Module 61CSE215: Object-oriented Programming with Java

Control Statements, Debug, Array, String

Vietnamese-German University
Ngoc Tran, Ph.D.
ngoc.th@vgu.edu.vn

Binh Duong, 2025

Content

- Control Statements
  - Selection statements
  - Iteration statements
  - Jump statements.
- Debug
- Array
- String

Selection Statements

- if
- switch

If

- The if statement is conditional branch statement.
- If can be used to route program execution through two different paths.
- Syntax:

```
if(condition)
        statement1;
else
        statement2;
```

Example:

```
int a, b;
//...
if(a < b) a = 0;
else b = 0;
```

If

- One statement within if without block

```
boolean dataAvailable;
//...
if (dataAvailable)
  ProcessData();
else
  waitForMoreData();
```

- many statements within if with block

```
int bytesAvailable;
// ...
if (bytesAvailable > 0) {
  ProcessData();
  bytesAvailable -= n;
} else
  waitForMoreData();
```

- many statements within if and else with blocks

```
int bytesAvailable;
// ...
if (bytesAvailable > 0) {
  ProcessData();
  bytesAvailable -= n;
} else {
  waitForMoreData();
  bytesAvailable = n;
}
```

Nested If

- An else statement always refers to the nearest if statement that is within the same block as the else, and that is not already associated with an else.
- E.g.,

```
if(i == 10) {
  if(j < 20) a = b;
  if(k > 100) c = d; // this if is
  else a = c;        // associated with this else
}
else a = d;          // this else refers to if(i == 10)
```

The if-else-if Ladder

- The if-else-if Ladder has this following form:

```
if(condition)
   statement;
else if(condition)
   statement;
else if(condition)
   statement;
.
.
.
else
   statement;
```

The if-else-if Ladder

```
public class IfElse {

    public static void main(String[] args) {
        int month = 4;
        String season;

        if (month == 12 || month == 1 || month == 2) {
            season = "Winter";
        } else if (month == 3 || month == 4 || month == 5) {
            season = "Spring";
        } else if (month == 6 || month == 7 || month == 8) {
            season = "Summer";
        } else if (month == 9 || month == 10 || month == 11) {
            season = "Autumn";
        } else {
            season = "Bogus Month";
        }

        System.out.println("April is in the " + season + ".");
    }
}
```

The if-else-if Ladder

```
public class IfElse {

    public static void main(String[] args) {
        int month = 4;
        String season;

        if (month == 12 || month == 1 || month == 2) {
            season = "Winter";
        } else if (month == 3 || month == 4 || month == 5) {
            season = "Spring";
        } else if (month == 6 || month == 7 || month == 8) {
            season = "Summer";
        } else if (month == 9 || month == 10 || month == 11) {
            season = "Autumn";
        } else {
            season = "Bogus Month";
        }

        System.out.println("April is in the " + season + ".");
    }
}
```

Here is the output produced by the program:

```
April is in the Spring.
```

Switch

- The switch statement is multiway branch statement.
- Switch provides a better alternative than a large series of if-else-if statements.
- The value of the expression is compared with each of the values in the case statements.
  - If a match is found, the code sequence following that case statement is executed.
  - If none of the constants matches the value of the expression, then the default statement is executed.
  - However, the default statement is optional.
  - If no case matches and no default is present, then no further action is taken.

Switch

- Syntax:

```
switch (expression) {
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
. . .
case valueN :
// statement sequence
break;
default:
// default statement sequence
}
```

Switch

```
public class SampleSwitch {

    public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            switch (i) {
            case 0:
                System.out.println("i is zero.");
                break;
            case 1:
                System.out.println("i is one.");
                break;
            case 2:
                System.out.println("i is two.");
                break;
            case 3:
                System.out.println("i is three.");
                break;

            default:
                System.out.println("i is greater than 3.");
            }
        }
    }
}
```

Switch

```
public class SampleSwitch {

    public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            switch (i) {
            case 0:
                System.out.println("i is zero.");
                break;
            case 1:
                System.out.println("i is one.");

                ntln("i is two.");

                ntln("i is three.");

            default:
                System.out.println("i is greater than 3.");
            }
        }
    }
}
```

The output produced by this program is shown here:

```
i is zero.
i is one.
i is two.
i is three.
i is greater than 3.
i is greater than 3.
```

Switch

```
public class MissingBreak {

    public static void main(String[] args) {
        for (int i = 0; i < 12; i++) {
            switch (i) {
            case 0:
            case 1:
            case 2:
            case 3:
            case 4:
                System.out.println("i is less than 5");
                break;
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
                System.out.println("i is less than 10");
                break;
            default:
                System.out.println("i is 10 or more");
            }
        }
    }
}
```

Switch

```
public class MissingBreak {

    public static void main(String[] args) {
        for (int i = 0; i < 12; i++) {
            switch (i) {
            case 0:
                                                    println("i is less than 5");
                                                    println("i is less than 10");
            default:
                System.out.println("i is 10 or more");
            }
        }
    }
}
```

This program generates the following output:

```
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 10
i is less than 10
i is less than 10
i is less than 10
i is less than 10
i is 10 or more
i is 10 or more
```

Switch

```
public class Switch {

    public static void main(String[] args) {
        int month = 4;

        String season;

        switch (month) {
        case 12:
        case 1:
        case 2:
            season = "Winter";
            break;
        case 3:
        case 4:
        case 5:
            season = "Spring";
            break;
```

```
case 6:
        case 7:
        case 8:
            season = "Summer";
            break;
        case 9:
        case 10:
        case 11:
            season = "Autumn";
            break;
        default:
            season = "Bogus Month";
        }

        System.out.println("April is in the
" + season + ".");
    }
}
```

