5 Static Methods: Functions

1 Functions

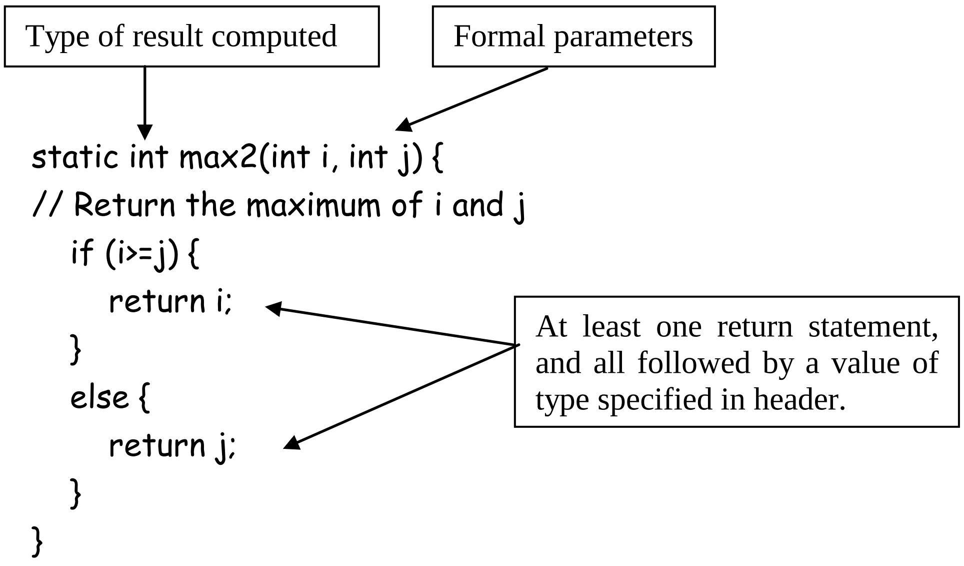
A function does not carry out an action (such as printing or deleting a file) but computes a value of a particular type. The type of the value returned is called the *return type* of the function and is included in the header in place of the `void` of procedures. For example, the return type of the following function is `double`:

```
static double squareRoot(int x)
```

The value is returned to the point of invocation by the function when it executes a return statement. The return statement includes an expression of the return type. For example, if the return type is int, then the function will contain at least one statement of the form

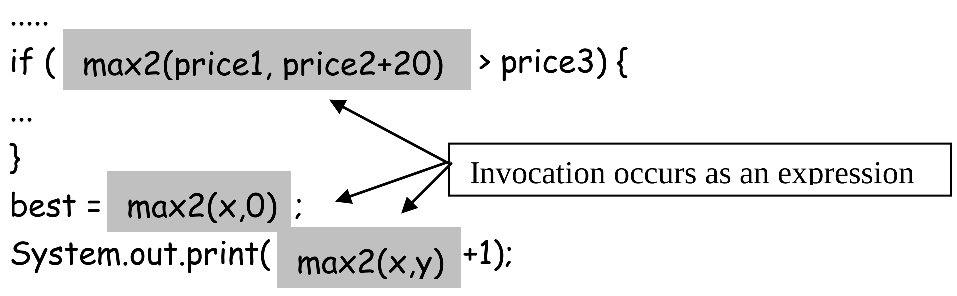
```
return 3*4;
```

(of course, 3*4 here can be replaced with any integer expression). Every function must have at least one return statement ending in an expression, but it may have more than one. In other respects, static functions are declared like static procedures:


Type of result computed
Formal parameters
At least one return statement, and all followed by a value of type specified in header.

```
static int max2(int i, int j) {
// Return the maximum of i and j
   if (i>=j) {
      return i;
   }
   else {
      return j;
   }
}
```

A static function is invoked by using its name (plus arguments in brackets) as an expression of the type stated in the header:



```
.....
if ( max2(price1, price2+20) > price3) {
...
}
best = max2(x,0) ;
System.out.print( max2(x,y) +1);
```

Important! Note the contrast with procedures: procedures are invoked using an invocation *statement*, whereas functions are invoked by the appearance of the name as (or within) an *expression*. When a function invocation `func(e1)`, say, is encountered by the system – in an expression, remember –:it replaces `func(e1)` in the expression with the value obtained by executing `func(e1)` as follows:

1. The parameter in `func` is initialised with `e1`.
2. The body of `func` is executed until a `return` is encountered.
3. When a `return e2` is encountered, the value of e2 is taken as the result of the invocation.

If you do not understand this, study it further in the textbook.

