Today's topics

- Recursion
- Arrays
- Databases

Reading
  ➤ Chapter 4

Methods review

- Method definition
  
  ```
  return-value-type method-name( parameter-list )
  {
    declarations and statements
  }
  ```
- Method calling
  ```
  int x = factorial(n-1);
  ```
- Terms
  ➤ declarations, variables, types,
  ➤ arguments (actual parameters), formal parameters

Mechanics of method-calling

1. The values of each argument are computed as part of the operation of the calling-program.

2. The value of each argument is copied into the corresponding formal parameter. If there is more than one argument, the arguments are copied into the parameters in order. The types must match or be able to undergo automatic type conversion.

3. The statements in the function body are evaluated until a return statement appears.

4. The value of the return expression is evaluated and converted, if necessary, to the result type specified for the method.

5. The calling program continues, with the returned value substituted in place of the call.

Methods & Scope

- Scope: the portion of a program in which a variable can be used

- Instance variables
  ➤ Variables defined for the whole applet
  ➤ Shared by all methods in the applet

- Local variables
  ➤ Every variable defined within a function including the formal parameters
  ➤ Scope is the method in which it was defined
  ➤ Collection of variables for each method is called the frame
Recursion

- Function invokes itself
- Dictionary lookup model
  - Lookup word
  - Lookup word from definition
- Decompose the problem
  - base case
  - recursive case which gets closer to solution
- Factorial
  - RecFact.java
- Exponentiation
  - Recurse.java

Recursion Trace

Call factorial(3)
  \( n = 2 \)
  Call factorial(2)
    \( n = 1 \)
    Call factorial(1)
      \( n = 0 \)
      Call factorial(0)
        return 1
        return 1 * (value from call) = 1 * 1 = 1
        return 2 * (value from call) = 2 * 1 = 1
        return 3 * (value from call) = 3 * 2 = 6

Recursion Example #2

- Process of solving large problems by breaking them down into smaller, more simple problems that have identical forms.
- Permutation: arrangement of \( N \) objects in a linear order
  - \( N \) ways to choose first object, \( N-1 \) ways to choose second, ...
  - Generate all permutations of array \( A: \{A, B, C, D, \} \)
    1. Generate permutations of \( \{ABC, D\} \) changing only \( \{BCD\} \)
    2. Generate permutations of \( \{BAC, D\} \) changing only \( \{ACD\} \)
    3. Generate permutations of \( \{CAB, D\} \) changing only \( \{ABD\} \)
    4. Generate permutations of \( \{DAB, C\} \) changing only \( \{ABC\} \)
  - Generate and print all permutations of array \( A \), manipulating only those that occur between some position index and the end of the array

Using arrays

- subscript or index to access element
  \( x[5] = 20; \)
  \( \text{foo.setText("Result is "+ x[5]);} \)
- Often used in loops
  \( \text{int } k = 0; \text{ sum } = 0; \)
  \( \text{while ( } k < 10 \text{ )} \)
  \( \{ \)
    \( k = k + 1; \)
    \( \text{sum } = \text{sum } + \text{name}[k]; \)
  \( \} \)
Creating Arrays

- Declaration
  ```java
double weights[];
```
- Definition
  ```java
weights = new double[50];
```
- Combine
  ```java
double weights[] = new double[50];
```

```java
int num[] = new int[6];
```

- 21
- 4
- 9
- 16
- 25

Arrays & Loops

```java
int k = 2;
while(k<6)
{
    num[k] = k*k;
    k = k+1;
}
```

Database Applications

- What is a database?

  Information retrieval
  - Would like to model phone book
    - Data held in phone book
      - Name
      - Phone Number
      - Address
    - How would we typically like to access?
      - How can we make access efficient?
    - What if we want to access through other fields

Basic Database Functions

- Load data (File I/O would be nice)
- Display data (Complete Dump)
- Query/Search (Selective Information Retrieval)
- Edit/Update (Update or Correct)

  Advanced
  - Wildcard retrieval
  - Output counts instead of data
  - Logical expressions
  - UsedCars.java