Switch

```
//Use a string to control a switch statement.
public class StringSwitch {

    public static void main(String[] args) {
        String str = "two";

        switch (str) {
        case "one":
            System.out.println("one");
            break;
        case "two":
            System.out.println("two");
            break;
        case "three":
            System.out.println("three");
            break;
        default:
            System.out.println("no match");
            break;
        }
    }
}
```

Nested switch Statements

```
switch(count) {
  case 1:
    switch(target) { // nested switch
      case 0:
        System.out.println("target is zero");
        break;
      case 1: // no conflicts with outer switch
        System.out.println("target is one");
        break;
    }
    break;
  case 2: // ...
```

Exercises

1. Solve quadratic equations $ax^2 + bx + c = 0$.
2. Write a Java program to find the number of days in a month.
3. Write a Java program that takes a year from user and print whether that year is a leap year or not.

   Hint: if the year is divisible by 4 and it is not a century year (which is divisible by 100), it is a leap year. Otherwise, if the year is divisible by 400, the year is a leap year. Otherwise, it is not a leap year.

Exercises

4. Input a number x. Check if x is a valid week day. If yes, print out the day name in characters of x at the console.

   Hint: x = 8 => Sunday

   x = 1 => Error

   x = 3 => Tuesday

5. Input a date including day d, month m. Print verbal description the date.

   E.g., Input d = 10, m = 2 => Output: The tenth of February

Iteration Statements

- While
- Do… while
- For
- For each…

These statements create what we commonly call loops.

A loop repeatedly executes the same set of instructions until a termination condition is met.

While

- Syntax:

```
while(condition) {
    // body of loop
}
```

- The condition can be any Boolean expression.
- The body of the loop will be executed while the conditional expression is true.
- When condition becomes false, control breaks the loop.

While

Example

```
// Demonstrate the while loop.
class While {
  public static void main(String args[]) {
    int n = 10;

    while(n > 0) {
      System.out.println("tick " + n);
      n--;
    }
  }
}
```

While

Example

```
// Demonstrate the while loop.
class While {
  public static void main(String args[]) {
    int n = 10;

    while(n > 0) {
      System.out.println("tick " + n);
      n--;
    }
  }
}
```

When you run this program, it will "tick" ten times:

```
tick 10
tick 9
tick 8
tick 7
tick 6
tick 5
tick 4
tick 3
tick 2
tick 1
```

While

- The body of the while can be empty because a null statement is syntactically valid in Java.

```
// The target of a loop can be empty.
class NoBody {
  public static void main(String args[]) {
    int i, j;

    i = 100;
    j = 200;

    // find midpoint between i and j
    while(++i < --j); // no body in this loop

    System.out.println("Midpoint is " + i);
  }
}
```

While

- The body of the while can be empty because a null statement is syntactically valid in Java.

```
// The target of a loop can be empty
```

This program finds the midpoint between **i** and **j**. It generates the following output:

```
Midpoint is 150
```

```
    // find midpoint between i and j
    while(++i < --j); // no body in this loop

    System.out.println("Midpoint is " + i);
  }
}
```

Do – While

- The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.
- Syntax:

```
do {
    // body of loop
} while(condition)
```

- If this expression is true, the loop will repeat. Otherwise, the loop terminates.

Do – While

- Example

```
// Demonstrate the do-while loop.
class DoWhile {
  public static void main(String args[]) {
    int n = 10;

    do {
      System.out.println("tick " + n);
      n--;
    } while(n > 0);
  }
}
```

Do – While

- Example

```
// Demonstrate the do-while loop.
class DoWhile {
  public static void main(String args[]) {
    int n = 10;

    do {
      System.out.println("tick " + n);
      n--;
    } while(n > 0);
  }
}
```

```
do {
  System.out.println("tick " + n);
} while(--n > 0);
```

Do – While

```java
import java.io.IOException;

public class Menu {

    public static void main(String[] args) throws IOException
{
        char choice;

        do {
            System.out.println("Help on: ");
            System.out.println("   1. if");
            System.out.println("   2. switch");
            System.out.println("   3. while");
            System.out.println("   4. do-while");
            System.out.println("   5. for\n");
            System.out.println("Choose one:");
            choice = (char) System.in.read();
        } while (choice < '1' || choice > '5');

        System.out.println("\n");

        switch (choice) {
        case '1':
            System.out.println("The if:\n");
            System.out.println("if(condition) statement;");
            System.out.println("else statement;");
            break;
        case '2':
            System.out.println("The switch:\n");
            System.out.println("switch(express) {");
            System.out.println("case constant:");
            System.out.println("    statement sequence");
            System.out.println("    break;");
            System.out.println("    //...");
            System.out.println("}");
            break;
        case '3':
            System.out.println("The while:\n");
            System.out.println("while(condition) statement;");
            break;
        case '4':
            System.out.println("The do-while:\n");
            System.out.println("do {");
            System.out.println("    statement;");
            System.out.println("} while (condition);");
            break;
        case '5':
            System.out.println("The for:\n");
            System.out.println("for(init; condition; iteration)");
            System.out.println("    statement;");
            break;
        }
    }
}
```

