

PROGRAMMING

Lecture 3

Sushil Paudel

How people reacts differently to a single word.

"Bug"



Tester



Developer



Manager

LAST WEEK

- Separators
- Comments
- Identifiers
- Keywords
- Java Class Library
- Syntax
- Indentation
- Variable
- Data types
- Operators

TODAY'S TOPIC

- Control Statements – If Else
- Control Statement – Switch
- Control Statement Loop – For, While, Do-While
- Break and Continue

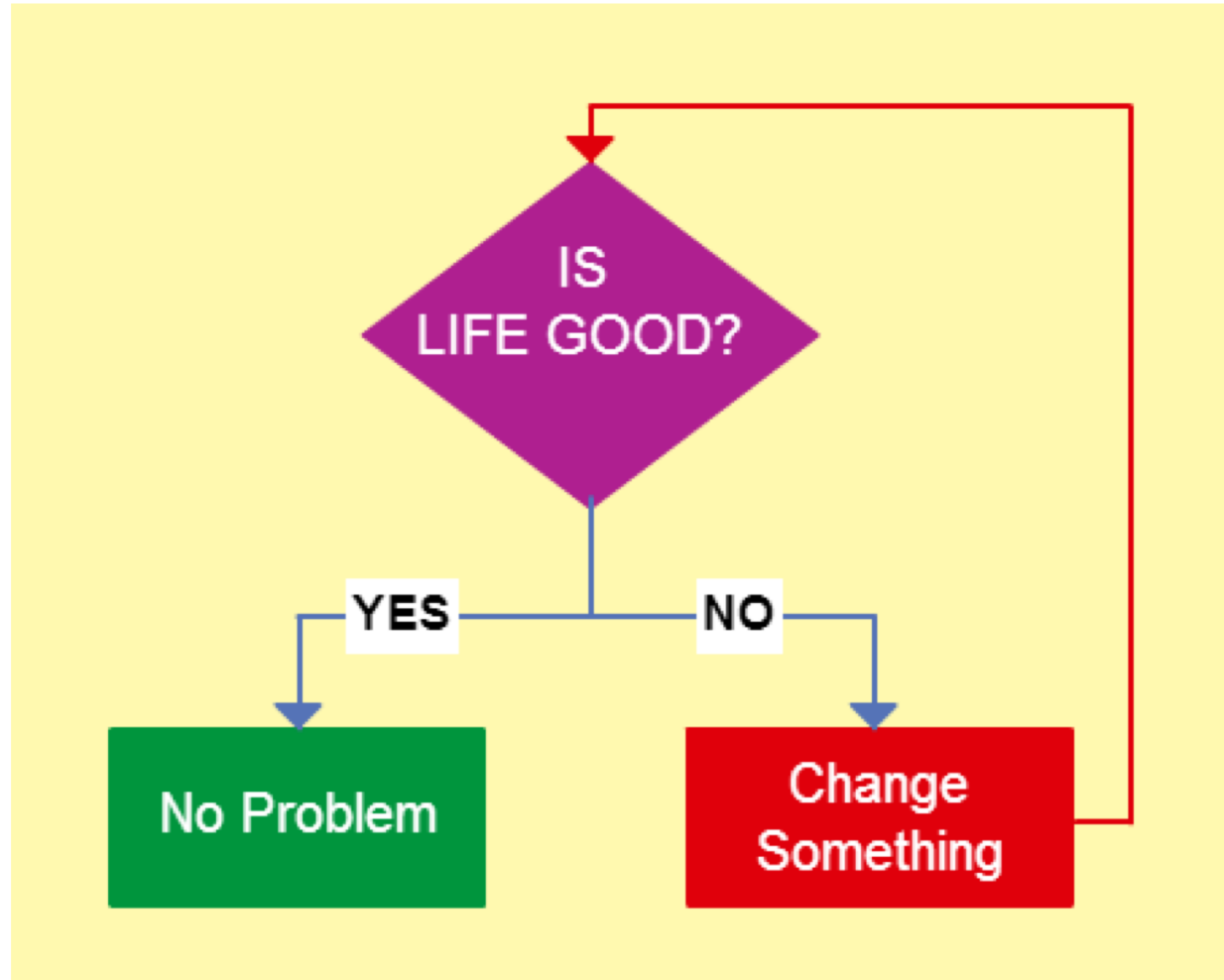
CONTROL STATEMENT



CONTROL STATEMENT – DECISION STATEMENTS

- Decision making statements (selection statements) in Java
 - if statement
 - if-else statement
 - if-else-if statement
 - switch statement

DECISION MAKING



IF STATEMENT

- The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax


```
if (expression) {  
    // statements  
}
```

- Here expression is a **boolean** expression (returns either true or false).
- If the expression is evaluated to true, statement(s) inside the body are executed and if it is false, the statements are skipped from execution.

