WIT COMP1000

Methods

Methods

- Programs can be logically broken down into a set of tasks
- Example from horoscope assignment:
  - » Get input (month, day) from user
  - » Determine astrological sign based on inputs and output horoscope
- Individual tasks can be separated out from the main program into *methods*
- A method is simply a mini-program that completes a specific task

Predefined Methods

- Java includes many predefined methods for common programming tasks
- Example of using the predefined square root method, `Math.sqrt()`:

```
double root, input_value;
System.out.print("Enter a number: ");
input_value = input.nextDouble();

root = Math.sqrt(input_value);

System.out.println("The square root is " + root);
```

returned value

method call

argument

Generic Form

```
RETURN_TYPE METHOD(PARAMETER_1, PARAMETER_2, …, PARAMETER_N)
```

- A method can have any number of parameters
  - » Each parameter has a specified type (`int`, `double`, `String`, etc)
- A method has either zero or one return value(s)
  - » The return value is commonly the result of the method
  - » If it has a return value, the value also has a specified type
  - » The return value can be assigned to a variable of the same type (as in the previous example)
  - » Alternatively, the method call can be placed directly in another Java expression and the return value will be used in place of the method call

A Few Java Methods

- Square root: `double Math.sqrt(double a)`
- Power: `double Math.pow(double base, double exp)`
- Absolute value: `double Math.abs(double a)`
- Natural log: `double Math.log(double a)`
- Log base 10: `double Math.log10(double a)`

More Examples

```
double number, cube, log2;
System.out.print("Enter a number: ");
number = input.nextDouble();

System.out.println(number + "'s square root is " + Math.sqrt(number));

cube = Math.pow(number, 3.0);
System.out.println(number + "^3.0=" + cube);

log2 = Math.log(number) / Math.log(2.0);
System.out.println("log2(" + number + ")=" + log2);
```

Terminology Notes

- We use *parameters* to refer to the list of variables a method requires (in parentheses)
  - » They are place holders for values that will be used when the method is called
- We use *arguments* (a.k.a. actual parameters) to refer to the specific values and/or variables that are passed in when you invoke the method
- Also note that other languages refer to methods as *functions* or *procedures*

Exercise

- Write a program that prints out the value of 2^x for x=1,2,3,…,32
- Use the `Math.pow()` method and a **`while`** loop

Answer

```
double x = 1;
double pow2;

while (x <= 32) {
   pow2 = Math.pow(2, x);
   System.out.printf("2^%.0f=%.0f%n", x, pow2);
   x++;
}
```

`System.out.printf()` is just another method! It has a `String` parameter followed by one argument for each % place holder

Programmer-Defined Methods

- Java allows you to define your own methods to meet the needs of your specific program
- To define your own method, you need to write the method *signature* and the method *body*
- The signature includes the method name, parameter list, and return type
- The body is the set of Java statements that will be executed when the method is invoked

No Parameters, No Return Value

```
public class ClassExamples {

    public static void main(String[] args) {

        sayHello();

    public static void sayHello() {
        System.out.println("hello!");
```

method call to execute the method

empty ( ) means no parameters

void means no return value

For now, always put `public static` in front

statements to execute when the method is called

Methods

- When a program executes a method, it temporarily stops where it is in `main()`, goes to the lines of code in the method, and executes those lines like normal
- Then, when you get to the end of the method (or a **`return`** statement) it goes back to `main()` and resumes executing after the method call

No Parameters, One Return Value

```
public class ClassExamples {

    public static void main(String[] args) {
        double pi;
        pi = getPiApprox();
        System.out.println("Pi ~ " + pi);
    }

    public static double getPiApprox() {
        return 3.14159;
    }

}
```

`double` means the method returns a double

Have to return an double value

Return Values

- Methods have zero or one return value(s)
- If a method has a return value, it is of a specific type (`int`, `double`, `String`, …)
  - » Type is defined as part of the method signature
- Use the `return` statement to return a value of the specified type
  - » Can be a constant, variable, expression, method, or anything that is evaluated to the required type

Another Example

```
public class ClassExamples {

    public static void main(String[] args) {
        String s;
        s = turnThatFace();
        System.out.println(s);
    }

    public static String turnThatFace() {
        String oldFace = ":(";
        String newFace = oldFace.replace('(', ')');
        return newFace;
    }

}
```

Return a Method Call

```
public class ClassExamples {

    public static void main(String[] args) {
        String s;
        s = turnThatFace();
        System.out.println(s);
    }

    public static String turnThatFace() {
        String oldFace = ":(";
        return oldFace.replace('(', ')');
    }

}
```

Exercise

- Write a method named `leibniz()` that approximates π by the Leibniz formula: (4/1) – (4/3) + (4/5) – (4/7) …
- First just write a method that returns the result of these first four elements as a double
- For a stretch goal, write a loop in the method to compute to any iteration!