Do – While

```
public class Menu {

    public static void main(String[]
{
        char choice;

        do {
            System.out.println("
            System.out.println("
            System.out.println("
            System.out.println("
            System.out.println("
            System.out.println("
            System.out.println("
            choice = (char) Syst
        } while (choice < '1' ||

        System.out.println("\n")

        switch (choice) {
        case '1':
            System.out.println("
            System.out.println("
            System.out.println("el
            break;
```

Here is a sample run produced by this program:

```
Help on:
  1. if
  2. switch
  3. while
  4. do-while
  5. for
Choose one:
4
The do-while:
do {
  statement;
} while (condition);
```

```
        case '2':
            t.println("The switch:\n");
            println("switch(express) {");
            rintln("case constant:");
            intln("    statement sequence");
            intln("    break;");
            intln("    //...");
            intln("}");

            intln("The while:\n");
            intln("while(condition) statement;");

            intln("The do-while:\n");
            intln("do {");
            intln("    statement;");
            intln("} while (condition);");

            intln("The for:\n");
            rintln("for(init; condition; iteration)");
            println("    statement;");
        }
    }
}
```

For

- Syntax:

```
for(initialization; condition; iteration){
        // body
}
```

- The initialization expression is executed only once.
- The condition must be a Boolean expression used to control the loop.
  - If this condition expression is true, the body of the loop is executed.
  - If it is false, the loop terminates.
- The iteration is executed.
  - This expression increments or decrements the loop control variable.
  - The loop iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass.
  - This process repeats until the controlling expression is false.

For

Example

```
// Demonstrate the for loop.
class ForTick {
  public static void main(String args[]) {
    int n;

    for(n=10; n>0; n--)
      System.out.println("tick " + n);
  }
}
```

For

- Declaring loop control variables inside the for loop

```
// Declare a loop control variable inside the for.
class ForTick {
  public static void main(String args[]) {

    // here, n is declared inside of the for loop
    for(int n=10; n>0; n--)
      System.out.println("tick " + n);
  }
}
```

For

- Here is a simple program that tests for prime numbers.
- Notice that the loop control variable, i, is declared inside the for since it is not needed elsewhere.

```
public class FindPrime {

    public static void main(String[] args) {
        int num;
        boolean isPrime;

        num = 14;

        if (num < 2)
            isPrime = false;
        else
            isPrime = true;

        for (int i = 2; i <= num / 2; i++) {
            if ((num % i) == 0) {
                isPrime = false;
                break;
            }
        }

        if (isPrime)
            System.out.println("Prime");
        else
            System.out.println("Not Prime");
    }
}
```

For

- Using the Comma: To include more than one statement in the initialization and iteration portions of the for loop.

```
class Sample {
  public static void main(String args[]) {
    int a, b;

    b = 4;
    for(a=1; a<b; a++) {
      System.out.println("a = " + a);
      System.out.println("b = " + b);
      b--;
    }
  }
}
```

For

```
//Using the comma.
public class Comma {

    public static void main(String[] args)
{
        int a, b;

        for (a = 1, b = 4; a < b; a++, b--)
{
            System.out.println("a = " + a);
            System.out.println("b = " + b);
        }
    }
}
```

For

```
//Using the comma.
public class Comma {

    public static void main(String[] args)
{
        int a, b;

        for (a = 1, b = 4; a < b; a++, b--)
{
            System.out.println("a = " + a);
            System.out.println("b = " + b);
        }
    }
}
```

The program generates the following output:

a = 1
b = 4
a = 2
b = 3

For

- Some for loop variations

```
boolean done = false;

for(int i=1; !done; i++) {
  // ...
  if(interrupted()) done = true;
}
```

```
// Parts of the for loop can be empty.
class ForVar {
  public static void main(String args[]) {
    int i;
    boolean done = false;

    i = 0;
    for( ; !done; ) {
      System.out.println("i is " + i);
      if(i == 10) done = true;
      i++;
    }
  }
}
```

For

- This loop will run forever because there is no condition under which it will terminate.

```
for( ; ; ) {
  // ...
}
```

For each

- Syntax

```
for(type itr-var : collection)
    statement-block
```

- type specifies the type.
- itr-var specifies the name of an iteration variable that will receive the elements from a collection, *one at a time*, from beginning to end.

For each

Example

```
int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int sum = 0;

for(int i=0; i < 10; i++) sum += nums[i];
```

```
int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int sum = 0;

for(int x: nums) sum += x;
```

For each

```
public class ForEach {

    public static void main(String[] args) {
        int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int sum = 0;

        for (int x : nums) {
            System.out.println("Value is: " + x);
            sum += x;
        }

        System.out.println("Summation: " + sum);
    }
}
```

For each

```
public class ForEach {

    public static voi
        int nums[] =
        int sum = 0;

        for (int x :
            System.ou
            sum += x;
        }

        System.out.pr
    }
}
```

The output from the program is shown here:

```
Value is: 1
Value is: 2
Value is: 3
Value is: 4
Value is: 5
Value is: 6
Value is: 7
Value is: 8
Value is: 9
Value is: 10
Summation: 55
```

For each

```
public class ForEach2 {

    public static void main(String[] args) {
        int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int sum = 0;

        for (int x : nums) {
            System.out.println("Value is: " + x);
            sum += x;
            if (x == 5)
                break;
        }

        System.out.println("Summation of first 5 elements: " + sum);
    }
}
```

