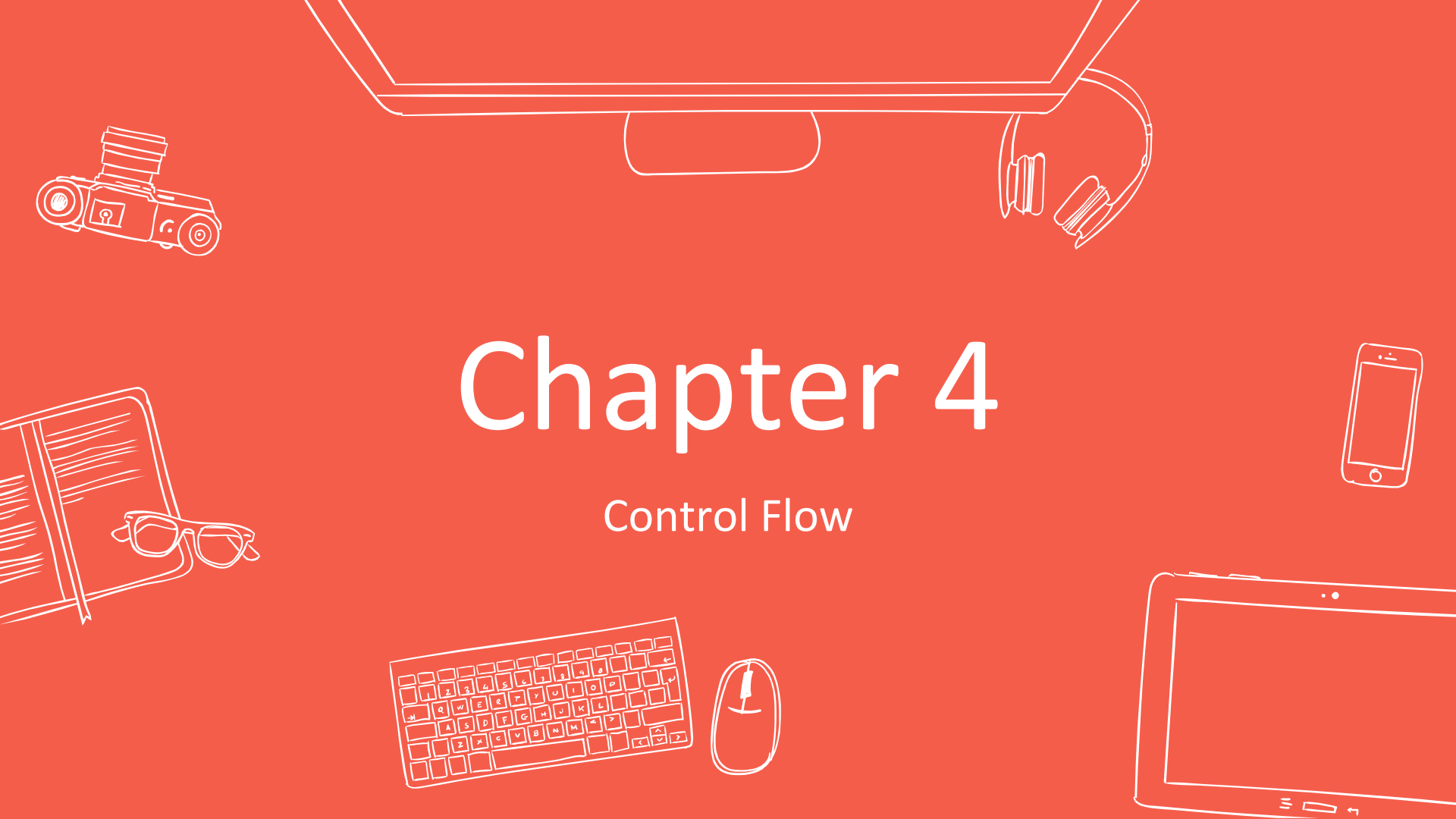


Chapter 4

Control Flow



Expressions

Expressions

Statements

Blocks

Expression

An expression has a value, computed from variables or fields, operators, method calls. An expression can be assigned to a variable, passed as parameter or returned from a method.

E.g.:

`2*3`

`num1 > num2`

`5 + subOfNumber(5,4)`

`result = 1 + 2`

Statement

Statement is made up of one or more expressions

These operations are done in statements, which can also be a conditional, a loop, etc. So, an expression is part of a statement.

