CS 241 — Introduction to Problem Solving and Programming

Fundamentals of Programming

More methods

Feb 7, 2005

Overview

- Review and non-trivial example
- Documenting methods
- Methods that return nothing
- Overloading
- Pitfalls

Methods and scope

```
public static void main(String[] args) {
    int x = 1;
    int y = 2;
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    int z = crazy(x, y);
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    System.out.println("z: " + z);
    z = crazy(y, x);
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    System.out.println("z: " + z);
}
static int crazy(int x, int y) {
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    x++; y++;
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    return x + y;
}
```

```
x: 1
y: 2
x: 1
y: 2
x: 2
y: 3
x: 1
y: 2
z: 5
x: 2
y: 1
x: 3
y: 2
x: 1
y: 2
z: 5
```

The scope of a variable declared in a method is only that method.

The scope of parameters is the method for which they are defined.

Methods and scope

```
public static void main(String[] args) {
    int x = 1;
    int y = 2;
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    int z = crazy(x, y);
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    System.out.println("z: " + z);
    z = crazy(y, x);
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    System.out.println("z: " + z);
}
static int crazy(int x, int y) {
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    x++; y++;
    System.out.println("x: " + x);
    System.out.println("y: " + y);
    return x + y;
}
```

```
x: 1
y: 2
x: 1
y: 2
x: 2
y: 3
x: 1
y: 2
z: 5
x: 2
y: 1
x: 3
y: 2
x: 1
y: 2
z: 5
```

Arguments are passed by value.

Example

An algorithm for adding two fractions, with result in simplest form.

- Input the first numerator (n1).
- Input the first denominator (d1).
- Input the second numerator (n2).
- input the second denominator (d2).

Example

An algorithm for adding two fractions, with result in simplest form.

- Input the first numerator (n1).
- Input the first denominator (d1).
- Input the second numerator (n2).
- input the second denominator (d2).
- Find the numerator of the sum by n1 $\times$ d2 + n2 $\times$ d1.
- Find the denominator of the sum by d1 $\times$ d2.

Example

An algorithm for adding two fractions, with result in simplest form.

- Input the first numerator (n1).
- Input the first denominator (d1).
- Input the second numerator (n2).
- input the second denominator (d2).
- Find the numerator of the sum by n1 $\times$ d2 + n2 $\times$ d1.
- Find the denominator of the sum by d1 $\times$ d2.
- Find the greatest common factor of the numerator and the denominator.

Example

An algorithm for adding two fractions, with result in simplest form.

- Input the first numerator (n1).
- Input the first denominator (d1).
- Input the second numerator (n2).
- input the second denominator (d2).
- Find the numerator of the sum by n1 $\times$ d2 + n2 $\times$ d1.
- Find the denominator of the sum by d1 $\times$ d2.
- Find the greatest common factor of the numerator and the denominator.
- Divide the numerator by the gcd.
- Divide the denominator by the gcd.
- Print the result

Example

But how do we find the gcd?.

- Input the first numerator (n1).
- Input the first denominator (d1).
- Input the second numerator (n2).
- input the second denominator (d2).
- Find the numerator of the sum by n1 $\times$ d2 + n2 $\times$ d1.
- Find the denominator of the sum by d1 $\times$ d2.
- Find the greatest common factor of the numerator and the denominator.
- Divide the numerator by the gcd.
- Divide the denominator by the gcd.
- Print the result

Adding fractions

```
int numerator1 = DocsIO.readint("Please enter the first numerator--> ");
int denominator1 = DocsIO.readint("Please enter the first denominator--> ");
int numerator2 = DocsIO.readint("Please enter the second numerator-->");
int denominator2 = DocsIO.readint("Please enter the second denominator--> ");

int numeratorSum =
    (numerator1 * denominator2) + (numerator2 * denominator1);
int denominatorSum = denominator1 * denominator2;
int factor = gcd(numeratorSum, denominatorSum);
numeratorSum /= factor;
denominatorSum /= factor;

System.out.println("Sum: " + numeratorSum + " / " + denominatorSum);
```

The Euclidean algorithm

- Given two integers, a and b, with a $>$ b:
- While b is not zero
  - Let r $=$ a mod b.
  - Set a $=$ b
  - Set b $=$ r
- Return a as the GCD.

