Local variables

Local variable: A variable declared in a method body.

Form of declaration:

\[
\begin{align*}
\text{<type>} \text{<variable-name>} & \text{ ;} \\
\text{or} & \\
\text{<type>} \text{<variable-name>} & = \text{<expression>} \\
\end{align*}
\]

1. When a local variable is created and destroyed.
2. The scope of a local variable.
3. Guidelines for naming a local variable.
Creation & destruction of local variables

call: \texttt{m(5, 7, 3)}

\begin{align*}
\text{x} & \equiv 5 \\
\text{y} & \equiv 7 \\
\text{z} & \equiv 3 \\
\text{temp} & \equiv \_ \\
\text{temp} & \equiv \_ \\
\end{align*}

All params & local variables created, and arg values stored in params, before execution of method body

\begin{verbatim}
/** = smallest of \texttt{x}, \texttt{y}, and \texttt{z} */

public void m(int x, int y, int z) {
    if (x > y) {
        // Swap \texttt{x} and \texttt{y}.
        int temp;
        temp = x; x = y; y = temp;
    }
    if (x > y) {
        // Swap \texttt{x} and \texttt{z}.
        int temp;
        temp = x; x = z; z = temp;
    }
    return x;
}
\end{verbatim}

Exist as long as body is executed

Destroyed when execution terminates
Scope of local variables

call: p(5, 7)

m | 5 | n | 7 | s | ? | k | ?

**Scope of local variable**: from just after its declaration to end of block in which it is declared.

```java
/** = sum of values in range m..n. 
   Precondition: m <= n+1. */
public void p(int m, int n) {
  int s = m; // inv: s = sum of m..k–1  
  int k;
  for (k = m; k <= n; k = k+1) {
    s = s + k;
  }
  return s;
}
```
**Scope of for-loop counter**

Call: \( p(5, 7) \)

\[
\begin{array}{|c|c|c|c|}
\hline
m & 5 & n & 7 \\
\hline
s & ? & k & ? \\
\hline
\end{array}
\]

**Scope of local variable**: from just after its declaration to end of block in which it is declared.

\[
/** = \text{sum of values in range } m..n. \\
\text{Precondition: } m \leq n+1. */
\]

```java
public void p(int m, int n) {
    int s = m;
    // inv: s = sum of m..k–1
    for (int k = m; k <= n; k = k+1) {
        s = s + k;
    }
    return s;
}
```

**Scope of \( k \)**

```java
k = 10; \text{ illegal}
```
/** = sum of values in range m..n.
   Precondition: m <= n+1. */
public void p(int m, int n) {
    int s = m;
    // inv: s = sum of m..k–1
    for (int k = m; k <= n; k = k+1) {
        s = s + k;
    }

    return s;
}
/** = sum of values in range first_value..last_value.  
   Precondition: first_value <= last_value+1. */
public void p(int first_value, int last_value)   {
   int s= first_value;
   // inv: s = sum of first_value..k–1
   for (int k= first_value; k <= last_value;
       k= k+1) {
       s= s + k;
   }

   return s;
}
Short parameter names simplify — as do short local-variable names

/** = sum of values in range m..n.
   Precondition: m <= n+1. */

public void p(int m, int n) {
  int s = m;
  // inv: s = sum of m..k–1
  for (int k = m; k <= n; k = k+1) {
    s = s + k;
  }
  return s;
}

Short parameter names are better — as long as the specification mentions the parameters appropriately.

Short local-variables names are better — as long the local variables are appropriately described.