CS100J 15 February 2005

Methods: procedures, functions, constructors
You are responsible for: Sections 2.1, 2.2. It's a good idea to do the self-review exercises at the end of 2.2.4, 2.3.4

Rsrecah on spleilng

Aoccdrnig to a rscheearch at Cmabirgde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is that the frsit and lsat ltteer be at the rghit pclae. The rset can be a total mses and you can sitll raed it wouthit porbelm.Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.

Notes on assignment A2

1. Warning from Javadoc. disregard it:

   Warning: warning - First sentence is interpreted to be:

2. In FamilyMemberTester:

```
public void testFirstConstructor(…) {
  FamilyMember  m1= new FamilyMember(…)
   …
   assertEquals(1, FamilyMember.getFamilySize());
}

public void testFirstConstructor(…) {
   FamilyMember  fa= new FamilyMember(…)
   FamilyMember  m1= new FamilyMember(…)
   …
   assertEquals(?, FamilyMember.getFamilySize());
}
```

Note: The CMS allows you to submit an assignment several times.

We grade only the last one submitted.

Notes on assignment A2

3. Testing for **null**

```
/** = "fm is this family member's brother". Precondition: fm not null. */
public boolean isBrother(FamilyMember fm) {
   // No need to test for null. It's the caller's duty not to have fm null.
}

/** = "fm1 and fm2 are not null and fm1 and fm2 are siblings */
public static boolean areSiblings(FamilyMember fm1,
                                  FamilyMember fm2) {
   // The result depends on fm1, fm2 not being null, and the
   // return expression should somehow include that test.

   return …
}
```

In FamilyMemberTester, when testing areSiblings, need to test calls like

areSiblings(m1, **null**)

areSiblings(**null**, m2)

Procedure: a form of method

```
/** Print a, b, and their sum on one line */
public static void print(int a, int b) {
    System.out.println(a + "  " + b + "  " + (a+b));
}
```

header

This procedure call prints out the value of its argument, i.e the expression within ( ) .

See top of page 58 for declaration of procedure and function

Definition: a *parameter* is a variable that is declared within the parentheses of the method header.

Parameters: a and b.

The comment is a ***specification*** of the method. It says WHAT the method does.

Method body: the "block" { … }


```
/** Print b, c, and their sum */
public static void print(int b, int c) {
   System.out.println( b);
   System.out.println(c);
   System.out.println(b + c);
}

/** Print b */
public static void print (int b) {
   System.out.println(b);
}
```

Parameters b and c are *variables*. They are created when the method is called and destroyed when the method call is finished.

The *scope* of a parameter --the places where it can be referenced or used, is the method body itself.

Procedure call

```
/** Print a, b, and their sum on one line */
public static void print(int a, int b) { …}
```

header

See top of page 59 for procedure call

Procedure call has the syntax:

<procedure name> (*<arguments>*) ;

<arguments> is a list of expressions, separated by commas. The type of each expression must match the type of the corresponding parameter of the procedure.

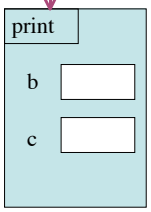
When writing or understanding a call on a method, look only at the specification and not the method body. What does this call do?

print(3+4, 6); ← **call, with arguments 3+4 and 6**

Print 3+4, 6, and their sum on one line.

```
/** Print b, c, and their sum */
public static void print(int b, int c) {
   System.out.println( b);
   System.out.println(c);
   System.out.println(b + c);
}
```

name of method being called



How is a call like this executed? print(5, 6)?

Step 1: Draw a box that contains the parameters (variables).
Step 2: Assign the argument values to the parameters.
Step 3: Execute the method body --execute its statements, one at a time.
Step 4: Erase the box.

A method can call another method

```
/** Print b, c, and the sum of their squares */
public static void print(int b, int c) {
   System.out.println( b);
   System.out.println(c);
   printSum(b*b, c*c);
}

/** Print b + d */
public static void printSum(int  b, int d) {
   System.out.println(b+d);
}
```

Two methods can have a parameter with the same name. The b in print is diffferent from the b in printSum. A parameter (a variable) is in existence only when the method body is being executed.

We execute the call **print(3, 4);**

The if-statement

```
// Set y to the maximum of x and y
  if (x  >=  y)
        y=  x;
```

if-condition

if (*<boolean expression>*)

Syntax

<statement>

Semantics: to execute if-statement: evaluate the if-condition; if it is true, execute the the *<statement>*

The if-else-statement

```
// Set z to the maximum of x and y
  if  (x  >=  y)
        z=  x;
  else  z=  y;
```

if (*<boolean expression>*)

<statement1>

else *<statement2>*

Syntax

Semantics: to execute if-statement:

evaluate the if-condition; if it is true, execute the *<statement1>*; otherwise, execute *<statement2>*

The block: a sequence of statements/declarations enclosed in { }

```
// Swap x and y (if necessary) to place their maximum in x
  if (y > x) {
        temp=  y;
        y=  x;
        x=  temp;
  }
```

from here to here is a block

Statements that you know about:

- **Assignment statement**
- **Procedure call**
- **If-statement**
- **If-else-statement**
- **Block**

Another procedure

```
/** print the smallest of b, c, d */
public static void smallest(int b, int, c, int d) {
        if (b <= c  && b <= d) {
            System.out.println(b);
            return ;
        }
        // { The smallest is either c or d }
        if (c <= d) {
            System.out.println(c);
            return;
        }
        // { the smallest is d }
        System.out.println(d);
}
```

Execution of statement **return;** terminates execution of the procedure body. Nothing else is done in the procedure body.

Assertion: a true-false statement; we are asserting that it is true at the point it appears