For each

```
public class ForEach2 {

    public static void main(String[] args) {
        int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int sum =

        for (int
            Syste
            sum +
            if (x
                b
        }

        System.out.println("Summation of first 5 elements: " + sum);
    }
}
```

This is the output produced:

```
Value is: 1
Value is: 2
Value is: 3
Value is: 4
Value is: 5
Summation of first 5 elements: 15
```

For each

```
//The for-each loop is essentially read-only.
public class NoChange {

    public static void main(String[] args) {
        int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        for (int x : nums) {
            System.out.print(x + " ");
            x = x * 10;
        }

        System.out.println();

        for (int x : nums) {
            System.out.print(x + " ");
        }

        System.out.println();
    }
}
```

For each

```
//The for-each loop is essentially read-only.
public class NoChange {

    public static void main(String[] args) {
        int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        for (int x : nums) {
            System.out.print(x + " ");
            x = x * 10;
        }

        System.out.println();

        for (int x : nums) {
            System.out.print(x + " ");
        }

        System.out.println();
    }
}
```

The output

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

Iterating Over Multidimensional Arrays

- In Java, multidimensional arrays consist of arrays of arrays.
- E.g., a two-dimensional array is an array of one-dimensional arrays.

Iterating Over Multidimensional Arrays

```
//Use for-each style for on a two-dimensional array.
public class ForEach3 {

    public static void main(String[] args) {
        int sum = 0;
        int nums[][] = new int[3][5];

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 5; j++) {
                nums[i][j] = (i + 1) * (j + 1);
            }
        }

        for (int x[] : nums) {
            for (int y : x) {
                System.out.println("Value is: " + y);
                sum += y;
            }
        }

        System.out.println("Summation: " + sum);
    }
}
```

Iterating Over Multidimensional Arrays

```
//Use for-each style for on a two-dimensional array.
public clas

    public
        int
        int

        for


        }

        for


        }

        Sys
    }
}
```

The output from this program is shown here:

```
Value is: 1
Value is: 2
Value is: 3
Value is: 4
Value is: 5
Value is: 2
Value is: 4
Value is: 6
Value is: 8
Value is: 10
Value is: 3
Value is: 6
Value is: 9
Value is: 12
Value is: 15
Summation: 90
```

Applying the Enhanced for

Example: write a program using a for loop to search an unsorted array for a value. It stops if the value is found

```
//Search an array using for-each style for.
public class Search {

    public static void main(String[] args) {
        int nums[] = { 6, 8, 3, 7, 5, 6, 1, 4 };
        int val = 5;
        boolean found = false;

        for (int x : nums) {
            if (x == val) {
                found = true;
                break;
            }
        }

        if (found) {
            System.out.println("Value found!");
        }
    }
}
```

Nested Loops

```
// Loops may be nested.
class Nested {
  public static void main(String args[]) {
    int i, j;

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

The output produced by this program is shown here:

```
..........
.........
........
.......
......
.....
....
...
..
.
```

Exercises

- Input an integer n at the console.
  1. Display all even and odd numbers less than n.
  2. Display all numbers less than or equal to n are primes. *Hint: a prime is divisiable only by 1 and itself.*
  3. Display Fibonacci sequence of n numbers (n > 2) at the console. *Hint:* $F_1 = 0$, $F_2 = 1$, $F_n = F_{n-1} + F_{n-2}$.
  4. Display the factorial of n. *Hint: Fact(n) = 1*2*3*...*(n-1)*n.*
  5. Input the second integer m. Display the greatest common division (GCD) of n and m.

✓Apply loops statements including for, for each, do… while, while without jump statements.

Jump Statement

- Java supports three jump statements: break, continue, and return.
- These statements transfer control to another part of your program.

Break

- When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

```
//Using break to exit a loop.
public class BreakLoop {

    public static void main(String[] args) {
        for (int i = 0; i < 100; i++) {
            if (i == 10) {
                break;
            }
            System.out.println("i: " + i);
        }
        System.out.println("Loop complete.");
    }
}
```

Break

- When a break statement is encountere loop, the loop terminated an control resum next statemer the loop.

Output:
i: 0
i: 1
i: 2
i: 3
i: 4
i: 5
i: 6
i: 7
i: 8
i: 9
Loop complete.

```
//Using break to exit a loop.
public class BreakLoop {

    public static void main(String[] args) {
        for (int i = 0; i < 100; i++) {
            if (i == 10) {
                break;
            }
            System.out.println("i: " + i);
        }
        System.out.println("Loop complete.");
    }
}
```

Break

```
//Using break to exit a while loop.
public class BreakLoop2 {

    public static void main(String[] args) {
        int i = 0;

        while (i < 100) {
            if (i == 10) {
                break;
            }
            System.out.println("i: " + i);
            i++;
        }
        System.out.println("Loop complete.");
    }
}
```

Break

```
//Using break with nested loops.
public class BreakLoop3 {

    public static void main(String[] args) {
        for (int i = 0; i < 3; i++) {
            System.out.print("Pass " + i + ": ");
            for (int j = 0; j < 100; j++) {
                if (j == 10) {
                    break;
                }
                System.out.print(j + " ");
            }
            System.out.println();
        }
        System.out.println("Loops complete.");
    }
}
```

Break

```
//Using break with nested loops.
public class BreakLoop3 {

    public static void main(String[] args) {
        for (int i = 0; i < 3; i++) {
            System.out.print("Pass " + i + ": ");
            for (int j = 0; j < 100; j++) {
                if (j == 10) {
                    break;
                }
                System.out.print(j + " ");
            }
            System.out.println();
        }
        System.out.println("Loops complete.");
    }
}
```

This program generates the following output:

