

Anatomy of a method

- Name, possibly qualified by the dot operator
 - If not qualified it's a local class method, inherited or user defined
 - If qualified by a class, it's a static method
- Arguments in parenthesis
- Arguments are separated by commas
- Arguments can be constants, variables, or expressions (that include methods)
 - This provides communication with other methods
 - In addition, a method can define local variables
- Methods may optionally return an identifier
- Example: `System.out.println( Math.sqrt( 900.0 ) );`
 - `Println()` is a qualified method from the System package that takes a string as input and outputs it to the standard device
 - `Sqrt()` is a qualified from the Math package that takes a double and returns a double.
 - Math methods are given on page 171
- A method (or function) can be used anywhere where the return value is valid; it can be used in place of a variable
 - A 2-step process - defined (or declared), then used
- All methods belong to a class
 - Static methods are referenced by the class name -there is only one, ever!
 - Can be referenced as long as the class file is available
 - Non static objects are referenced by an object - there is one for each object
 - Can only be referenced after an object is created
 - If a method is called from another method of the same class, it can be called directly without being referenced
 - Static methods can only call other static methods directly

Why Methods

- Manageability – divide and conquer
- Reusability
- Avoiding repetition
- SW engineering – data hiding
- Along with data, methods are the building blocks of classes and objects

Figure 4.3

- `square()` is a user defined method in the `SquareInt` class
- An integer valued is passed to `square()` and an integer value is returned
 - `void` would indicate that nothing is returned
 - `y` is a local variable in `square()`
- Explanation lines 14 - 15
 - `square( x )` is called with the current value of `x`

- The value returned is converted to a string
- The string is displayed at the location `xPosition` in the x direction and 25 in the y direction from the upper left hand corner of the applet
- The string context can be "forced" with
`" " + square(x)`

Fig 4.4

- Better names: `labelNum1, ..., labelMax, textNum1, ..., textMax`
 - `number1` is a `TextField`; `num1` is an `int`
- `return Math.max(x, Math.max(y, z));` -- Explanation of line 42
 - Two calls to the `max` method, one nested in the other
 - The return value of the "inner" call gets substituted as a parameter in the "outer" call
 - Lines 25-27 enable pressing `enter` on any field - they are all registered to detect events
- Coercion and promotion – if types in a declaration and a call to the method are different, Java uses the promotion rules on page 178 (page 103 has the actual size)
 - A variable can only get promoted to a larger primitive type
 - `long` cannot get promoted to `int`
 - The cast operator has to be used to demote, or convert to a lower (shorter) type
 - Loss of data, truncation, can occur
 - In an expression calculations are done at the highest level of any variable
 - Java will attempt to find an appropriate method to call using the promotion rules on page 178
 - For example, `Math.sqrt()` can be called with an `int` parameter even though it expects a `float` parameter

Skip 4.6 Java API Packages

Figure 4.7

- Scaling a random number

```
Math.random()    // random double between 0 and 1

Math.random() * 6  // random double between 0 and 6

(int) (Math.random() * 6)  // random int between 1 and 5

(int) (Math.random() * 6) + 1; // random integer from 1
                               // to 6
```

- Math methods are given on page 171

- Generalized scaling
`n = shift + (int) ( Math.random() * scale );`
 - `shift` is the shifting value, the first number in the range
 - `scale` is the scaling factor, the “width of the desired range of consecutive integers
- Pseudocode for positioning

Set xPos to 25, yPos to 25

vary i from 1 to 20 (i stores the number of random numbers so far)

generate a random number and display it at xPos, yPos

if i is evenly divisible by 5 (the number of items per line; the remainder is items so far)

increment yPos by 15

reset xPos to 25

else

increment xPos by 15

Fig 4.8

- Rolls a single die 6000 times and calculates the frequency
- Note `start()` method - see page 206
 - Don't forget `repaint()`
 - `setSize()` may not be implemented

Fig 4.9 - Craps.java, see rules on page 185

- `final` – indicates a named constant or read-only variable, line 9
 - Initialized when declared
 - Can't be changed
 - Convention is uppercase
- Note `setEditable( boolean )` method of the `TextField` class, line 28
 - An alternative to output instead of `showStatus()` or `drawstring()`
- `rollDice()` gives a value between 1 and 6 to `die1` and `die2`
 - `WorkSum` is a poor variable name
 - How about `localDieSum`?
- Note definition of a `Button` in line 49
 - `roll` is registered with `ActionListener()`
- `play()` method is in `actionPerformed()` since `roll` is the only component listened for
- `rollDice()` can be “factored” to the beginning of `play()`

Scope and Duration

- Attributes of variables include name, type, size, and value
- Duration refers to the time during which an identifier exists in memory
- An identifier's scope is where the identifier can be referenced
- Automatic duration – variables that are created when a block is entered. They are declared in that block

- The identifiers are destroyed when the block is exited
- Instance variables of a class exist as long as an object exists
 - They are initialized; automatic variables are not
 - They are declared before any methods
- Static duration – exist from the instantiation of an object until the program terminates.
- Methods and instance variables have class scope, and are defined in the body of the class definition.
 - Methods and instance variables can be referenced directly
 - They are global to the methods of the class in which they are defined
- Identifiers (variables) have block scope
 - The body where the identifier is defined.
 - Local variables (non instance variables)
- If a local variable has the same name as an instance variable, it hides or overrides the instance variable. —not a good practice

Fig 4.10

- Local declarations override instance declarations
- `x` is both an instance variable of the class `Scoping` and an instance variable of the method `a()`
- When using variables of the same name, beware!

Skip Recursion - pages 193-203**Method Overloading**

- Unique methods with the same name, distinguished by their signatures.
 - A signature is the name, type and order of the parameter list.
 - The return value is not part of the signature
 - Order of parameters is important and can be used for uniqueness
 - Overloaded methods do not have to have the the same number of parameters
- Used for similar tasks, but different data types
- Figure 4.16, page 204, square method
 - It can take a `double` or an `int`

Applet Methods

- `resize()` allows you to dynamically resize the applet area, SOMETIMES!
 - Many browsers don't implement this.
- See page 206

Converting a String to a double - page 216, Exercise 4.6

- Create a `Double` object using a `String`

```
String doubleString = "123.12345";  
Double doubleObject = new Double(doubleString);
```

- Get the primitive `double` instance value from the `Double` object
`double doubleVal = doubleObject.doubleValue();`
- Also works:
`double doubleVal = Double.parseDouble(doubleString);`