The Euclidean algorithm

```
static int gcd(int a, int b) {




}
```

The Euclidean algorithm

```
static int gcd(int a, int b) {
    if (a < b) {
        int temp = a;
        a = b;
        b = temp;
    }









}
```

The Euclidean algorithm

```
static int gcd(int a, int b) {
    if (a < b) {
        int temp = a;
        a = b;
        b = temp;
    }
    while (b != 0) {



    }

}
```

The Euclidean algorithm

```
static int gcd(int a, int b) {
    if (a < b) {
        int temp = a;
        a = b;
        b = temp;
    }
    while (b != 0) {
        int temp = a % b;
        a = b;
        b = temp;
    }

}
```

The Euclidean algorithm

```
static int gcd(int a, int b) {
    if (a < b) {
        int temp = a;
        a = b;
        b = temp;
    }
    while (b != 0) {
        int temp = a % b;
        a = b;
        b = temp;
    }
    return a;
}
```

The Euclidean algorithm

```
ar1121: {81} java FractionAdder
Please enter the first numerator--> 3
Please enter the first denominator--> 7
Please enter the second numerator-->2
Please enter the second denominator--> 8
Sum: 19 / 28
ar1121: {82} java FractionAdder
Please enter the first numerator--> 1
Please enter the first denominator--> 2
Please enter the second numerator-->1
Please enter the second denominator--> 6
Sum: 2 / 3
```

Documenting methods

Begin each method with a block comment.

```
/**
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */
static int gcd(int a, int b) {
    if (a < b) {
        . . .
```

Documenting methods

The first sentence: a brief description in the imperative.

```
/**
 * Find the greatest common divisor of two integers.
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */
static int gcd(int a, int b) {
    if (a < b) {
        . . .
```

Documenting methods

Describe the algorithm–succinctly but completely

```
/**
 * Find the greatest common divisor of two integers.
 * This uses the Euclidean algorithm. Maintain a large number
 * and a small number. Repeatedly replace the small number
 * with the large mod the small and replace the large number
 * with the previous small number, until the small number is zero.
 * The resulting large number is the gcd.
 *
 *
 *
 */
static int gcd(int a, int b) {
    if (a < b) {
        . . .
```

Documenting methods

Use the "@param" tag for parameters like when declaring variables.

```
/**
 * Find the greatest common divisor of two integers.
 * This uses the Euclidean algorithm. Maintain a large number
 * and a small number. Repeatedly replace the small number
 * with the large mod the small and replace the large number
 * with the previous small number, until the small number is zero.
 * The resulting large number is the gcd.
 * @param a The first of the two integers for which to find the gcd.
 * @param b The second of the two integers for which to find the gcd.
 *
 */
static int gcd(int a, int b) {
    if (a < b) {
        . . .
```

Documenting methods

Describe what is returned with a "@return" tag.

```
/**
 * Find the greatest common divisor of two integers.
 * This uses the Euclidean algorithm. Maintain a large number
 * and a small number. Repeatedly replace the small number
 * with the large mod the small and replace the large number
 * with the previous small number, until the small number is zero.
 * The resulting large number is the gcd.
 * @param a The first of the two integers for which to find the gcd.
 * @param b The second of the two integers for which to find the gcd.
 * @return The integer greatest common divisor of the two parameters.
 */
static int gcd(int a, int b) {
    if (a < b) {
        . . .
```

Documenting methods

This is a good example of a confusing piece of code that could use some documentation.

```
     * The resulting large number is the gcd.
     * @param a The first of the two integers for which to find the gcd.
     * @param b The second of the two integers for which to find the gcd.
     * @return The greatest common divisor of the two parameters.
     */
    static int gcd(int a, int b) {
        // a needs to be bigger than b.
        // If it's not, switch them.
        if (a < b) {
            int temp = a;  // Place holder while we swap a and b
            a = b;
            b = temp;
        }
```

Documenting methods

Alternately, state your assumptions.

```
 * The resulting large number is the gcd.
 * @param a The first of the two integers for which to find the gcd.
 * @param b The second of the two integers for which to find the gcd.
 * PRECONDITION: a is greater than b
 * @return The greatest common divisor of the two parameters.
 */
static int gcd(int a, int b) {
    while (b != 0) {
        . . .
```

Documenting methods

Summary of the rules. Method documentation should contain:

- A one-sentence description of the functionality of the method in the imperative.
- Except for trivial methods, a few sentences describing the algorithm.
- A param tag and description for each parameter.
- A return tag with a description of what's returned.

Testing methods

Methods make testing more complicated.

```
ar1121: {107} java FractionAdder
Please enter the first numerator--> 1
Please enter the first denominator--> 2
Please enter the second numerator-->1
Please enter the second denominator--> 6
Sum: 1 / 1
```

Was it the method or the rest of the program??

