EXECUTION OF METHODS

Whenever one method calls a second, the JVM diverts execution control to the second method by jumping to its first statement. This second method may call other methods and so the process continues.

Eventually the JVM finishes executing a method and must return to its caller. This normally occurs when execution reaches either (a) a return statement or (b) the closing right brace (}) of the method’s block.

Example
Let’s follow the execution of the BirthdaySong1 application through its output of the first verse of Happy Birthday.

The JVM starts executing at the first statement of method main, which calls printVerse, which calls printLine.

```
public class BirthdaySong1 {
    public static void main( String [ ] args ) {
        printVerse( "Tom" );
        printVerse( "Dick" );
        printVerse( "Harry" );
    }
    public static void printVerse( String name ) {
        printLine( "to you" );
        printLine( "to you" );
        printLine( "dear " + name );
        printLine( "to you\n" );
    }
    public static void printLine( String ending ) {
        System.out.println( "Happy birthday " + ending );
    }
}
```

The println method within printLine is called, which outputs Happy birthday to you.
**Example, continued**
When execution reaches the closing right brace of `printLine`, it returns to the statement immediately after that which called it.

```java
public class BirthdaySong1 {
    public static void main( String [ ] args ) {
        printVerse( "Tom" );
        printVerse( "Dick" );
        printVerse( "Harry" );
    }

    public static void printVerse( String name ) {
        printLine( "to you" );
        printLine( "to you" );
        printLine( "dear " + name );
        printLine( "to you\n" );
    }

    public static void printLine( String ending ) {
        System.out.println( "Happy birthday " + ending );
    }
}
```

This is another call to `printLine`, causing it to be executed a second time outputting *Happy Birthday to you.*

As `printVerse` continues its execution, method `printLine` is called two more times outputting:

```
Happy birthday dear Tom
Happy birthday to you
```
Having finished all four statements within `printVerse`, execution reaches its closing right brace and so returns to the statement immediately after that which called it.

```java
public class BirthdaySong1 {
    public static void main( String [] args ) {
        printVerse( "Tom" );
        printVerse( "Dick" );
        printVerse( "Harry" );
    }

    public static void printVerse( String name ) {
        printLine( "to you" );
        printLine( "to you" );
        printLine( "dear " + name );
        printLine( "to you\n" );
    }

    public static void printLine( String ending ) {
        System.out.println( "Happy birthday " + ending );
    }
}
```

Execution continues with two more calls to `printVerse` and, in total, eight more calls to `printLine`, which outputs the remainder of the song.
### Exercises

Enter the application *BirthdaySong1* into jGRASP and save it to a file. Compile it. Perform the following experiments using jGRASP’s symbolic debugger.

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

   ```java
   printVerse( "Tom" );
   ```

   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:

   ```java
   5  printVerse( "Tom" );
   6  printVerse( "Dick" );
   ```

   Program execution halts immediately *before* the statement has been executed.

3. **Line 5 calls method printVerse.** 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 `printVerse`:

   ```java
   12  println( "to you" );
   13  println( "to you" );
   ```

4. **The first statement of printVerse is a call to method printLine.** If we click the `Step Into` button again, we’ll execute the call to `printLine` and be at its first statement:

   ```java
   20  System.out.println( "Happy birthday " + ending);
   21  }
   ```

5. **Click the Step button** to execute the output operation. What is the output? Where did the “to you” come from?
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 6.   | The debugger is now at line 21:  
|      | System.out.println( "Happy birthday " + ending );  
|      | Click the Step button to execute the return to `printVerse`. Execution returns to its second statement:  
|      | `println( "to you" );`  
|      | `println( "to you" );` |
| 7.   | Click the Step button to line 13. The debugger executes the entire `printLine` method as a single statement. What is the output? |
| 8.   | Click the Step button two more times to advance to line 16, the end of the `printVerse` method.  
|      | `println( "to you\n" );`  
| 9.   | Click the Step button to execute the return to `main`. Execution returns to its second statement:  
|      | `printVerse( "Tom" );`  
|      | `printVerse( "Dick" );` |
| 10.  | Click the Step button two more times to advance to end of the `main` method and finish printing the song. |
| 11.  | One final click of the Step button executes the return from `main` and terminates the program. |
What is the output of this Java application?

```java
public class MethodPlacementAndExecution
{
    public static void method1()
    {
        System.out.println( "ONE" );
        method2();
        System.out.println( "TWO" );
    }

    public static void main( String[] args )
    {
        System.out.println( "THREE" );
        method2();
        method1();
        System.out.println( "FOUR" );
    }

    public static void method2()
    {
        System.out.println( "FIVE" );
        System.out.println( "SIX" );
    }
}
```