```
Pass 0: 0 1 2 3 4 5 6 7 8 9
Pass 1: 0 1 2 3 4 5 6 7 8 9
Pass 2: 0 1 2 3 4 5 6 7 8 9
Loops complete.
```

Using Break as a Form of Goto

```
public class Break {

    public static void main(String[] args) {
        boolean t = true;

        first: {
            second: {
                third: {
                    System.out.println("Before the break.");
                    if (t)
                        break second;
                    System.out.println("this won't execute");
                }
                System.out.println("this won't execute");
            }
            System.out.println("This is after second block.");
        }
    }
}
```

Using Break as a Form of Goto

```
public class Break {

    public static void main(String[] args) {
        boolean t = true;

        first: {
            second: {
                third: {
                    System.out.println("Before the break.");
                    if (t)
                        break second;
                    System.out.println("this won't execute");
                }
                System.out.println("this won't execute");
            }
            System.out.println("This is after second block.");
        }
    }
}
```

Running this program generates the following output:

```
Before the break.
This is after second block.
```

Break

```
//Using break to exit from nested loops.
public class BreakLoop4 {

    public static void main(String[] args) {
        outer: for (int i = 0; i < 3; i++) {
            System.out.print("Pass " + i + ": ");
            for (int j = 0; i < 100; j++) {
                if (j == 10)
                    break outer;
                System.out.print(j + " ");
            }
            System.out.println("This will not print");
        }
        System.out.println("Loops complete.");
    }
}
```

Break

```
//Using break to exit from nested loops.
public class BreakLoop4 {

    public static void main(String[] args) {
        outer: for (int i = 0; i < 3; i++) {
            System.out.print("Pass " + i + ": ");
            for (int j = 0; i < 100; j++) {
                if (j == 10)
                    break outer;
                System.out.print(j
            }
            System.out.println("This will not print");
        }
        System.out.println("Loops complete.");
    }
}
```

This program generates the following output:

```
Pass 0: 0 1 2 3 4 5 6 7 8 9 Loops complete.
```

Break

- Cannot break to any label which is not defined for an enclosing block. E.g., the following program is invalid and will not compile

```
//This program contains an error.
public class BreakErr {

    public static void main(String[] args) {
        one: for (int i = 0; i < 3; i++) {
            System.out.print("Pass " + i + ": ");
        }

        for (int j = 0; j < 100; j++) {
            if (j == 10)
                break one; // WRONG
            System.out.print(j + " ");
        }
    }
}
```

continue

- In while and do-while loops, a continue statement causes control to be transferred directly to the conditional expression that controls the loop.
- In a for loop, control goes first to the iteration portion of the for statement and then to the conditional expression.
- For all three loops, any intermediate code is bypassed.

continue

```
//Demonstrate continue.
public class Continue {

    public static void main(String[] args)
{
        for (int i = 0; i < 10; i++) {
            System.out.print(i + " ");
            if (i % 2 == 0)
                continue;
            System.out.println("");
        }
    }
}
```

- This code uses the % operator to check if i is even. If it is, the loop continues without printing a newline.

continue

```
//Demonstrate continue.
public class Continue {

    public static void main(String[] args)
{
        for (int i = 0; i < 10; i++) {
            System.out.print(i + " ");
            if (i % 2 == 0)
                continue;
            System.out.println("");
        }
    }
}
```

Output

```
0 1
2 3
4 5
6 7
8 9
```

continue

- Using continue to print a triangular multiplication table for 0 through 9

```
//Using continue with a label.
public class ContinueLabel {

    public static void main(String[] args) {
        outer: for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                if (j > i) {
                    System.out.println();
                    continue outer;
                }
                System.out.print(" " + (i * j));
            }
            System.out.println();
        }
    }
}
```

continue

- Using continue to print a triangular multiplication table for 0 through 9

```
//Using continue with a label.
public class ContinueLabel {

    public static void main(String[] args) {
        outer: for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                if (j > i) {
                    System.out.println();
                    continue outer;
                }
                System.out.print(" " + (i * j));
            }
            System.out.println();
        }
    }
}
```

Output:

```
0
0 1
0 2 4
0 3 6 9
0 4 8 12 16
0 5 10 15 20 25
0 6 12 18 24 30 36
0 7 14 21 28 35 42 49
0 8 16 24 32 40 48 56 64
0 9 18 27 36 45 54 63 72 81
```

return

- At any time in a method, the return statement immediately terminates the method in which it is executed.

```
public class Return {

    public static void main(String[] args) {
        boolean t = true;

        System.out.println("Before the return.");

        if (t) {
            return;
        }

        System.out.println("This won't execute");
    }
}
```

return

- At any time in a method, the return statement immediately terminates the method in which it is executed.