Testing methods

Think of methods as components. Test them separately.

```
public class FractionAdder {
   public static void main(String[] args) {
     int numerator1 = DocsIO.readint("Please enter the first numerator--> ");
     int denominator1 = DocsIO.readint("Please enter the first denominator--> ");
     System.out.println("GCD: " + gcd(numerator1, denominator1));
     /*
     int numerator2 = DocsIO.readint("Please enter the second numerator-->");
         . . .
```

Testing methods

```
ar1121: {110} java FractionAdder
Please enter the first numerator--> 16
Please enter the first denominator--> 4
GCD: 4
ar1121: {111} java FractionAdder
Please enter the first numerator--> 12
Please enter the first denominator--> 9
GCD: 9
ar1121: {112} java FractionAdder
Please enter the first numerator--> 12
Please enter the first denominator--> 11
GCD: 11
```

Testing methods

```
static int gcd(int a, int b) {
    if (a < b) {
        int temp = a;
        a = b;
        b = temp;
   }
    while (b != 0) {
        int temp = a / b;
        a = b;
        b = temp;
    }
    return a;
}
```

Void methods

What if a method doesn't have anything to return; for example, if it merely displays some text?

```
static void printMenu(String userName) {
    System.out.println(userName + ", please choose from: ");
    System.out.println("1. Diameter of circle");
    System.out.println("2. Circumference of circle");
    System.out.println("3. Area of circle");
    System.out.println("4. Volume of sphere");
    System.out.println("5. Surface area of sphere");
    System.out.println("6. Quit");
}
```

Void methods

`void` is like a type that says "no value." Notice that the method has no `return` statement.

```
static void printMenu(String userName) {
    System.out.println(userName + ", please choose from: ");
    System.out.println("1. Diameter of circle");
    System.out.println("2. Circumference of circle");
    System.out.println("3. Area of circle");
    System.out.println("4. Volume of sphere");
    System.out.println("5. Surface area of sphere");
    System.out.println("6. Quit");
}
```

Void methods

`return`, by itself, with no expression, can be used to signal the method should stop.

```
static void printMenu(String userName) {
    if (userName.equals("Bill Gates")) {
        System.out.println("No menu for you!");
        return;
    }
    System.out.println(userName + ", please choose from: ");
    System.out.println("1. Diameter of circle");
    System.out.println("2. Circumference of circle");
    System.out.println("3. Area of circle");
    System.out.println("4. Volume of sphere");
    System.out.println("5. Surface area of sphere");
    System.out.println("6. Quit");
}
```

Void methods

You could use `return` like `break`.

```
static void sayAloha(int n) {
    for (;;) {
        System.out.println("Aloha");
        if (n == 0) return;
        n--;
    }
}
```

Rolling back the curtain

Now we can make more sense of the beginning of our programs. This defines a main method.

```
public static void main(String[] args) {
  . . .
```

Rolling back the curtain

This is the same `static` we use in other methods. We will see some non-static later. . .

```
public static void main(String[] args) {
  . . .
```

Rolling back the curtain

The main method does not return any value, since there is nothing to return a value to. Its return type is `void`.

```
public static void main(String[] args) {
  . . .
```

Rolling back the curtain

A parameter. We'll later find out where it comes from, how to use it, and what `[]` means.

```
public static void main(String[] args) {
  . . .
```

Method signatures

A method's full name is more than just the identifier. A method's signature consists of

- its name (identifier)
- the number of parameters
- their type
- and their order.

```
static double cylinderVolume(double radius, double height) {
    return 3.14159 * radius * radius * height;
}
```

Signature:

```
cylinderVolume(double, double)
```

Overloading

You can reuse the name of a method if the signature is different.

This is called method overloading.

```
static int area(int base, int height) {
    return (base * height) / 2;
}
static double area(double base, double height) {
    return (base * height) / 2;
}
static double area(double equilateralSide) {
    return equilateralSide * equilateralSide * .433;
}
```

Overloading

```
public class Triangle {
    public static void main(String[] args) {
        System.out.println(area(5, 10));
        System.out.println(area(5.0, 10.0));
        System.out.println(area(4));
        System.out.println(area(5, 10.0));
    }
    static int area(int base, int height) {
        return (base * height) / 2;
    }
    static double area(double base, double height) {
        return (base * height) / 2;
    }
    static double area(double equilateralSide) {
        return equilateralSide * equilateralSide * .433;
    }
}
```

