Java Programming

Methods

Method Definition

- A **method** is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the **method's** name. Think of a **method** as a subprogram that acts on data and often returns a value.

Method Syntax

- Syntax of a Method:

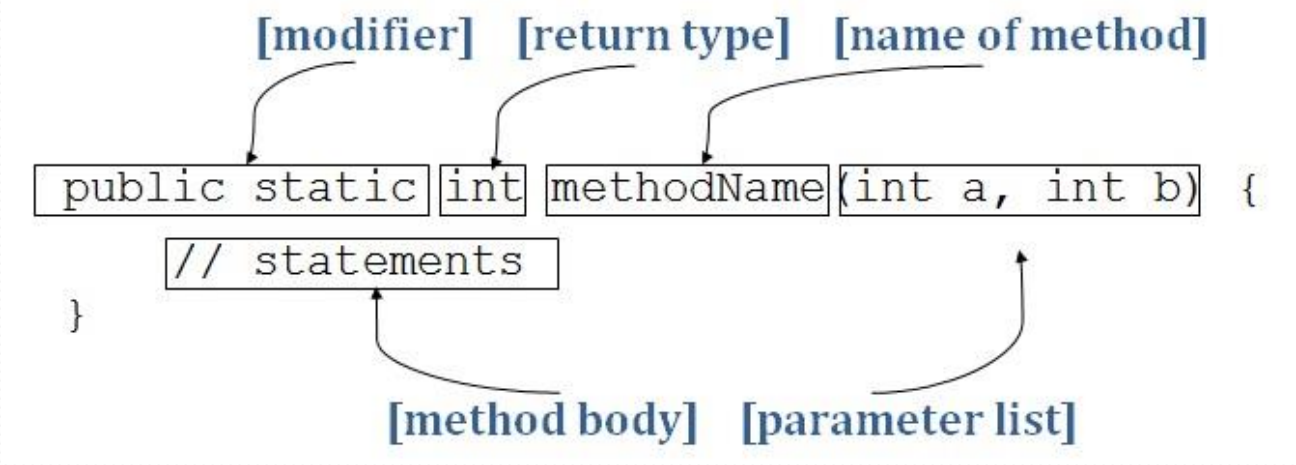
```
public static int methodName(int a, int b) {
    // statements
}
```

  - **public static** : modifier.
  - **int**: return type
  - **methodName**: method name
  - **a, b**: formal parameters
  - **int a, int b**: list of parameters

- Methods are like Procedures or Functions from other languages:
  - **Procedures:** They don't return any value.
  - **Functions:** They return value.

Method Syntax

- The syntax shown above includes:
  - **modifier:** It defines the access type of the method and it is optional to use.
  - **returnType:** Method may return a value.
  - **nameOfMethod:** This is the method name. The method signature consists of the method name and the parameter list.
  - **Parameter List:** The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
  - **method body:** The method body defines what the method does with statements.



Method Overloading

```
public void setHeight(int height) {
      // code
}

public void setHeight(double height) {
      // code
}
```

try…catch…finally

The *try* Block

- The first step in constructing an exception handler is to enclose the code that might throw an exception within a try block.

- In general, a try block looks like the following:

```
try {
    code
}
catch and finally blocks . . .
```

Example *try* Block

```
private List<Integer> list;
private static final int SIZE = 10;

public void writeList() {
    PrintWriter out = null;

    try {
        System.out.println("Entered try statement");
        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + list.get(i));
        }
    }
    catch and finally blocks  . . .
}
```

The *catch* Block

- You associate exception handlers with a try block by providing one or more catch blocks directly after the try block.
- No code can be between the end of the try block and the beginning of the first catch block.

```
try {

} catch (ExceptionType name) {

} catch (ExceptionType name) {

}
```

Example *catch* Blocks

- The following are two exception handlers for the writeList method:

```
try {
    // code
} catch (IndexOutOfBoundsException e) {
    System.err.println("IndexOutOfBoundsException: " + e.getMessage());
} catch (IOException e) {
    System.err.println("Caught IOException: " + e.getMessage());
}
```

Catching More Than One Exception

- Catching More Than One Type of Exception with One Exception Handler

```
catch (IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

The *finally* Block

- The finally block always executes when the try block exits.
- This ensures that the finally block is executed even if an unexpected exception occurs.
- But finally is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break.
- Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

Example *finally* Block

```
finally {
    if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
    } else {
        System.out.println("PrintWriter not open");
    }
}
```

Putting it All Together

```
public void writeList() {
    PrintWriter out = null;

    try {
        System.out.println("Entering" + " try statement");

        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + list.get(i));
        }
    } catch (IndexOutOfBoundsException e) {
        System.err.println("Caught IndexOutOfBoundsException: "
                           +  e.getMessage());

    } catch (IOException e) {
        System.err.println("Caught IOException: " +  e.getMessage());

    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        }
        else {
            System.out.println("PrintWriter not open");
        }
    }
}
```