JAVA Programming

Methods and parameters

Overview

- Methods
- Parameters
- Overloading
- Recursion

Defining Methods

- Main is a method
  - Use the same syntax for defining your own methods (“static” only in special cases)

```
package nl.javanc;

public class MyClass {

       public void myMethod() {
              System.out.println("running myMethod");
              }
}
```

  - Every method has a return type, even if it does not return a value

Calling Methods

- After you define a method, you can:
  - Call a method from within the same class
    - Use method's name followed by a parameter list in parentheses
  - Call a method that is in a different class
    - You must indicate to the compiler which object contains the method to call and create an instance of that object.
    - The called method must be declared with the **public** keyword
  - Use nested calls
    - Methods can call methods, which can call other methods, and so on

Using Local Variables

- Local variables
  - Created when method begins
  - Private to the method
  - Destroyed on exit
- Shared variables
  - Variables reachable by all code within the class
- Scope conflicts
  - Compiler will not warn if local variable name and shared variable name clashes

Returning Values

- Declare the method with non-void type
- Add a return statement with an expression
  - Sets the return value
  - Returns to caller
- Non-void methods must return a value

```
public int TwoPlusTwo( ) {
    int a,b;
    a = 2;
    b = 2;
    return a + b;
}
```

```
int x;
x = TwoPlusTwo( );
System.out.println(x);
```

Using Parameters

- Declaring and Calling Parameters
- Mechanisms for Passing Parameters
- Pass by Value
- Using Variable-Length Parameter Lists
- Guidelines for Passing Parameters
- Using Recursive Methods

Declaring and Calling Parameters

- Declaring parameters
  - Placed between parentheses after method name
  - Define type and name for each parameter
- Calling methods with parameters
  - Supply a value for each parameter

```
public void MethodWithParameters(int n, String y) {
 ...
}

MethodWithParameters(2, "Hello, world");
```

Pass by Value

- Default mechanism for passing parameters:
  – Parameter value is copied
  – Variable can be changed inside the method
  – Has no effect on value outside the method
  – Parameter must be of the same type or compatible type

```
public void AddOne(int x){
        x++; // Increment x
}
public void Test( ) {
        int k = 6;
        AddOne(k);
        Console.WriteLine(k);
        // Display the value 6, not 7
}
```

Using Variable-Length Parameter Lists

- Use the . . . notation
- Basically an array
- Always at the end of the parameter list

```
public double average(int... values) {
int sum=0;
for (int i = 0; i < values.length; i++) {
       sum+=values[i];
}
return sum/values.length;
}
```

```
double result=average(1,2,3,4,5,6,7,8,9);
System.out.println(result);
```

Using Overloaded Methods

- Declaring overloaded methods
- Method signatures
- Using overloaded methods

Declaring Overloaded Methods

- Methods that share a name in a class
  - Distinguished by examining signature

```
class OverloadingExample {

    public int add(int a, int b) {
        return a + b;
    }

    public int add(int a, int b, int c) {
        return a + b + c;
    }
}
```

Method Signatures

- Method signatures must be unique within a class
- Signature definition

Part of Signature Definition	No Effect on Signature
Name of method	Name of parameter
Parameter type	Return type of method
Number of Parameters	

Overloading

- Consider using overloaded methods when:
  - You have similar methods that require different parameters
  - You want to add new functionality to existing code

Recursion

- A method can call itself
  - Directly
  - Indirectly
- Useful for solving certain problems
- Do not overuse because:
  - Hard to debug
  - Hard to maintain

Lab 5: Creating and Using Methods