If a static function name is invoked from outside the class where it is declared, it must be prefixed with its class name and a period, just as with procedures. For example, the function round() which we met previously is a static function defined in a class called Math in the Java library, and so a typical invocation is Math.round(3.14).

The return type of a function can be any type at all. For example, the function endOfFile() in class Console is a static function whose return type is boolean. If the return type is a character then we say that the function is "character-valued"; if the return type is boolean then the function is boolean-valued, and so on.

The result-write trap

As with procedures, a common mistake among beginners is to think that parameters of a function must be initialised at the start of the function by reading in values for them from the keyboard. This is not so: parameters acquire their initial values from the corresponding arguments at each invocation.

Another error made by beginners is to believe that the result computed by a function must be printed. This is not so: a function computes a value which it returns it to the caller:

```
static int max2(int i, int j) {
// Return the maximum of i and j
   if (i>=j)
      System.out.print(i);
   else
      System.out.print(j)
}
```

WRONG! This is a function, & so results not printed, but returned!

Example

To illustrate functions in a complete program, we write a program to read the lengths (in centimetres, say) of three rods, and determine whether or not the rods can be used to make a triangle. Three rods form a triangle if the longest length is less than the sum of the other two. An example of input/output for the program is

```
Enter three positive lengths: 30 10 15
You don't have a triangle
```

Another is:

```
Enter three positive lengths: 3 4 5
You have a triangle
```

In the program, we employ three functions, one to compute the maximum of three integers, one to find the sum of the two smallest integers from three given integers, and a boolean-valued function to determine whether three lines constitute a triangle. The invocations of the functions are highlighted with boxes. Note that one of the function max() is invoked from two locations. Of course, we could have written the program using just method main(), but we deliberately employ functions for illustrative purposes, and in any event the use of auxiliary functions gives the program a more understandable structure as a collection of small parts.

```
class Triangle {

   static int max(int i, int j, int k) {
   // Return the maximum of i, j, and k
      if (i>=j && i>=k)  // i is the maximum
         return i;
```

Invoked by sumSmall() & isTriangle()

```
      else if (j>=i && j>=k)  // j is the maximum
         return j;
      else  // k is the maximum
         return k;
   }

   static int sumSmall(int i, int j, int k) {        ← Invoked by isTriangle()
   // Return the sum of the smaller two among i, j, and k
      return i+j+k-max(i,j,k);
   }

   static boolean isTriangle(int i, int j, int k) {    ← Invoked by main()
   // Do positives i, j, k constitute a triangle?
      return max(i, j, k)  <  sumSmall(i, j, k)  ;  // see note below
   }

   public static void main(String[] args) {        ← Invoked by command line
      System.out.print("Enter three positive lengths: ");
      int b = Console.readInt(); int c = Console.readInt(); int d = Console.readInt();
      if (  isTriangle(b, c, d)  )
         System.out.println("You have a triangle");
      else
         System.out.println("You don't have a triangle");
   }
}
```

Some students might be inclined to write the body of isTriangle() as

```
      if (max(i, j, k) < sumSmall(i, j, k))  return true;
      else return false;
```

This is not wrong, but it is needlessly clumsy.

2 Hybrid procedure/functions

Occasionally, a procedure returns a result as well as carrying out an action. The most common example of such hybrids is when a procedure wishes to indicate to the invoker whether or not it carried out its action successfully, and does so by returning a boolean. Below is a small example. It is another version of drawLine() which checks whether the length of the line to be drawn is not too small or too big.

```
static boolean drawLine(int n) { // Draw a line of n *'s
   if (n<0 || n>80)
        return false; // indicate failure
   else {  // draw the line
      int i = 0;
      while (i<n) {
            System.out.print("*"); i++;
      }
      System.out.println();
      return true; // indicate success
   }
}
```

A typical use of this is:

```
public static void main(String[] args) {
        ............
        int num = Console.readInt();
        boolean success = drawLine(n);
        if (!success) System.out.print("No can do");
        ............
}
```

Invocation is an expression....

An invoker who is not interested in the value returned can treat the hybrid just as a normal procedure:

```
drawLine(80);
```

... or invocation is a statement

-- the boolean value returned in this case (which can only be `true`) is discarded.

Very occasionally, a function may effect a change as well as computing a value. The most common example of such hybrids are methods such as Console.readString(), Console.readInt() etc. which read a value from the keyboard and return it, but also display the value entered on the screen. Again, such hybrids can be used as procedures by ignoring the value returned. For example, the *statement*

```
Console.readString();
```

carries out the useful role of pausing the program until the user presses the return key – any text entered on the line is ignored.