**METHOD CALL AND RETURN**

In strictly machine terms a *method* is a sequence of operations that a computer program can execute on demand. It is an IPO computer program in miniature – taking data and transforming it to meaningful output.

The method is executed when the computer *calls* it. The computer *passes data*, which we call *arguments*, to variables in the method, which we call *parameters*. The method executes and, when finished, returns to the point from which it was called, giving its output back through what we call a *return value*.

**Example**
Application *MethodCall* calls method *totalMinutes* at line 5. The pictures on the next page illustrate what happens at runtime.

```java
public class MethodCall {
    public static void main( String [] args ) {
        int m = totalMinutes( 2, 30 );
        System.out.println( m );
    }

    public static int totalMinutes( int hours, int minutes ) {
        return (hours * 60) + minutes;
    }
}
```
Call and Return

```java
public static void main( String[] args )
{
    int m = totalMinutes( 2, 30 );
    System.out.println( m );
}

public static int totalMinutes( int hours, int minutes )
{
    return (hours * 60) + minutes;
}
```

Detail of Machine Operations During Call and Return

When the computer executes:
```
int m = totalMinutes( 2, 30 );
```

1: Pass arguments
2: Execute the method
   `(hours * 60) + minutes`
3: Return to caller

Value of `m`: 150
The **method call** is the executable statement that tells the computer to transfer execution to the method body. The method containing the call is the **caller**.

**Example**
Line 5 in the above example contains the method call:

```
totalMinutes( 2, 30 )
```

Line 5 is inside method **main**, which acts the role of the caller.

After returning, the computer resumes executing the caller from the point immediately after the call.

**Example**
In the example above, the **totalMinutes** method returns the value of 150. Execution of the **main** method resumes within line 5 by storing the return value into the variable **m** and continuing to line 6.
## Exercises

Enter the application `MethodCall` into jGRASP and save it to a file. Compile it. Perform the following experiments using jGRASP’s Interactions pane.

1. In jGRASP’s main window, set a breakpoint at line 5:

   ```
   int m = totalMinutes( 2, 30 );
   ```

   To set the breakpoint, move the mouse cursor to the gray bar at the left of your code. It will display a red dot. Click the mouse to set the breakpoint.

2. Start jGRASP’s debugger by clicking the Ladybug button. Execution will pause at the line with the breakpoint. The line is highlighted yellow and marked with a blue arrow:

   ```
   int m = totalMinutes( 2, 30 );
   ```

   Program execution halts immediately _before_ the statement has been executed.

3. Line 5 contains a method call. We want to execute the call and jump into the method.

   To do so, click the Step Into button. Execution steps to the first statement of method `totalMinutes`:

   ```
   return (hours * 60) + minutes;
   ```

   In jGRASP’s Debug pane, what is the value of the method’s arguments `hours` and `minutes`?

4. To execute the return statement, click the Step button. Execution returns back to the call within `main`:

   ```
   int m = totalMinutes( 2, 30 );
   ```

5. Click the Step button again to finish executing line 5.

   In jGRASP’s Debug pane, what is the value of the variable `m`?

6. Repeatedly click the Step button until execution ends.

   What is the output of the application?