Here are two "procedures"

```java
static void joe() {
    out.print('W');
}
```

```java
public static void main() {
    joe();
    out.print('S');
    joe();
}
```

So, what is a procedure?

**Answer:** something similar to the "main programs" you have already created several times.
Computer Scientists have many different names for the chunk of code known as a procedure:

- procedure
- subroutine
- subprogram
- void function
- static method
- static void member function

And they all "pretty much" mean the same thing.

You need to recognize ALL of these different names.
Here is an example of a simple "main" program:

```java
public static void main (String[] args ) {
    int x = 5;
    out.println("ans="+(x*5));
}
```

Here is the same thing with its parts broken out into separate lines

```java
public static void main (String[] args ) {
    int x = 5;
    out.println("ans="+(x*5));
}
```
Here is the same thing with preliminary explanations

```java
public static void main(String[] args)
{
    int x = 5;
    out.println("ans="+(x*5));
}
```

**VISIBILITY**: this procedure has maximum visibility to other parts of a program. A "main" procedure MUST have this much visibility.

Not explained now - use for ALL of your procedures.

"procedures" do NOT "return" an answer.

Every procedure needs a **name**. This procedure is named "main"

List of **formal parameters** - to be explained later. ALWAYS in a pair ( ) of parentheses

A command sequence
Every procedure has exactly one command sequence - called its BODY. Don't forget the braces {}
Here is a similar anatomy for the other procedure shown on the first page.

```
static void joe() {
    out.print('W');
}
```

- The VISIBILITY of `public` is not required in simple programs - except for the one procedure named "main"
- `static` marks this as being a `procedure` rather than a "member method" - something explained later.
- "procedures" do NOT "return" an answer
- This procedure is named "joe"
- This procedure has NO formal parameters - has () anyway
- BODY of procedure named "joe"
Here is the main procedure shown at the beginning

```java
public static void main (String[] args ) {
    int x = 5;
    out.println("ans="+(x*5));
}
```

Here is a tabular analysis of that same procedure

<table>
<thead>
<tr>
<th>VISIBILITY</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNERSHIP</td>
<td>static</td>
</tr>
<tr>
<td>RETURN TYPE</td>
<td>void</td>
</tr>
<tr>
<td>NAME</td>
<td>main</td>
</tr>
<tr>
<td>FORMAL PARAMETERS</td>
<td>String[] args</td>
</tr>
<tr>
<td>BODY</td>
<td>int x = 5;</td>
</tr>
<tr>
<td></td>
<td>out.println(&quot;ans=&quot;+(x*5));</td>
</tr>
</tbody>
</table>
Here is the other procedure shown at the beginning

```java
static void joe() {
    out.print('W');
}
```

Here is a tabular analysis of that same procedure

<table>
<thead>
<tr>
<th>VISIBILITY</th>
<th>omitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNERSHIP</td>
<td>static</td>
</tr>
<tr>
<td>RETURN TYPE</td>
<td>void</td>
</tr>
<tr>
<td>NAME</td>
<td>joe</td>
</tr>
<tr>
<td>FORMAL PARAMETERS</td>
<td>none</td>
</tr>
<tr>
<td>BODY</td>
<td>out.print('W');</td>
</tr>
</tbody>
</table>
Here is another example procedure

```java
public static int compMer(int age)
{
    int merit = 5*age+3;
    return merit;
}
```

Here is a tabular analysis of that same procedure

<table>
<thead>
<tr>
<th>VISIBILITY</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNERSHIP</td>
<td>static</td>
</tr>
<tr>
<td>RETURN TYPE</td>
<td>int</td>
</tr>
<tr>
<td>NAME</td>
<td>compMer</td>
</tr>
<tr>
<td>FORMAL PARAMETERS</td>
<td>int age</td>
</tr>
<tr>
<td>BODY</td>
<td>int merit = 5*age+3; return merit;</td>
</tr>
</tbody>
</table>
Here is another example procedure

```java
static String reportPctg(int passed, int size) {
    double percent = (0.0 + passed) / size;
    return "Percent passed = "+percent;
}
```

