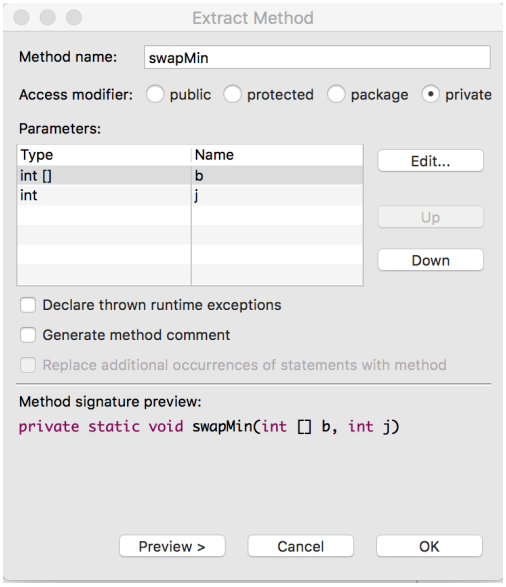
Refactoring selectionSort

Extracting a method

To the right is a version of selectionSort. We want to make a procedure out of the highlighted text, so that the highlighted text will be replaced by:

swapMin(*b*, *j*);

With the text highlighted, we choose menu item *Refactor -> Extract method*. This causes this window to pop up:




```
/** Sort b */
public static void selSort(int[] b) {
   int j= 0;
   // inv P: b[0..j-1] is sorted and
   //        b[0..j-1] <= b[j..]}
   while (j != b.length) {
      int p= min(b, j, b.length-1);
      // {b[p] is min of b[j..]}
      // Swap b[j] and b[p]
      int t= b[j]; b[j]= b[p]; b[p]= t;

      j= j+1;
   }
}
```

We already typed in the method name *swapMin*. The method will be private. And Eclipse has decided that there will be two parameters: array b and integer j. We can edit these and change their order, but what shows is what we want.

After we typed the method name, the Method signature preview appeared.

When we hit OK, the method is changed to what you see on the right, and the new method shown to the left is created. We added the specification.

```
/** Swap min of b[j..] into b[j] */
private static void swapMin(int[] b, int j) {
    int p= min(b, j, b.length-1);
    // {b[p] is minimum of b[j..]}
    // Swap b[j] and b[p]
    int t= b[j]; b[j]= b[p]; b[p]= t;
}
```

```
/** Sort b */
public static void selSort(int[] b) {
   int j= 0;
   // inv P: b[0..j-1] is sorted and
   //        b[0..j-1] <= b[j..]}
   while (j != b.length) {
      swapMin(b, j);

      j= j+1;
   }
}
```

Inlining a method

Method *swapMin*, above, contains a call on function *min*. The body of *swapMin* is simple enough that we want to replace the call on function *min* by its method body, with suitable replacement of parameters, of course. This is called *inlining*. Method *min* appears to the right.

Here's how we perform the inlining:

1. Place the cursor in the name *min* (or select the whole name) in procedure *swapMin*.

```
/** Return the position of min value of b[h..k].
    Precondition: h < k. */
public static int min(int[] b, int h, int k) {
   int p= h; // will contain index of min
   int i= h;
   // {inv: b[p] is the min of b[h..i]}
   while (i != k) {
       i= i+1;
       if (b[i] < b[p]) p= i;
   }
   return p;
}
```

Refactoring selectionSort

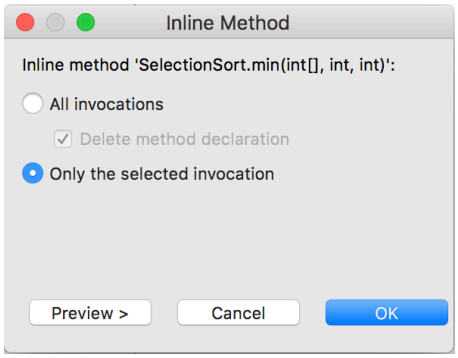
2. Choose menu item *Refactor -> Inline…*.

3. The window to the right pops up. The name of the method to be inlined is *SelectionSort.min*. That's because all these static methods are in class *SelectionSort*.

We choose to inline only this one call.

4. Click button *OK*. That changes method *swapMin* to the method shown below to the left.

Note that Eclipse introduced a local variable p1 to contain the result of the call. Then, after the inlined code, it assigned p1 to p. We don't need this extra local variable and edit it out to get the code on the right. Some editing in this fashion will almost always be required,




```
/** Swap min of b[j..] into b[j] */
private static void swapMin(int[] b, int j) {
   int p1= j; // will contain index of min
   int i= j;
   // {inv: b[p] is the min of b[h..i]}
   while (i != b.length-1) {
       i= i+1;
       if (b[i] < b[p1]) p1= i;
   }
   int p= p1;
   // {b[p] is minimum of b[j..]}
   // Swap b[j] and b[p]
   int t= b[j]; b[j]= b[p]; b[p]= t;
}
```

```
/** Swap min of b[j..] into b[j] */
private static void swapMin(int[] b, int j) {
   int p= j; // will contain index of min
   int i= j;
   // {inv: b[p] is the min of b[h..i]}
   while (i != b.length-1) {
       i= i+1;
       if (b[i] < b[p]) p= i;
   }
   // {b[p] is minimum of b[j..]}
   // Swap b[j] and b[p]
   int t= b[j]; b[j]= b[p]; b[p]= t;
}
```