IF STATEMENT


Expression is true.

```
int test = 5;  
  
if (test < 10)  
{  
    // codes  
}  
  
// codes after if
```

A flow diagram showing a horizontal line from the left entering the 'if' statement, then a vertical line going down and a horizontal line going right to enter the code block.

Expression is false.

```
int test = 5;  
  
if (test > 10)  
{  
    // codes  
}  
  
// codes after if
```

A flow diagram showing a horizontal line from the left entering the 'if' statement, then a vertical line going down and a horizontal line going right to skip the code block and enter the code after the 'if' statement.

EXAMPLE

```
class IfExample {  
    public void checkGreater() {  
        int i = 10;  
  
        if (i > 15) {  
            System.out.println("10 is less than 15");  
        }  
  
        // This statement will be executed  
        System.out.println("I am always executed");  
    }  
}
```

OUTPUT

I am always executed

EXAMPLE USING AND

```
class IfExample {  
    public void isValidMarks() {  
  
        int i = 10;  
  
        if (i > 0 && i < 100) {  
            System.out.println("The marks is valid");  
        }  
    }  
}
```

OUTPUT

The marks is valid

EXAMPLE USING OR

```
class IfExample {  
    public void isNumberValid() {  
  
        int i = 10;  
  
        if (i == 10 || i == 50) {  
            System.out.println("The number is either 10 or 50");  
        }  
    }  
}
```

OUTPUT

The number is either 10 or 50

IF – ELSE STATEMENT

- The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't.
- But what if we want to do something else if the condition is false.
- Here comes the else statement.
- We can use the else statement with if statement to execute a block of code when the condition is false.

IF ELSE SYNTAX

```
if (condition) {  
    // Executes this block if condition is true  
} else {  
    // Executes this block if condition is false  
}
```

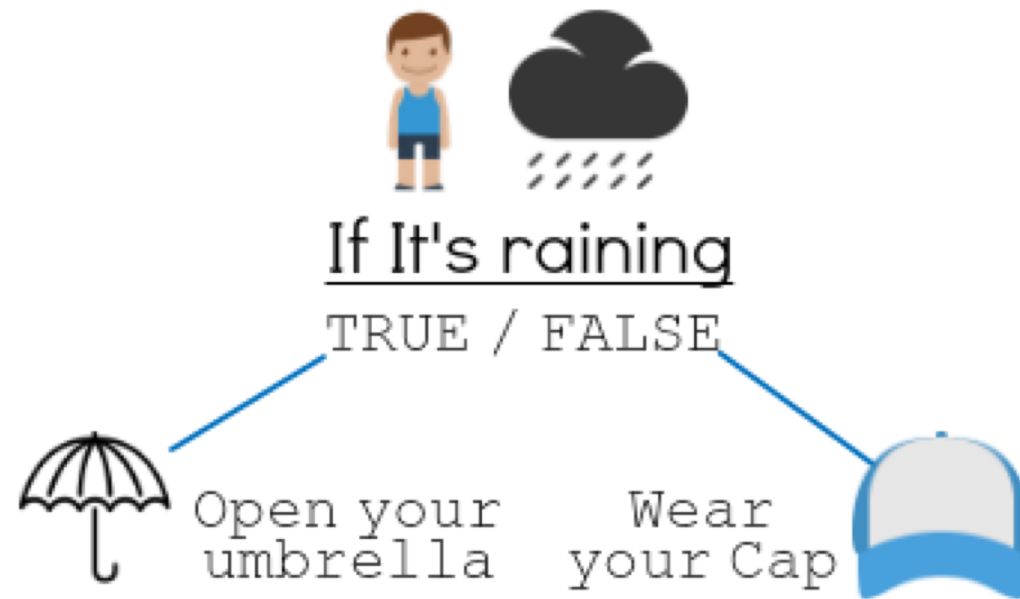
IF ELSE EXAMPLE

```
class IfElseTest {  
    public void checkSmaller() {  
        int i = 30;  
        if (i < 15) {  
            System.out.println("i is smaller than 15");  
        } else {  
            System.out.println("i is greater than 15");  
        }  
    }  
}
```

OUTPUT

i is greater than 15

EXAMPLE



IF-ELSE-IF LADDER

- A user can decide among multiple options.
- The if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed.