```
public class Return {

    public static void main(String[] args) {
        boolean t = true;

        System.out.println("Before the return.");

        if (t) {
            return;
        }

        System.out.println("This won't execute");
    }
}
```

The output from this program is shown here:

```
Before the return.
```

Array

- An array is a group of like-typed variables that are referred to by a common name.
- Arrays of any type can be created and may have one or more dimensions.
- A specific element in an array is accessed by its index.

One-Dimensional Arrays

- A one-dimensional array is, essentially, a list of like-typed variables.
- Syntax:

```
type var-name[];
```

- E.g., the following declares an array named month_days with the type "array of int":

```
int month_days[];
```

- This declaration establishes the `month_days` to be an array variable, but there is no array actually exists.
- To link `month_days` with a physical array of integers, you must allocate one using new and assign it to `month_days`. new is a special operator that allocates memory.

One-Dimensional Arrays

- Syntax:

```
array-var = new type [size];
```

  - type specifies the type of data being allocated,
  - size specifies the number of elements in the array,
  - array-var is the array variable that is linked to the array
- The elements in the array allocated by new will automatically be initialized to zero (for numeric types), false (for boolean), or null (for reference types, which are described in a later chapter).
- E.g.,`month_days = new int[12];`

One-Dimensional Arrays

- After an array is allocated, the specific elements in the array can be accessed by specifying its index within square brackets, [].
- The array index starts at zero.
- E.g., `month_days[1] = 28;`
- E.g., The next line displays the value stored at index 3:

```
System.out.println(month_days[3]);
```

One-Dimensional Arrays

```
//Demonstrate a one-dimensional array.
public class Array {

    public static void main(String[] args) {
        int month_days[];
        month_days = new int[12];
        month_days[0] = 31;
        month_days[1] = 28;
        month_days[2] = 31;
        month_days[3] = 30;
        month_days[4] = 31;
        month_days[5] = 30;
        month_days[6] = 31;
        month_days[7] = 31;
        month_days[8] = 30;
        month_days[9] = 31;
        month_days[10] = 30;
        month_days[10] = 31;
        System.out.println("April has " +
month_days[3] + " days.");
    }
}
```

When you run this program, it prints the number of days in April.
As mentioned, Java array indexes start with **zero**, so the number of days in April is month_days[3] or 30.

One-Dimensional Arrays

```
//An improved version of the previous program.
public class AutoArray {

    public static void main(String[] args) {
        int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
        System.out.println("April has " + month_days[12] + " days.");
    }
}
```

- When you run this program, you see the same output as that generated by the previous version.

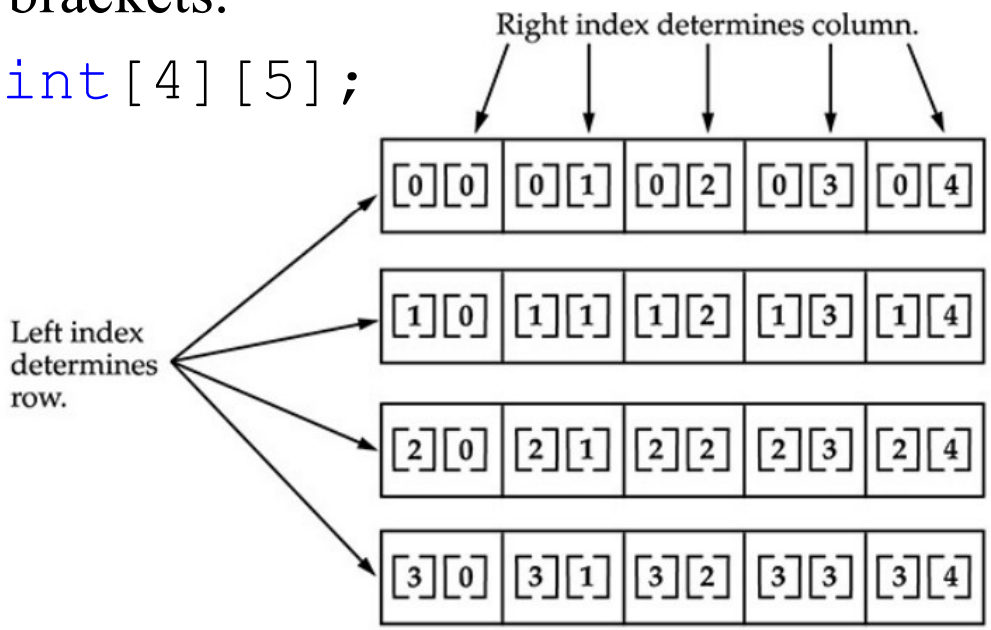
One-Dimensional Arrays

```
//Average an array of values.
public class Average {

    public static void main(String[] args) {
        double nums[] = { 10.1, 11.2, 12.3, 13.4, 14.5 };
        double result = 0;
        int i;
        for (i = 0; i < 5; i++) {
            result = result + nums[i];
        }
        System.out.println("Average is " + result / 5);
    }
}
```

- The Java run-time system will check that all array indexes are in the correct range.
- E.g., the run-time system will check the value of each index into nums to make sure that it is between **0** and **4** inclusive.
- If you try to access elements outside the range of the array (negative numbers or numbers greater than the size of the array), you will cause a run-time error.

Multidimensional Arrays

- Multidimensional arrays are implemented as arrays of arrays.
- To declare a multidimensional array variable, specify each additional index using another set of square brackets.
- E.g., `int twoD[][] = new int[4][5];`
- This allocates a 4 by 5 array and assigns it to twoD. Internally, this matrix is implemented as an array of arrays of int.



Given: int twoD [ ] [ ] = new int [4] [5] ;

Multidimensional Arrays

```
//Demonstrate a two-dimensional array.
public class TwoDArray {

    public static void main(String[] args) {
        int twoD[][] = new int[4][5];
        int i, j, k = 0;

        for (i = 0; i < 4; i++) {
            for (j = 0; j < 5; j++) {
                twoD[i][j] = k;
                k++;
            }
        }

        for (i = 0; i < 4; i++) {
            for (j = 0; j < 5; j++) {
                System.out.print(twoD[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

- When you allocate memory for a multidimensional array, you need only specify the memory for the first (leftmost) dimension.

