انشاء ملف على الحاسوب

Create file in computer

```
package first1;
import java.io.File;
import java.io.IOException;
public class NewClass {
 public static void main(String[] args) {
    File myObj = new File("filename.txt");
    try{
        myObj.createNewFile();
}
catch(IOException e){    }
}
}
```

الكتابة على الملفات

```
package first1;
import java.io.FileWriter;
import java.io.IOException;
public class writeonfile {
  public static void main(String[] args) {
    try {
     FileWriter myWriter = new FileWriter("filename.txt");
     myWriter.write("Files in Java might be tricky, but it is fun enough!");
```

```
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
        }
}
}
```

القراءة من الملفات

```
package first1;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class readfile {
   public static void main(String[] args) {
   try {
     File myObj = new File("filename.txt");
     Scanner myReader = new Scanner(myObj);
     while (myReader.hasNextLine()) {
       String data = myReader.nextLine();
       System.out.println(data);
     }
     myReader.close();
   } catch (FileNotFoundException e) {
     System.out.println("An error occurred.");
   }
```

```
  }
}
```

حذف ملف

```
package first1;
import java.io.File;
public class delfile {
public static void main(String[] args) {
   File myObj = new File("filename.txt");
   if (myObj.delete()) {
     System.out.println("Deleted the file: " + myObj.getName());
   } else {
     System.out.println("Failed to delete the file.");
   }
  }
}
```

العمليات في لغة جافا

Arithmetic Operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Java Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3

Java Comparison Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Java Logical Operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

The if Statement

Use the **if** statement to specify a block of Java code to be executed if a condition is true.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

Example

```
if (20 > 18) {
  System.out.println("20 is greater than 18");
}
```

The **else** Statement

Use the **else** statement to specify a block of code to be executed if the condition is **false**.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

The **else if** Statement

Use the **else if** statement to specify a new condition if the first condition is **false**.

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 {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

Short Hand If...Else

There is also a short-hand if else, which is known as the **ternary operator** because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

Syntax

variable = (condition) ? expressionTrue : expressionFalse;

Example

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```

You can simply write:

Example

```
int time = 20;
String result = (time < 18) ? "Good day." : "Good evening.";
System.out.println(result);
```

Java Switch Statements

Use the **switch** statement to select one of many code blocks to be executed.

Syntax

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

```
    break;
  default:
    // code block
}
int day = 4;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  case 4:
    System.out.println("Thursday");
    break;
  case 5:
    System.out.println("Friday");
    break;
  case 6:
```

```
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
}
// Outputs "Thursday" (day 4)
```

The break Keyword

When Java reaches a **break** keyword, it breaks out of the switch block. This will stop the execution of more code and case testing inside the block. When a match is found, and the job is done, it's time for a break. There is no need for more testing.

The default Keyword

The **default** keyword specifies some code to run if there is no case match:

Example

```
int day = 4;
switch (day) {
  case 6:
    System.out.println("Today is Saturday");
    break;
  case 7:
    System.out.println("Today is Sunday");
    break;
```

```
  default:
    System.out.println("Looking forward to the Weekend");
}
// Outputs "Looking forward to the Weekend"
```

Loops

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

Java While Loop

The **while loop** loops through a block of code as long as a specified condition is **true**:

Syntax

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

Example

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

The Do/While Loop

The **do/while** loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is **true**, then it will repeat the loop as long as the condition is true.

Syntax

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

The example below uses a **do/while** loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
int i = 0;
do {
  System.out.println(i);
  i++;
}
while (i < 5);
```

java For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

Syntax

```
for (statement 1; statement 2; statement 3) {
```

```
  // code block to be executed
}
```

Example

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

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
}
```

Example

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

Java Break

You have already seen the **break** statement used .It was used to **"jump out" of a switch statement**.

The break statement can also be used to jump out of a loop.

This example stops the loop when i is equal to 4:

Example

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  System.out.println(i);
}
```

Java Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

Example

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}
```

Break and Continue in While Loop

You can also use break and continue in while loops:

Break Example

```
int i = 0;
while (i < 10) {
  System.out.println(i);
  i++;
  if (i == 4) {
    break;
  }
}
```

Continue Example

```
int i = 0;
while (i < 10) {
  if (i == 4) {
    i++;
    continue;
  }
  System.out.println(i);
  i++;
}
```

Java Arrays

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

To declare an array, define the variable type with **square brackets**:

String [] cars;

We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

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

To create an array of integers, you could write:

int[] myNum = {10, 20, 30, 40};

Access the Elements of an Array

You access an **array** element by referring to the **index number**.

This statement accesses the value of the first element in cars:

Example

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

System.out.println(**cars[0]**);

// Outputs Volvo

Change an Array Element

To change the value of a specific element, refer to the index number:

Example

Cars [0] = "Opel";

Example

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

cars [0] = "Opel";

System.out.println(cars[0]);

// Now outputs Opel instead of Volvo

Array Length

To find out how many elements an array has, use the length property:

Example

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

System.out.println(cars.length);

// Outputs 4
```

Loop Through an Array

You can loop through the array elements with the for loop, and use the length property to specify how many times the loop should run.

The following example outputs all elements in the cars array:

Example

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

for (int i = 0; i < cars.length; i++) {
  System.out.println(cars[i]);
}
```

Loop Through an Array with For-Each

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

Syntax

```
for (type variable : arrayname) {
  ...
```

}

The following example outputs all elements in the **cars** array, using a "**for-each**" loop:

Example

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

for (String i : cars) {

  System.out.println(i);

}
```

Multidimensional Arrays

A multidimensional array is an array of arrays.

To create a two-dimensional array, add each array within its own set of curly braces:

Example

```
Int [][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers is now an array with two arrays as its elements.

To access the elements of the **myNumbers** array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of **myNumbers**:

Example

```
Int [][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

int x = myNumbers[1][2];

System.out.println(x); // Outputs 7
```

Example

```
public class Main {

  public static void main(String[] args) {

    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

    for (int i = 0; i < myNumbers.length; ++i) {

      for(int j = 0; j < myNumbers[i].length; ++j) {

        System.out.println(myNumbers[i][j]);

      }
    }

  }

}
```