Here is a tabular analysis of that same procedure

<table>
<thead>
<tr>
<th>VISIBILITY</th>
<th>omitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNERSHIP</td>
<td>static</td>
</tr>
<tr>
<td>RETURN TYPE</td>
<td>String</td>
</tr>
<tr>
<td>NAME</td>
<td>reportPctg</td>
</tr>
<tr>
<td>FORMAL PARAMETERS</td>
<td>int passed, int size</td>
</tr>
<tr>
<td>BODY</td>
<td>double percent = (0.0 + passed) / size; return &quot;Percent passed = &quot;+percent;</td>
</tr>
</tbody>
</table>
Question: Where do procedures go?
Answer: For JAVA they go in a **class**.

Here is an example using the two procedures shown at the beginning of this lesson.

```java
class Application {
    static void joe() {
        out.print('W');
    }
    public static void main() {
        joe();
        out.print('S');
        joe();
    }
}
```

Note procedures are indented one tab stop within the class code.

These are the braces marking the beginning and end of the class. They are **NOT** part of a sequence command.

A procedure has its own braces.

Put blank lines BETWEEN procedures but **NOT** INSIDE them.
Question: How are procedures accessed (activated/used/executed/run)?

Answer:
(1) The procedure named "main" is started by Eclipse when you ask for its class to be "run".

(2) Other procedures must be called to get them working.

Question: What does calling mean?

Answer:
One procedure (often main) can request the computer to activate and run another procedure. This is known as a procedure call.

The procedure making the request is known as the "caller".

The procedure being activated by the request is known as the "callee".
One procedure (often main) can request the computer to activate and run another procedure.

The procedure making the request is know as the caller.

The procedure being activated by the request is known as the callee.

The caller waits for the callee to return from doing whatever work the callee performs. It is as if the caller went to sleep while the callee runs. In fact this does not happen.

A CPU is a Central Processing Unit. It is what runs programs and procedures. The simple programs you have written are run by a single CPU.

Hence that CPU cannot be working on two procedures at the same time. That is why the caller seems to sleep while the callee is running.

A procedure activation chart is a diagram that keeps track of the work of the CPU as it runs callee procedures and returns to complete running caller procedures that seemed to go to sleep.
Go back to the two procedures introduced at the beginning of this lesson:

```java
static void joe() {
    out.print('W');
}

public static void main() {
    joe();
    out.print('S');
    joe();
}
```

There are two call commands in the main procedure. Both of them call on the procedure named "joe" to be run by the CPU. Thus "joe" gets run twice.
All procedure call commands have the following GENERIC structure - which has been broken down into separate lines:

```
procedure_name
(
    parameter_value_one,
    parameter_value_two,
    parameter_value_three
)
;
```

- **Name of procedure** - ALWAYS REQUIRED
- **Introduces list of parameter values** - ALWAYS REQUIRED.
- **Parameter value list**
  - It can be empty.
  - It can have just one value.
  - It can have many values.
  - More than one value must be separated by commas (,)
- **Ends list of parameter values** - ALWAYS REQUIRED.
- **End of call command** - ALWAYS REQUIRED.
joe();

workHardAt(5,"digging");

makeGradeTable(myTable, 13, "CSCI 2510");
joe();

workHardAt(5, "digging");

makeGradeTable(
    myTable, 13, "CSCI 2510" );

Parameter list () required - even though there are no parameter values.

Parameter list is required - even though there are no parameter values.

Typical procedure name combines English verb, adverb, and preposition into ONE word (NO BLANKS).

Procedure call can be split over more than one line.  
GOOD PRACTICE has the final ); on the very last line of the call ALL BY ITSELF.  
GOOD PRACTICE puts each parameter value on a line by itself.

Commas at END of line

ALL but first and last line of call command are INDENTED one tab stop.