Multidimensional Arrays

```
//Demonstrate a two-dimensional array.
public class TwoDArray {

    public static void main(String[] args) {
        int twoD[][] = new int[4][5];
        int i, j, k = 0;

        for (i = 0; i < 4; i++) {
            for (j = 0; j < 5; j++) {
                twoD[i][j] = k;
                k++;
            }
        }

        for (i = 0; i < 4; i++) {
            for (j = 0; j < 5; j++) {
                System.out.print(twoD[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

This program generates the following output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

Multidimensional Arrays

```
//Manually allocate differing size second dimensions.
public class TwoDAgain {

    public static void main(String[] args) {
        int twoD[][] = new int[4][];
        twoD[0] = new int[1];
        twoD[1] = new int[2];
        twoD[2] = new int[3];
        twoD[3] = new int[4];

        int i, j, k = 0;

        for (i = 0; i < 4; i++) {
            for (j = 0; j < i + 1; j++) {
                twoD[i][j] = k;
                k++;
            }
        }

        for (i = 0; i < 4; i++) {
            for (j = 0; j < i + 1; j++) {
                System.out.print(twoD[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

- You can allocate the remaining dimensions separately too.

Multidimensional Arrays

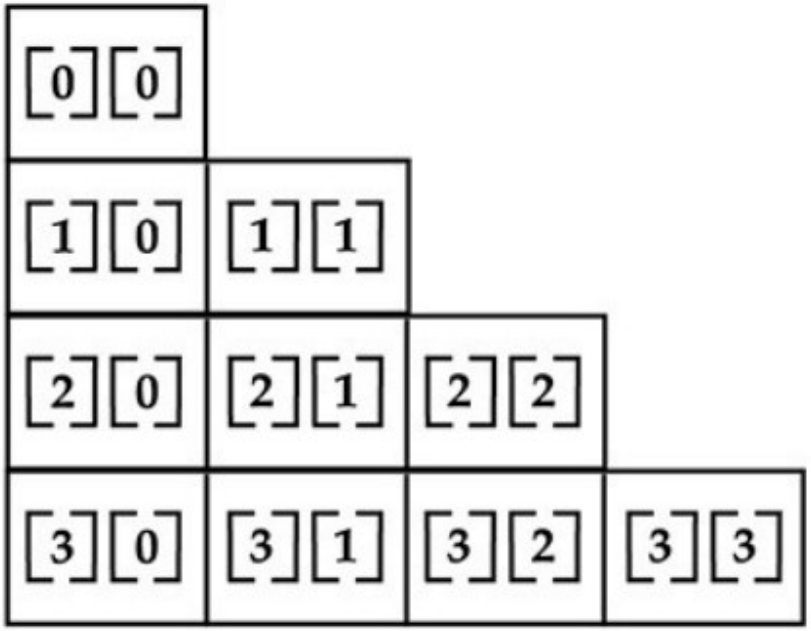
```
//Manually allocate differing size second dimensions.
public class TwoDAgain {

    public static void main(String[] args) {
        int twoD[][] = new int[4][];
        twoD[0] = new int[1];
        twoD[1] = new int[2];
        twoD[2] = new int[3];
        twoD[3] = new int[4];

        int i, j, k = 0;

        for (i = 0; i < 4; i++) {
            for (j = 0; j < i + 1; j++) {
                twoD[i][j] = k;
                k++;
            }
        }

        for (i = 0; i < 4; i++) {
            for (j = 0; j < i + 1; j++) {
                System.out.print(twoD[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

This program generates the following output:

```
0
1 2
3 4 5
6 7 8 9
```

Multidimensional Arrays

- The array created by the program `TwoDAgain` looks like this:



Multidimensional Arrays

- To initialize multidimensional arrays, simply enclose each dimension's initializer within its own set of curly braces.
- E.g., create a matrix where each element contains the product of the row and column indexes.

Multidimensional Arrays

```
//Initialize a two-dimensional array.
public class Matrix {

    public static void main(String[] args) {
        double m[][] = {   { 0 * 0, 1 * 0, 2 * 0, 3 * 0 },
                           { 0 * 1, 1 * 1, 2 * 1, 3 * 1 },
                           { 0 * 2, 1 * 2, 2 * 2, 3 * 2 },
                           { 0 * 3, 1 * 3, 2 * 3, 3 * 3 }
                       };
        int i, j;

        for (i = 0; i < 4; i++) {
            for (j = 0; j < 4; j++) {
                System.out.print(m[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Multidimensional Arrays

```
//Initialize a two-dimensional array.
public class Matrix {

    public static void main(String[] args) {
        double m[][] = {   { 0 * 0, 1 * 0, 2 * 0, 3 * 0 },
                           { 0 * 1, 1 * 1, 2 * 1, 3 * 1 },

        int

        for

            }
            System.out.println();
        }
    }
}
```

When you run this program, you will get the following output:

```
0.0 0.0 0.0 0.0
0.0 1.0 2.0 3.0
0.0 2.0 4.0 6.0
0.0 3.0 6.0 9.0
```