IF-ELSE-IF SYNTAX

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false and condition2 is true  
} else if (condition3) {  
    // block of code to be executed if the condition1, condition 2 are false and  
    condition3 is true  
} else {  
    // block of code to be executed if all the conditions are false  
}
```

IF-ELSE-IF EXAMPLE

```
class Ladder {  
    public static void main(String[] args) {  
        int number = 0;  
        if (number > 0) {  
            System.out.println("Number is positive.");  
        } else if (number < 0) {  
            System.out.println("Number is negative.");  
        } else {  
            System.out.println("Number is 0.");  
        }  
    }  
}
```

OUTPUT

Number is 0

SWITCH STATEMENT

- The switch statement in java language is used to execute one statement from multiple conditions.
- It is like if-else-if statement.
- The syntax of switch statement given below

SWITCH STATEMENT

```
switch(expression) {  
    case x:  
        // code block  
        break;  
  
    case y:  
        // code block break;  
        break;  
  
    default:  
        // code block  
  
}
```

SWITCH STATEMENT

- When Java reaches a **break** keyword, it breaks out of the switch block.
- The break statements are important because if they are not used, all statements after the matching case label are executed in sequence until the end of switch statement.
- The **default** keyword specifies some code to run if there is no case match:
- Note that if the default statement is used as the last statement in a switch block, it does not need a break.
- Switch statement has more readability and efficient for large comparison.

SWITCH EXAMPLE

```
class SwitchTest {  
    public static void main(String[] args) {  
        int i = 9;  
        switch (i) {  
            case 0: // if i == 0  
                System.out.println("i is zero.");  
                break;  
            case 1: // Else if i == 1  
                System.out.println("i is one.");  
                break;  
            default: // Else  
                System.out.println("i is not 0 and 1");  
        }  
    }  
}
```

OUTPUT

i is not 0 and 1

NESTED IF

- Placing If Statement inside another IF Statement is called as Nested If in Java Programming.
- Here, the inner if block condition executes only when outer if block condition is true.

NESTED IF EXAMPLE

```
public class Test {  
    public static void main(String args[]) {  
        int x = 10;  
        if( x < 100 ) {  
            if( x%2 == 0 ) {  
                System.out.print("X is even");  
            } else{  
                System.out.print("X is odd");  
            }  
        }  
    }  
}
```

OUTPUT

X is even

BREAK

2 minutes

INTERMISSION

CONTROL STATEMENT - LOOPS

- In programming languages, loops are used to execute a set of instructions/functions repeatedly.
- Loop is used in programming to repeat a specific block of code until certain condition is met (test expression is false).
- Loops are what makes computers interesting machines. Imagine you need to print a sentence 50 times on your screen.
- Well, you can do it by using print statement 50 times (without using loops).
- How about you need to print a sentence one million times?

TYPES OF LOOPS

There are three types of loops in java.

- for loop
- while loop
- do-while loop

FOR LOOP

- The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
- There are two types of for loops in java.
 - Simple For Loop
 - For-each or Enhanced For Loop

SIMPLE FOR LOOP

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

- **Initialization:** It is the initial condition which is executed once when the loop starts.
- **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.
- **Statement:** Codes which are executed till the conditions are true
- **Increment/Decrement:** It increments or decrements the variable value. It is an optional condition.

EXAMPLE

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

OUTPUT

1

2

3

4

5

6

7

8

9

10

```
for (int i = 0; i < 3; i++) {  
    System.out.println("i is now " + i);  
}
```


EXAMPLE

```
public class ForExample {  
    public static void main(String[] args) {  
        for (int i = 10; i >= 0; i--) {  
            System.out.println(i);  
        }  
    }  
}
```

OUTPUT

10

9

8

7

6

5

4

3

2

1

0

FOR-EACH LOOP

- There is also a "**for-each**" loop, which is used exclusively to loop through elements in an **array**:

Syntax

```
for (type variableName : arrayName) {  
    // code block to be executed  
}
```

FOR EACH LOOP EXAMPLE

```
public class ForEachExample {  
    public static void main(String[] args) {  
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
        for (String car : cars) {  
            System.out.println(car);  
        }  
    }  
}
```

WHILE LOOP

A **while** loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

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

WHILE LOOP EXAMPLE

```
public class WhileTest {  
    public static void main(String args[]) {  
        int x = 10;  
        while (x < 20) {  
            System.out.println("value of x : " + x);  
            x++;  
        }  
    }  
}
```

OUTPUT

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

value of x : 18

value of x : 19

```
int a = 1;  
while ( a < 4 )  
{  
    System.out.println("Hello World");  
    a ++;  
}
```

Output

DO WHILE LOOP

- The Java *do-while loop* is executed at least once because condition is checked after loop body.
- do while loop in Java is similar to while loop except that the condition is checked after the statements are executed, so do while loop guarantees the loop execution at least once.

DO WHILE SYNTAX

