Modular Programming

CSX3002/ITX2001 Object-Oriented Concepts and Programming
CS4402 Selected Topic in Object-Oriented Concepts
IT2371 Object-Oriented Programming

Kwankamol Nongpong, Ph.D.
Assumption University

Modular Programming (1)

- Divide and conquer
  - Break down a big problem into smaller problems
  - We tackle one smaller problem at a time, further applying the D&C if not small enough.
  - Each smaller piece is more manageable than original one
  - Solution to each smaller problem is often made into a module (function).
  - Each function can accept parameters and return
  - Parameters and return values are medium of communication between the function and the caller.

Modular Programming (2)

- Boss to worker analogy
  - A boss (the calling function or caller) asks the secretary (the called function) to perform a task giving her data to process and asks her to return (i.e., report) the results when the task is done.

Methods

- Modules: methods and classes
- Programs use new and "prepackaged" modules
  - New: programmer-defined functions, classes
  - Prepackaged: from the standard library
- Functions enable modular programming and support the D&C design technique
- Extra Benefits:
  - Code-reuse
  - Information Hiding

Methods

- Useful functions are written once and can be used many times.
  - in the same program
  - in different programs
- Users (callers) of a function do not need to know how the function does its job. The caller is only interested in what it does.
  - communicating through parameters and return value

Built-in (Pre-packaged) Libraries: Standard Java Libraries

- Java comes with large number of built-in functions organized into set of libraries.
- You can use them without knowing how they are written as long as you understand what they do and how to communicate with them.
  - parameters and return value
- For example, there are useful mathematical functions in the math library.
- To use these functions, you have to import java.lang.Math

Built-in Libraries

- In the Math class,

```
double sqrt(double x);
```

- Example

```
System.out.println(Math.sqrt(900.0));
```

  - The preceding statement would print 30

Math Library Functions

- Function arguments can be
  - Constants
    - `Math.sqrt( 4 );`
  - Variables
    - `Math.sqrt( x );`
  - Expressions
    - `Math.sqrt(Math.sqrt( x ) ) ;`
    - `Math.sqrt( 3 - 6 * x );`

Defining a Method

Method Declaration

```
<return type> methodName(<list of parameters>)
{

   // body of the method

}
```

Example: Method Declaration

```
public double calculateAnswer(double wingSpan,
                              int numberOfEngines,
                              double length,
                              double grossTons) {
    //do the calculation here

}
```

Components of Method Declaration

- Modifiers
  - such as public, private, and others you will learn about later.
- Return type
  - the data type of the value returned by the method, or void if the method does not return a value.
- Method name
  - the rules for field names apply to method names as well, but the convention is a little different.
- Parameter list in parenthesis
  - a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, ().
  - If there are no parameters, you must use empty parentheses.
- Exception list—to be discussed later
- Method body, enclosed between braces
  - the method's code, including the declaration of local variables, goes here.

Signature of a Method

- Method Declaration

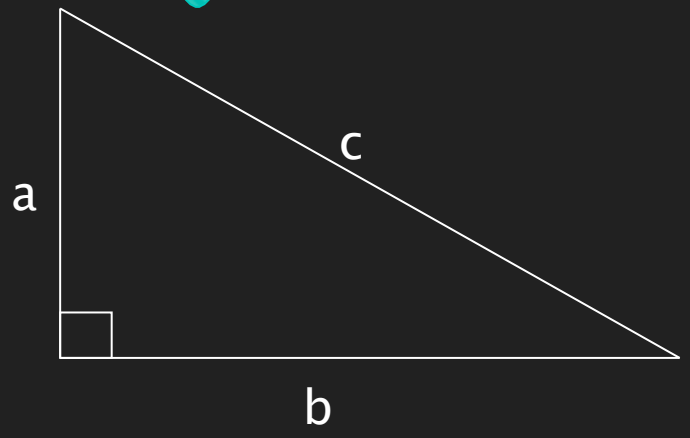
```
public double calculateAnswer(double wingSpan,
                              int numberOfEngines,
                              double length,
                              double grossTons) {
   //do the calculation here

}
```

- Signature

```
calculateAnswer(double, int, double, double)
```

Sample using built-in function



$$hypotenuse(c) = \sqrt{a^2 + b^2}$$

```
double hypothenuse(int a, int b) {
    return Math.sqrt(a*a, b*b);
}
```

Math.pow(a, 2)

Math.pow(a, 2)

In-Class Exercise 1

- Create a method determine if a given integer is positive or not.
- Design Issues
  - What is a good method name?
  - Should the method take any parameters?
    - If so, what are the types of those parameters?
  - What would be a proper return type?
- Then, write a program that repeatedly takes an integer and prints out if the input value is a positive or negative number.

In-Class Exercise 2

- Create a method that takes two numbers and returns the largest value.

In-Class Exercise 3a

- Create a method that takes three numbers and returns the largest value.

In-Class Exercise 3b

- Create a method that takes three numbers and returns the largest value.
- Use the method `Math.max()`

In-Class Exercise 4

- Create a method called rollDice() which returns a number from 1 to 6.
- Hint: You will need to randomize a number.

Parameter Passing

Parameters

- Formal Parameters
- Actual Parameters

```
public int multiply(int x, int y) {
   return x * y;
}
// This is where the method multiply gets called
int length = 10;
int width = 5;
int area = multiply(length, width);
```

Parameter Passing

Pass by Value

- A copy of the *value* of each actual parameter is passed to the method.
- You can change that copy inside the method, but this will have no effect on the actual parameter.