E.g.:

```
int number1 = 10;
```

```
System.out.println("Value of number1 is : " + number1);
```

```
int result = 1 + 2; // result is now 3
```

```
calculateInterest ((1000 * 2), tenure, (years * 12));
```

Block

A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed. The following example, BlockDemo, illustrates the use of blocks:

E.g.:

```
class BlockDemo {  
    public static void main(String[] args) {  
        boolean condition = true;  
        if (condition) { // begin block 1  
            System.out.println("Condition is true.");  
        } // end block one  
        else { // begin block 2  
            System.out.println("Condition is false.");  
        } // end block 2  
    }  
}
```

Method

Method declaration

We will use this way of method declaration in our assignment

```
public static <returnType> <methodName> ( zero or more parameters){ }
```

Return type can be void or some data type.

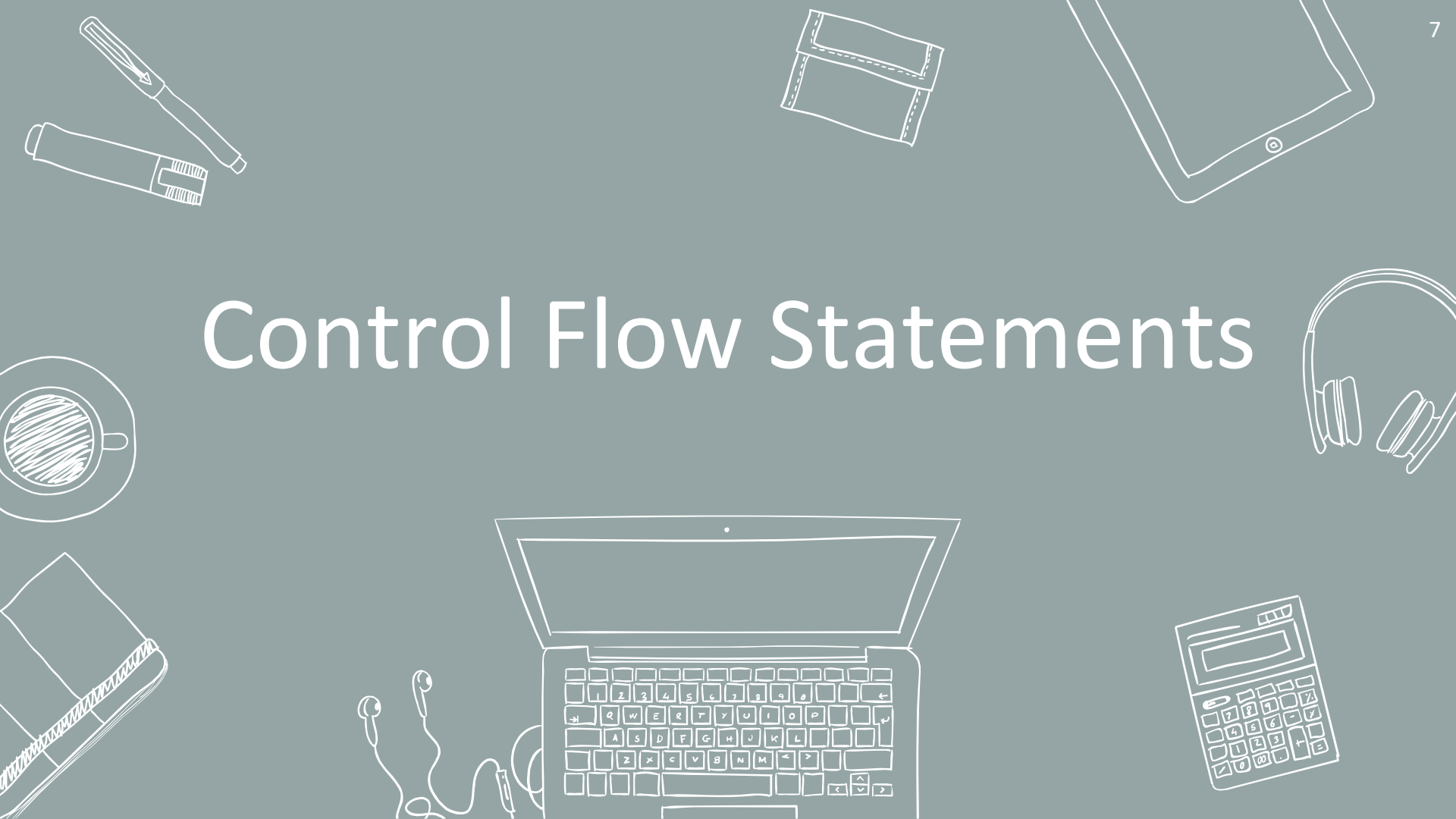
If return type is not void, then method should have return statement to return value.

E.g.

```
public static int add2Numbers (int aNum1, int aNum2) {  
    return (aNum1 + aNum2);  
}
```

```
Public static void printMessage (String aMsg) {  
    System.out.println ("Message " + aMsg);  
}
```

Control Flow Statements



Control Flow Statements

The statements inside your source files are generally executed from top to bottom, in the order that they appear. Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code.

