- And, the control goes to the first statement right after the loop.



labelled break

Example: 1

```
public class LabelledBreak
{
public static void main(String... ar)
{
int i=7;
loop1:
while(i<20)
{
        if(i==10)
                break loop1;
System.out.println("i ="+i);
        i++;
}System.out.println("Out of the loop");

} //main method ends

} //class ends
```

OUTPUT:

i =7

i =8

i =9

Out of the loop

Example: 2

```
public class LabelledBreak
{
public static void main(String... ar)
{
loop2:
for(int i=0;i<2;i++)
for(int j=0;j<5;j++)
{
        if(j==2)
                break loop2;
        System.out.println("i = "+i);
        System.out.println("j = "+j);
}System.out.println("Out of the loop");
} //main method ends
} //class ends
```

OUTPUT:

i = 0
j = 0
i = 0
j = 1
Out of the loop

The `continue` Keyword