CS201 Introduction to Java Programming

Methods and Recursion

Chengyu Sun
California State University, Los Angeles

Method Basics

- Why use method?
- Write a method
- Use, or *call*, a method

```
public class Classname
{
  public static void main( String args[] )
  {
    … …
  }

  // more methods here
  … …
}
```

Sun100n150.java

```
/**
 * Calculate 1+2+3+…+100 and
 * 1+2+3+…+150
 */
public class Sum100n150 {

  public static void main( String args[] )
  {
    int sum100 = 0;
    for( int i=1 ; i <= 100 ; ++i )
      sum100 +=  i;

    int sum150 = 0;
    for( int i=1 ; i <=150 ; ++i )
      sum150 += i;

    // output
  }
}
```

- Writing code performing the same function multiple times is *tedious* and *error-prone*

Another Sum100n150.java

```
public class Sum100n150 {

  public static void main( String args[] )
  {
    int sum100 = sumN( 100 );
    int sum150 = sumN( 150 );
    // output
  }

  public static int sumN( int n )
  {
    int sum = 0;
    for( int i=1 ; i <=n ; ++i )
      sum += n;

    return sum;
  }
}
```

- Pack certain functionality in a re-usable piece of code – *method*

Method

- Header
  - Access modifier
    - public
    - private
    - Protected
  - `static`
  - Return type
  - Name
  - Parameter list
- Body

Header (Declaration)

```
public static int sumN( int n )
```

Body

```
{
  int sum = 0;
  for( int i=1 ; i <=n ; ++i )
    sum += n;
  return sum;
}
```

Method

- Header
  - Access modifier
    - public
    - private
    - Protected
  - `static`
  - Return type
  - Name
  - Parameter list
- Body

Header (Declaration)

```
public static int sumN( int n )
```

Body

```
{
  int sum = 0;
  for( int i=1 ; i <=n ; ++i )
    sum += n;
  return sum;
}
```

Method

- Header
  - Access modifier
  - Return type
    - Class
    - Primitive type
    - void
  - Name
  - Parameter list
- Body

Header (Declaration)

```
public static int sumN( int n )
```

Body

```
{
   int sum = 0;
   for( int i=1 ; i <=n ; ++i )
     sum += n;
   return sum;
}
```

Method

- Header
  - Access modifier
  - Return type
  - Name
    - Rules and conventions
    - Descriptive
  - Parameter list
- Body

Header (Declaration)

```
public static int sumN( int n )
```

Body

```
{
   int sum = 0;
   for( int i=1 ; i <=n ; ++i )
     sum += n;
   return sum;
}
```

Method

- Header
  - Access modifier
  - Return type
  - Name
  - Parameter list
    - A list of *<type, variable_name>* pairs
    - Multiple parameters separated by comma
- Body

Header (Declaration)

```
public static int sumN( int n )
```

Body

```
{
   int sum = 0;
   for( int i=1 ; i <=n ; ++i )
     sum += n;
   return sum;
}
```

Method

- Header
  - Access modifier
  - Return type
  - Name
  - Parameter list
- Body
  - Enclosed in a pair of braces
  - Parameters can be used as if they are variables declared in the method body
  - Return statement

Header (Declaration)

```
public static int sumN( int n )
```

Body

```
{
   int sum = 0;
   for( int i=1 ; i <=n ; ++i )
     sum += n;
   return sum;
}
```

Method Call

- Use/call/invoke a method
  - Specify the method name
  - Initialize parameters – assign values to parameters
    - Type matching and *coercion*

```
int sum100 = sumN( 100 );
int sum150 = sumN( 150 );
```

Calling Methods in a Different Class

- Classname.method_name()
  - If that method is static
  - E.g. Math.sqrt(3.0)
- Objectname.method_name()
  - If that method is not static

Some Math Methods

- sqrt, pow
- min, max
- random
- floor, ceiling, round
- abs
- sin, cos, tan
- asin, acos, atan
- exp, log

http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Math.html

More Method Examples

- int max(int a, int b)
- boolean xor(boolean x, boolean y)
- void usage()
- void printBar(int counter)

Recursion

- A method calls itself

```
void print( int n )
{
   if( n <= 0 ) System.out.println();
   else
   {
      System.out.print("a");
      print(n-1);
   }
}
```

Ending Condition

- When the recursion should stop
- To avoid infinite recursion, make sure the ending condition
  - Exists
  - Reachable
  - Comes before the recursive call

Example: Fibonacci Series

- 0, 1, 1, 2, 3, 5, 8, 13, 21, …
- Definition
  - fibonacci(0) = 0
  - fibonacci(1) = 1
  - fibonacci(n) = fibonacci(n-1)+fibonacci(n-2)

Recursive Fibonacci

```
int fibonacci( int n )
{
   if( n == 0 ) return 0;
   else if( n==1 ) return 1;
   else
      return fibonacci(n-1) + fibonacci(n-2);
}
```

Non-recursive Fibonacci

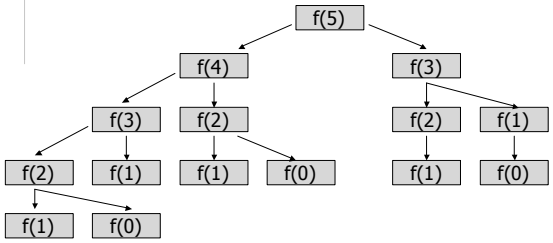
```
int fibonacci( int n )
{
   if( n == 0 || n==1 ) return n;

   int last1 = 1, last2 = 0, fibo;
   for( int i=2 ; ?? ; ++i )
   {
      fibo = last1+last2;
      ??
   }

   return fibo;
}
```

Recursion vs. Non-recursion

- Less code != more efficient



When Should We Use Recursion?

- When the homework problem says so
- When speed of code development takes precedence over code efficiency
- When the problem is naturally recursive
  - Fibonacci Series
- When the non-recursive solution is much harder
  - Hanoi tower
  - Solving maze

When Can We Use Recursion?

- A problem itself is recursively defined
  - Fibonacci → f(n) = f(n-1) + f(n-2)
  - Tree
    - A tree has a root
    - A root has two trees
- A problem of size n can be reduced to a problem of size less than n
  - Factorial: n → n-1
  - Sort: n → n-1
  - Binary search: n → n/2