Decision-making statements

The if-then Statement

The if-then-else Statement

The switch Statement

If-then

```
If (expression) {  
    Statements;  
}
```

E.g.

```
int num1 = 9;  
if (num1 > 10) {  
    System.out.println("Num1 " + num1 + " is greater then 10");  
}
```

If-then-else

```
If (expression) {  
    Statements;  
}  
else {  
    statements  
}
```

E.g.

```
int num1 = 9;  
if (num1 % 2 == 0) {  
    System.out.println("Num " + num1 + " is even number");  
}  
else {  
    System.out.println("Num " + num1 + " is add number");  
}
```

If-then-else ladder

```
If (expression) {  
    Statements;  
} else if (expression 2) {  
    statements  
} else {  
    Statements  
}
```

E.g.

```
if (num1 == 1) {  
    System.out.println("Today is Sunday");  
} else if (num2 == 2) {  
    System.out.println("Today is Monday");  
}
```

Switch Statement

Syntax:

```
switch (expression) {  
    case 1:  
        statement;  
        break;  
    case 2:  
    case 3:  
        statement;  
        break;  
    default:  
        statement;  
}
```

```
int day = 1;  
String dayName;  
switch (month) {  
    case 1:  
        dayName = "Sunday";  
        break;  
    case 2:  
        dayName = "Monday";  
        break;  
}
```

Array

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

Define array

```
datatype[] variableName;
```

E.g.

```
String[] cars;
```

Array

Initializing Array

`cars = new String[10]; OR`

`String[] cars = {"Volvo", "BMW", "Ford",
"Mazda"};`

Note: Once the length of the array is defined, it cannot be changed in the program.

Array

Accessing

- `cars[0] = "Nissan"; // Set first element`
- `cars[1] = "Tesla"; // initialize second element`

If the length of an array is n , the first element of the array will be `arrayName[0]` and the last element will be `arrayName[n-1]`.

Array - Try out

```
int[] numbers = {1,2,3,4};  
System.out.println("First element in array " + numbers[0]);  
System.out.println("Number of elements in array " + numbers.length);  
int arrayLength = numbers.length;  
System.out.println("Last element in array " + numbers[arrayLength-1]);  
//Try this:  
System.out.println("Element at " + arrayLength + " index in array " + numbers[arrayLength]);
```

Looping statements



Looping statements

While

do-while

For

While

Syntax:

```
while (expression) {  
    statement(s)  
}
```

E.g.

```
while (userOption == 'E') {  
    System.out.println ("Perform operation " + userOption);  
}
```

do-while

Syntax:

```
do {  
    statement(s)  
} while (expression);
```

Do block is executed at least once.

E.g.

```
int i=10;  
do {  
    System.out.println(i);  
    i--;  
} while(i>1);
```

For Loop

Syntax:

```
for (initialization; termination; increment) {  
    statement(s)  
}
```

E.g.

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

For Each

Syntax:

```
for (collectionElement : collection ) {  
    statement(s)  
}
```

E.g.

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
for (String car : cars) {  
    System.out.println("Car name " + car);  
}
```

Branching statements

Break

Continue

Return

Labeled break

Break statement

Break statement terminates the innermost switch, for, while, or do-while statement

E.g.

```
while (true) {  
    userInput = readUserInput();  
    if(userInput == 'E') {  
        // This will exit while loop and continue with next statement after while block.  
        break;  
    }  
    System.out.println("Continue processing");  
}
```

Continue statement

Continue statement is used inside loops. Whenever it is encountered, 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

E.g.

```
for (int i=0; i < 10; i++) {  
    if (isPrimeNumber(i)) {  
        continue;  
    }  
    printFactors(i);  
}
```

Return statement

A return statement causes the program control to transfer back to the caller of a method. Every method in Java is declared with a return type and it is mandatory for all java methods. Return statement in loops cause to terminate the loop and control get transfer to caller of the method.

```
public static boolean studentPassTheExam (int[] aStudentMarks) {  
    for (int mark : aStudentMarks) {  
        if (mark < 50) {  
            return false;  
        }  
    }  
    return true;  
}  
  
public static void main () {  
    int[] studentMarks = {67, 90, 54, 25};  
    if(studentPassTheExam (studentMarks)) {  
        System.out.println("Pass");  
    }  
}
```

Labeled break statement

labeled break terminates an outer statement. it does not transfer the flow of control to the label. Control flow is transferred to the statement immediately following the labeled (terminated) statement.

Thanks!

Any questions?

You can find me at:

`vijay_garry@hotmail.com`

<https://github.com/vijaygarry>