```
do{  
    //code to be executed  
}while(condition);
```

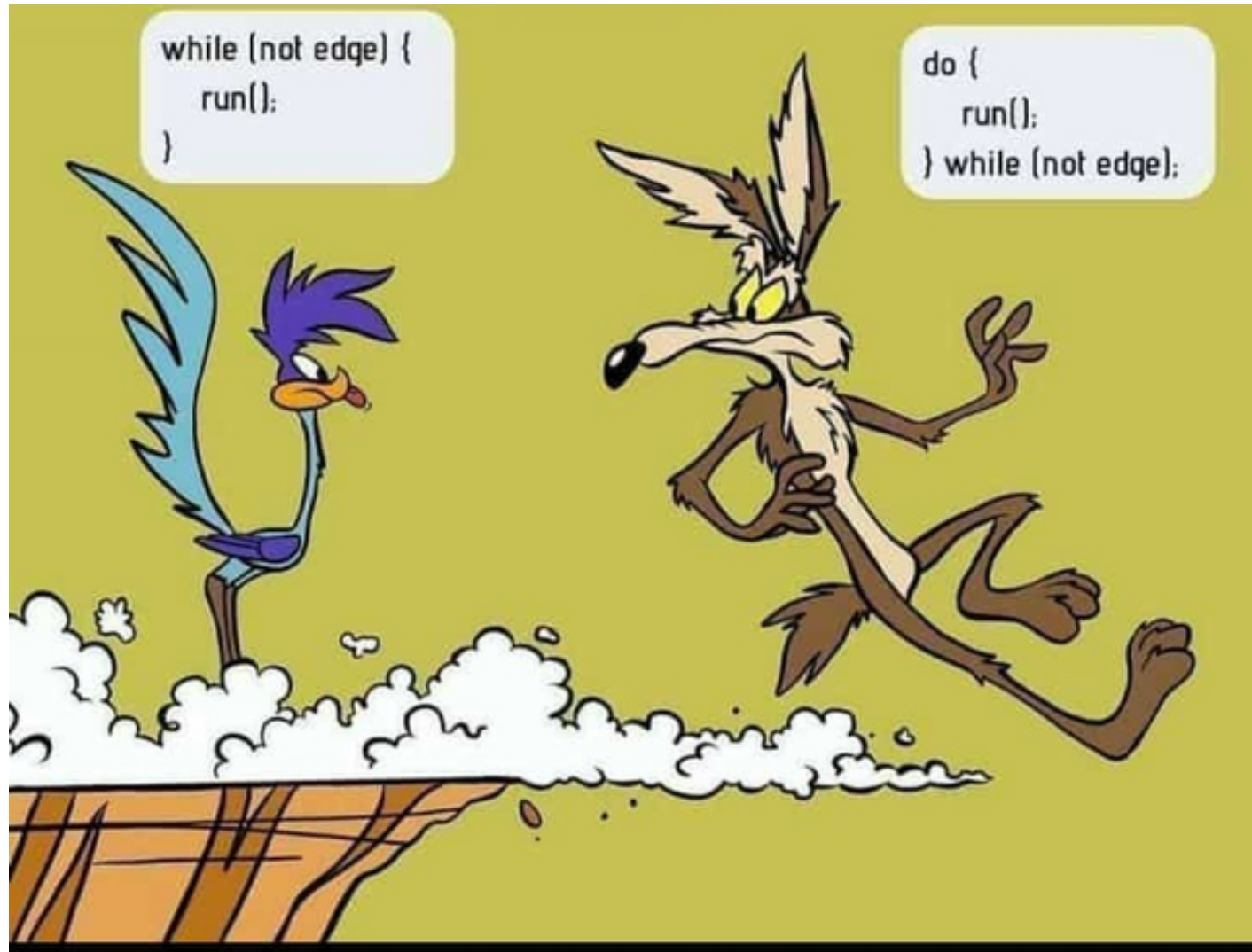
DO WHILE EXAMPLE

```
public class DoWhileExample {  
    public static void main(String[] args) {  
        int i = 1;  
        do {  
            System.out.println(i);  
            i++;  
        } while (i <= 10);  
    }  
}
```

OUTPUT

1
2
3
4
5
6
7
8
9
10

WHILE VS DO WHILE



BREAK IN LOOP

- The break statement is also be used to jump out of a **loop**.
- When the **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

BREAK IN LOOP EXAMPLE

```
public class BreakExample {  
    public static void main(String[] args) {  
        for (int i = 0; i < 10; i++) {  
            if (i == 4) {  
                break;  
            }  
            System.out.println(i);  
        }  
    }  
}
```

OUTPUT

0

1

2

3

CONTINUE

- Continue statement is mostly used inside loops.
- Whenever it is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop's body for the current iteration.
- This is particularly useful when you want to continue the loop but do not want the rest of the statements(after continue statement) in loop body to execute for that particular iteration.

EXAMPLE

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

OUTPUT

1

2

3

4

6

7

8

9

10

HEY, SOMEONE STOP THE LOOP!



END OF LECTURE 3

Any questions?