Lecture 11 – Wed (lecture 10 – Sat) - Methods

topics:
- review test 2
- ch. 5 Methods:
  - calling methods
  - writing methods
  - argument promotion

1. Writing Methods and Calling Methods

We write methods to modularize our code. A method is a programming module that performs one or more logically-related tasks.

A method declaration specifies the code that will be executed when the method is invoked (or called)

When a method is invoked, the flow of control jumps to the method and executes its code
When complete, the flow returns to the place where the method was called and continues.

The invocation may or may not return a value, depending on how the method was defined. Does the method header have a return type, and does the method contain a RETURN statement?

Terms:
- argument (actual parameter)
- parameter (formal parameter) - includes type
- all primitive types are passed by value
2. Writing Methods - syntax

\textit{method-modifiers return-type method-name(parameter-list)} \{ \\
\hspace{1em} declarations and statements \\
\}

example: a method which has no parameters, and does not return a value

\begin{verbatim}
public static void displayGreeting() {
    JOptionPane.showMessageDialog(null, "Hi, how are you?");
}
\end{verbatim}

example: a method with parameters, but does not return a value

\begin{verbatim}
public static void displayGreeting(String sName) {
    // method variables
    String sGreet1 = "How are you, ";
    String sGreet2 = ", welcome to the game!";
    // output greeting containing name
    JOptionPane.showMessageDialog(null, sGreet1 + sName + sGreet2);
}
\end{verbatim}

the method variables are called local variables. They are local to the method, because they are “scoped to” the method. The parameter sName is also a local variable.

a call to displayGreeting from another method in the class:
\begin{verbatim}
displayGreeting("Mickey");
\end{verbatim}
note that since displayGreeting does not return a value, we can just call it.

example: a method with parameters, which returns a value (note only 1 entrance and 1 exit, the definition of structured code)
public static int findLargest(int nNum1, nNum2, nNum3) {
    // the first input number is the largest so far
    int nLargest = nNum1;
    if (nNum2 > nLargest) // replace with larger if found
        nLargest = nNum2;
    if (nNum3 > nLargest) // replace with larger if found
        nLargest = nNum3;
    return nLargest;
}

Since method findLargest returns a value, a call to it should be part of an expression, which will use the value returned:

int myLargest = findLargest(4, 23, 17);

or

System.out.println(findLargest(5, 6, 3));

If you don't use the return value in an expression, your program will still compile and run, but the value returned will not be saved.

eexample: a method with no parameters, but which returns a value
public static String getSquares() {
    // method variables
    String sSquareList = "The first 10 squares are: ";
    int nSquare;
    // build string
    for (int i = 1; i <= 10; i++) {
        nSquare = i * i;
        sSquareList += nSquare + " ";
    }
    return sSquareList;
}

a call to the method:

JOptionPane.showMessageDialog(null, getSquares());

argument(actual parameter)
parameter(formal parameter) - includes type
all primitive types are passed by value
3. Argument Promotion –
widening coersion is automatic – to a type with more storage
or greater precision
(see chart on page 70)

type(bits) promotion
double(64) none
float(32) double
long(64) float or double
int(32) long, float, or double
char(16) int, long, float, or double
short(16) int, long, float, or double
byte(8) short, int, long, float, or double
boolean(1) none(not a number)
narrowing must be done explicitly with a cast
example - widening:
if a method has a parameter of a double, and an argument of
type int is used, the int ‘s value is temporarily converted to a
double

// method getSquare receives a double,
// and returns the square of the double
double getSquare(double dNum) {
    return dNum * dNum
}

// a call to getSquare

int nNum = 3;
double mySquare = getSquare(nNum); // 3 will be promoted to 3.0
narrowing conversion can only by done with an explicit (cast):

```java
// method getSquare receives an int,
// and returns the square of the int
int getSquare(int nNum) {
  return nNum * nNum
}

// a call to getSquare

double dNum = 3.4;
int mySquare = getSquare((int)dNum);
// 3.4 will be narrowed to 3, with loss of precision
```

4. Scope and Duration of identifiers

An identifier is a name for a variable, variable reference, or method.

Methods and instance variables are said to have class scope. This means that the name has meaning throughout the entire class.

The scope of a variable is limited to the block where it is declared.

- **Instance variable** – declared inside a class block, but outside any method block.
- **Method variable** – declared inside a method block, so its scope is limited to that method.
- **Loop counter variable** – declared inside a control structure, so its scope is limited to the body of the loop.
Duration of Identifiers (author calls “Lifetime”)

The duration of a variable is limited to the time when the flow of control is inside its block. This does not include the time the flow jumps from one method to another, for example:

```java
methodA() {
    int nTotal;
    methodB();
}
methodB() {
    statement; // nTotal is unknown here
    statement;
}
```

If we want methodB to access the variable nTotal, it must be declared outside methodA, within the class block (an instance variable).

Method variables are called **local variables**, and have **automatic duration**. These include both the variables declared in the method, and the parameters of the method. They “die” when the method is exited – their memory space is released back to the system.

Note that variable nTotal doesn’t die until the flow of control returns to methodA(from methodB) and exits methodA.

5. Discuss lab 6 – Programming Challenge 3, p.295
   RectangleArea.
   Case study method from UCBerkeley. A and B versions.