



---

# Lecture 4: Java Methods and Files

Instructor: Xin Wang



---

## CONTENTS OF THIS LESSON

- Method parameters
  - Return value
  - Method overloading
  - Java Recursion
  - Write Files
  - Load Files
  - Delete Files
-



# Method Parameters



- 
- Information can be passed to methods as parameter.
  - Parameters act as variables inside the method.
  - Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want.

```
public class Main {
    static void myMethod(String fname) {
        System.out.println(fname + " Refsnes");
    }
    public static void main(String[] args) {
        myMethod("Liam");
        myMethod("Jenny");
        myMethod("Anja");
    }
}
// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

---

Multiple parameters – The method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

```
public class Main {  
    static void myMethod(String fname, int age) {  
        System.out.println(fname + " is " + age);  
    }  
    public static void main(String[] args) {  
        myMethod("Liam", 5);  
        myMethod("Jenny", 8);  
        myMethod("Anja", 31);  
    }  
}
```

```
// Liam is 5  
// Jenny is 8  
// Anja is 31
```

---



# Return Value



---

Return value - If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method.

```
public class Main {
    static int myMethod(int x, int y) {
        return x + y;
    }
    public static void main(String[] args) {
        System.out.println(myMethod(5, 3));
    }
}
// Outputs 8 (5 + 3)
```

```
public class Main {
    static int myMethod(int x, int y) {
        return x + y;
    }
    public static void main(String[] args) {
        int z = myMethod(5, 3);
        System.out.println(z);
    }
}
// Outputs 8 (5 + 3)
```

---

---

## Method with If ... Else...


```
public class Main {
    // Create a checkAge() method with an integer variable called age
    static void checkAge(int age) {
        // If age is less than 18, print "access denied"
        if (age < 18) {
            System.out.println("Access denied - You are not old enough!");
        } // If age is greater than, or equal to, 18, print "access granted"
        else {
            System.out.println("Access granted - You are old enough!");
        }
    }
}

public static void main(String[] args) {
    checkAge(20); // Call the checkAge method and pass along an age of 20
}

// Outputs "Access granted - You are old enough!"
```

---





# Method Overloading



---

With method overloading, multiple methods can have the same name with different parameters.  
For example:

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y)
```

We can overload the `plusMethod` method to work for both `int` and `double`.

```
static int plusMethodInt(int x, int y) {
    return x + y;
}
static double plusMethodDouble(double x, double y) {
    return x + y;
}
public static void main(String[] args) {
    int myNum1 = plusMethodInt(8, 5);
    double myNum2 = plusMethodDouble(4.3, 6.26);
    System.out.println("int: " + myNum1);
    System.out.println("double: " + myNum2);
}
```

```
int: 13
double: 10.559999999999999
```

---



# Java Recursion



- 
- Recursion is the technique of making a function call itself.
  - Recursion can break complicated problems down into simple problems which are easier to solve.
  - For example, recursion can be used to add a range of numbers together by breaking it down into the simple task of adding two numbers.

Eg. Add all of the numbers up to 10 using recursion.

```
public class Main {
    public static void main(String[] args) {
        int result = sum(10);
        System.out.println(result);
    }
    public static int sum(int k) {
        if (k > 0) {
            return k + sum(k - 1);
        } else {
            return 0;
        }
    }
}
```

Calculation steps:

```
10 + sum(9)
10 + ( 9 + sum(8) )
10 + ( 9 + ( 8 + sum(7) ) )
...
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
```

---

- 
- Recursive functions can also run into the problem of infinite recursion.
  - Infinite recursion is when the function never stops calling itself.
  - Every recursive function should have a halting condition, which is the condition where the function stops calling itself.

Eg. Add all of the numbers between 5 to 10.

```
public class Main {  
    public static void main(String[] args) {  
        int result = sum(5, 10);  
        System.out.println(result);  
    }  
    public static int sum(int start, int end) {  
        if (end > start) {  
            return end + sum(start, end - 1);  
        } else {  
            return end;  
        }  
    }  
}
```

The halting condition for this recursive function is when end is not greater than start.

---



# Write Files



- 
- To create a file in Java, `createNewFile()` method needs to be used.
  - `createNewFile()` method returns a boolean value. If the file was successfully created, it will return true. If the file already exists, it will return false.

```
import java.io.File; // Import the File class
import java.io.IOException; // Import the IOException class to handle errors
public class CreateFile {
    public static void main(String[] args) {
        try {
            File myObj = new File("filename.txt");
            if (myObj.createNewFile()) {
                System.out.println("File created: " + myObj.getName());
            } else {
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

---

- 
- **FileWriter** class can be used with its **write ( )** method to write some text to the file.
  - When you finish writing to the file, **close ( )** method should be used to close the file.

```
import java.io.FileWriter;    // Import the FileWriter class
import java.io.IOException;  // Import the IOException class to handle errors
public class WriteToFile {
    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) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

---





# Load Files



- 
- **Scanner** class can be used to read the contents of the text file.

```
import java.io.File; // Import the File class
import java.io.FileNotFoundException; // Import this class to handle errors
import java.util.Scanner; // Import the Scanner class to read text files
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.");
            e.printStackTrace();
        }
    }
}
```

---



# Delete Files



- 
- `delete ()` method can be used to delete a file in Java.

```
import java.io.File; // Import the File class
public class DeleteFile {
    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.");
        }
    }
}
```



Thank you!  
Any questions?