Answer

```
public class ClassExamples {

    public static void main(String[] args) {
        double pi;
        pi = leibniz();
        System.out.println("Pi ~ " + pi);
    }

    public static double leibniz() {
        return (4./1.) - (4./3.) + (4./5.) - (4./7.);
    }

}
```

Stretch!

```
public class ClassExamples {

    public static void main(String[] args) {
        double pi;
        pi = leibniz();
        System.out.println("Pi ~ " + pi);
    }

    public static double leibniz() {
        int den;
        double sum = 0.0;
        double sign = 1.0;
        for (den = 1; den <= 7; den = den + 2) {
            sum = sum + (sign * (4.0 / den));
            sign = sign * -1.0;
        }
        return sum;
    }

}
```

Methods with Parameters

- Methods can take any number of parameters
- Each parameter has a predefined data type (`int`, `double`, `String`, …), defined as part of the method signature
- When called, the *value* of the argument is passed to the method
  - In other words, the values of the arguments are plugged in to the method, just like in a normal expression

Example with Two Parameters

```
import java.util.Scanner;

public class ClassExamples {

    public static void mai
        Scanner input = new
        double input1, inpu
        System.out.print("E
        input1 = input.nextDoub
        u
        a
        r

    public static double doCalculation(double a, double b) {
        return (a*a + b*b);
    }
}
```

the current value of `input1` is passed to the method as `a`

the current value of `input2` is passed to the method as `b`

`double` means the method returns a `double` value

`double` means the first parameter is a `double` value

`double` means the second parameter is a `double` value

Parameters

- Each time a method is called, you can pass it different arguments
  - The arguments are plugged in separately each time, so you can call a method many times with different arguments to get a different return value

```
public class ClassExamples {
    public static void main(String[] args) {
        double result1, result2;
        result1 = doCalculation(3, 4);
        System.out.printf("result1 is %.3f%n", result1);
        result2 = doCalculation(2, 8);
        System.out.printf("result2 is %.3f%n", result2);
    }
    public static double doCalculation(double a, double b) {
        return (a*a + b*b);
    }
}
```

Important Note

- Arguments are passed in by *value* to the method
- If you have a variable in `main()` that is used as an argument to a method then the value of that variable is used as the parameter in the method
- Any changes made to the value in the method do *not* affect the original variable in `main()`

Pass by Value Example

```
public class ClassExamples {

    public static void main(String[] args) {
        double x = 10, y = 20;
        double result;
        System.out.println("before doCalculation(), x=" + x);
        result = doCalculation(x, y);
        System.out.println("after doCalculation(), x=" + x);
        System.out.println("doCalcuation() result=" + result);

    }

    public static double doCalculation(double a, double b) {
        System.out.println("at start of doCalculation(), a=" + a);
        a = a - 5;
        System.out.println("at end of doCalculation(), a=" + a);
        return (a*a + b*b);
    }

}
```

Getting Input in a Method

- If you need to get input from the user within a method, it is best practice to pass an existing Scanner variable as a parameter to the method

```
public static int getInt(Scanner s) {
   return s.nextInt();
}
```

Exercise

- Write a method getPositiveNumber that takes a Scanner as a parameter and loops until the user types a positive number, which is then returned

Answer

```
import java.util.Scanner;

public class ClassExamples {

     public static void main(String[] args) {
          Scanner input = new Scanner(System.in);
          System.out.println(getPositiveNumber(input));
     }

     public static double getPositiveNumber(Scanner s) {
          double input_value;

          do {
               System.out.print("Enter a positive number: ");
               input_value = s.nextDouble();
          } while (input_value <= 0);

          return input_value;
     }

}
```

Multiple `return` Statements

- Methods (including the `main()` method) can have multiple **`return`** statements in them
- When a **`return`** statement is executed, the method stops and the stated value is returned to the caller immediately
  - » No more of the method is executed (unless it is called again, in which case it starts over)
  - » Note that in `main()`, **`return`** causes the entire program to end (`main()` is a method, too!)

Multiple `return` Statements Example

Methods that return a `boolean` are often used in boolean expressions

Methods can have multiple `return` statements

```
import java.util.Scanner;

public class ClassExamples {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int input_value;
        System.out.print("Enter an integer: ");
        input_value = input.nextInt();
        if(isEven(input_value)) {
            System.out.println(input_value + " is even!");
        } else {
            System.out.println(input_value + " is odd!");
        }
    }

    public static boolean isEven(int number) {
        if (number % 2 == 0) {
            return true;
        } else {
            return false;
        }
    }
}
```

Exercise

- Write a program that calculates the area of a rectangle given the two side lengths which are provided by the user. You must write a method that is passed the two side lengths and returns the area.

Answer

```
import java.util.Scanner;

public class ClassExamples {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        double l, w;
        double a;
        System.out.print("Enter rectangle length: ");
        l = input.nextDouble();
        System.out.print("Enter rectangle width: ");
        w = input.nextDouble();
        a = rectangleArea(l, w);
        System.out.printf("The area is %.3f%n", a);
    }

    public static double rectangleArea(double length, double width) {
        return (length*width);
    }

}
```

Take Home Points

- Methods are mini-programs that are generally used to contain all of the code to complete some particular task
- Methods have either zero or one return value(s)
  - » If it has one, the value is of a specified type
- Methods have zero or more parameters
  - » Each parameter (if any) has a specified type
  - » When called, the current values of the arguments are plugged in and passed as values to the method
- Methods can have multiple `return` statements