Overloading

```
public class Triangle {
    public static void main(String[] args) {
        System.out.println(area(5, 10));
        System.out.println(area(5.0, 10.0));
        System.out.println(area(4));
        System.out.println(area(5, 10.0));
    }
    static int area(int base, int height) {
        return (base * height) / 2;
    }
    static double area(double base, double height) {
        return (base * height) / 2;
    }
    static double area(double equilateralSide) {
        return equilateralSide * equilateralSide * .433;
    }
}
```

```
25
```

Overloading

```
public class Triangle {
    public static void main(String[] args) {
        System.out.println(area(5, 10));
        System.out.println(area(5.0, 10.0));
        System.out.println(area(4));
        System.out.println(area(5, 10.0));
    }
    static int area(int base, int height) {
        return (base * height) / 2;
    }
    static double area(double base, double height) {
        return (base * height) / 2;
    }
    static double area(double equilateralSide) {
        return equilateralSide * equilateralSide * .433;
    }
}
```

```
25
25.0
```

Overloading

```
public class Triangle {
    public static void main(String[] args) {
        System.out.println(area(5, 10));
        System.out.println(area(5.0, 10.0));
        System.out.println(area(4));
        System.out.println(area(5, 10.0));
    }
    static int area(int base, int height) {
        return (base * height) / 2;
    }
    static double area(double base, double height) {
        return (base * height) / 2;
    }
    static double area(double equilateralSide) {
        return equilateralSide * equilateralSide * .433;
    }
}
```

```
25
25.0
6.928
```

Overloading

```
public class Triangle {
    public static void main(String[] args) {
        System.out.println(area(5, 10));
        System.out.println(area(5.0, 10.0));
        System.out.println(area(4));
        System.out.println(area(5, 10.0));
    }
    static int area(int base, int height) {
        return (base * height) / 2;
    }
    static double area(double base, double height) {
        return (base * height) / 2;
    }
    static double area(double equilateralSide) {
        return equilateralSide * equilateralSide * .433;
    }
}
```

```
25
25.0
6.928
25.0
```

Pitfalls

1. Returning the wrong type.

```
    static int area(int base, int height) {
        return "area is: " + base + " * " + height;
    }

...

Triangle.java:9: incompatible types
found   : java.lang.String
required: int
        return "area is: " + base + " * " + height;
                                                ^
1 error
```

Pitfalls

2. Going against explicit cast in the method.

```
    static int area(int base, int height) {
        return ((double) base * height) / 2;
    }

...

Triangle.java:9: possible loss of precision
found   : double
required: int
        return ((double) base * height) / 2;
                                        ^
1 error
```

Pitfalls

3. Going against explicit casts on the caller side.

```
        System.out.println(area(4.0));
      ...
    static double area(int equilateralSide) {
        return equilateralSide * equilateralSide * .433;
    }

...

Triangle.java:5: cannot resolve symbol
symbol  : method area (double)
location: class Triangle
        System.out.println(area(4.0));
                           ^
1 error
```

Pitfalls

4. Returning a value from a `void` method.

```
    static void printArea(int base, int height) {
        return (base * height) / 2;
    }

...

Triangle.java:18: cannot return a value from method whose result type is void
        return (base * height) / 2;
                               ^
1 error
```

Pitfalls

5. Trying to use a value from a `void` method.

```
        System.out.println(area(5, 10));
        ...
    static void area(int base, int height) {
        System.out.println((base * height) / 2);
    }

...

Triangle.java:3: 'void' type not allowed here
        System.out.println(area(5, 10));
                                ^
1 error
```

Pitfalls

6. Double declaration.

```
    static int area(int base, int height) {
        return (base * height) / 2;
    }
    static int area(int side1, int side2) {
        return side1 * side2;
    }

...

Triangle.java:11: area(int,int) is already defined in Triangle
    static int area(int side1, int side2) {
               ^
1 error
```

Pitfalls

7. Ambiguous use of overloading.

```
        System.out.println(area(5, 10));
       ...
    static double area(int base, double height) {
        return (base * height) / 2;
    }
    static double area(double base, int height) {
        return (base * height) / 2;
    }

...

Triangle.java:3: reference to area is ambiguous, both method area(int,double) in Tr
        System.out.println(area(5, 10));
                           ^
1 error
```

Pitfalls

8. Overloading on the return type.

```
    static int area(int base, int height) {
        return (base * height) / 2;
    }
    static double area(int base, int height) {
        return ((double) base * height) / 2;
    }

...

Triangle.java:11: area(int,int) is already defined in Triangle
    static double area(int base, int height) {
                  ^
1 error
```

Summary

Know the following concepts.

- Scope (in the context of methods)
- How to document a method
- Void methods
- Method signatures
- Method overloading