Multidimensional Arrays

- The following program creates a 3 by 4 by 5, three-dimensional array.
- It then loads each element with the product of its indexes

```
//Demonstrate a three-dimensional array.
public class ThreeDMatrix {

    public static void main(String[] args) {
        int threeD[][][] = new int[3][4][5];
        int i, j, k;

        for (i = 0; i < 3; i++) {
            for (j = 0; j < 4; j++) {
                for (k = 0; k < 5; k++) {
                    threeD[i][j][k] = i * j * k;
                }
            }
        }

        for (i = 0; i < 3; i++) {
            for (j = 0; j < 4; j++) {
                for (k = 0; k < 5; k++) {
                    System.out.print(threeD[i][j][k] + " ");
                }
                System.out.println();
            }
            System.out.println();
        }
    }
}
```

Multidimensional Arrays

```
//Demonstrate a three-dimensional array.
public class ThreeDMatrix {

    public static void main(String[] args) {
        int threeD[][][] = new int[3][4][5];

                                    3; i++) {
                                    j < 4; j++) {
                                = 0; k < 5; k++) {
                                eeD[i][j][k] = i * j * k;



                                    3; i++) {
                                    j < 4; j++) {
                                = 0; k < 5; k++) {
                                tem.out.print(threeD[i][j][k] + " ");

                                out.println();
                            }
                            System.out.println();
                        }
                    }
                }
```

This program generates the following output:

```
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12

0 0 0 0 0
0 2 4 6 8
0 4 8 12 16
0 6 12 18 24
```

Exercises

1. Input an array A of integers at the console. Display A.
2. Input an array A with the number of elements n at the console. Reverse the array and display the result to the console.
3. Input 2 arrays A, B with the size n. Calculate the addition of A and B. Output the resulted array C at the console.
4. Input an n x m A at the console, where n is the number of rows and m is the number of columns. Display A at the console.
5. Input 2 n x m matrices A, B with n, m are respectively the number of rows and columns. Calculate the addition and subtraction of A and B. Output the results at the console.

Solution Exercise 1)

```java
import java.util.Scanner;
public class ArrayInputExample1
{
public static void main(String[] args)
{
int n;
Scanner sc=new Scanner(System.in);
System.out.print("Enter the number of elements you want to store: ");
//reading the number of elements from the that we want to enter
n=sc.nextInt();
//creates an array in the memory of length 10
int[] array = new int[n];
System.out.println("Enter the elements of the array: ");
for(int i=0; i<n; i++)
{
//reading array elements from the user
array[i]=sc.nextInt();
}
System.out.println("Array elements are: ");
// accessing array elements using the for loop
for (int i=0; i<n; i++)
{
System.out.println(array[i]);
}
}
}
```

Exercises

6. Input an array A of integers at the console.
   a) Display A at the console
   b) Sort out A in ascending order
   c) Sort out A in descending order
   d) Find all primes in A and display them at the console.
   e) Create a sub array A1 containing even numbers of A, and a sub array A2 containing odd numbers of A. Display A1, A2 at the console.
7. Input an integer n at the console. Output the array of n Fibonacci numbers at the console.

Exercises

8. Input n as the size of the matrix A. A half bottom of A consists of elements increasing by 1 from 1 to the console. E.g., n = 4

   1

   2 3

   4 5 6

   7 8 9 10

String

- The String type is used to declare string variables.

```
String str = "this is a test";
System.out.println(str);
```

- You can also declare arrays of strings

```
String[] s = new String[3];
```

`String` Methods

Details of String Methods (File or this site)

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	Char
concat() or + operator	Appends a string to the end of another string	String
contains()	Checks whether a string contains a sequence of characters	Boolean
indexOf()	Returns the position of the first found occurrence of specified characters in a string	Int
isEmpty()	Checks whether a string is empty or not	Boolean
length()	Returns the length of a specified string	Int
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
toLowerCase()	Converts a string to lower case letters	String
toUpperCase()	Converts a string to upper case letters	String
trim()	Removes whitespace from both ends of a string	String
substring(index1, index2) or substring(index1)	the characters in this string from index1 (inclusive) to index2 (exclusive); if index2 is omitted, grabs till end of string	String
…	…….	…

Exercises

Write the programs using the available methods of string class with the below requests:

1. Input strings s, s1 at the console. Confirm if s1 is inside s to the console.
2. Input strings s, s1, s2 at the console. Replace s1 by s2.
3. Print after removing duplicates from a given string s.
4. Input an integer n and a string s.
   a) Divide s in n equal parts and display them at the console.
   b) Divide s into n-character substrings and display them at the console.
5. Write a Java program to reverse words in a given string.

Exercises

6. Given an array M containing 3-characters strings (student needs to define the array themselves). Input a string. Find all substrings in s that are elements of M.

E.g.: M = {AAG, TAC, CAT, TGA, TAG}

Input s: AAGGTCAACAT

Substrings in M are AAG, CAT

References

1. Herbert Schildt, “Java - The Complete Reference”, 11th edition, Oracle Press, 2019. ISBN: 978-1-26-044024-9.
2. Kathy Sierra and Bert Bates, “Head First Java”, 2nd Edition, O’reilly Media Publisher, 2005. ISBN: 0596009208.
3. Ian F. Darwin, “Java Cookbook”, 4th Edition, O'Reilly Media, 2020. ISBN: 9781492072584.
4. Ben Evans, David Flanagan, “Java in a Nutshell”, 7th Edition, O'Reilly Media, 2018. ISBN: 9781492037255.
5. Richard L. Halterman, “Object-oriented Programming In Java”, 2008.
6. Java tutorials, https://www.w3schools.com/java/.