Pass by Reference

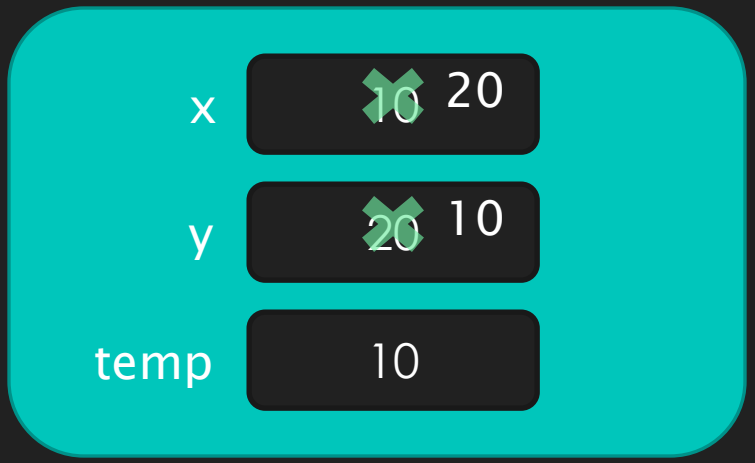
and others

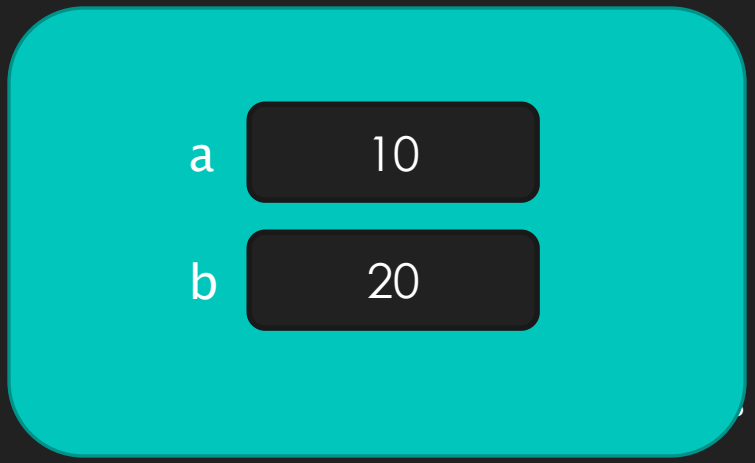
Example in C++: Swap Funct

Changes on x and y do not have effect on a and b.

```
void swap(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

```
// void swap (int, int);
int main()
{
  a = 10; b = 20;
  swap (a, b);
  cout << "a is " << a << endl;
  cout << "b is " << b << endl;
  return 0;
}
```





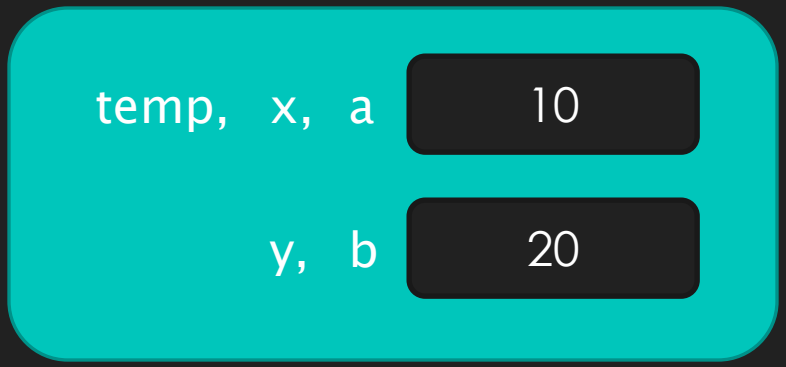
Example in C++: Swap Function

```
void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

x is an alias of a.
y is an alias of b.

Changes on x and y have effect on a and b.

```
int main()
{
  a = 10; b = 20;
  swap (a, b);
  cout << "a is " << a << endl;
  cout << "b is " << b << endl;
  return 0;
}
```



Java's Parameter Passing

- Pass by Value
  - Primitive Types
  - Object References
    - We can manipulate the object in any way, but we cannot make the reference refer to a different object.
- Pass by Reference (not really)

Passing Primitive Types in Java

```
public void swap(int a, int b) {
   int temp = a;
   int a = b;
   int b = temp;
}

// This is where the method swap gets called
int x = 10;
int y = 20;
swap(x, y);
```

What are the value of x and y after the method is called?

Passing Object References in Java (1)

```
class Student {
   String firstName;
   String lastName;
     // getters/setters
}
public static void tryObject(Student s)
{
   s.firstName = "Fred”;
   s.lastName = "Flintstone”;
}
```

Passing Object References in Java (2)

```
Student aStudent = new Student();
aStudent.setFirstName("Homer");
aStudent.setFirstName("Simpson");
tryObject(aStudent);
System.out.println(aStudent.getFirstName()
                        + "   "
                        + aStudent.getLastname());
```

What will be printed on the screen?

Does this method work?

```
public void createStudent(Student s,
                          String fName,
                          String lName)
{
    s = new Student();
    s.setFirstName(fName);
    s.setLastName(lName);
}
```

Passing String in Java

```
public static void tryString(String s)
{
    s = "a different string";
}

// This is where the method gets called.
String str = "This is a string literal.”;
tryString(str);
System.out.println("str = " + str);
```

What will be printed on the screen?