Chp 8: Methods

- Structures
  1. Basics:
     - Why Methods?
     - How to define a method
     - How to use a method -> calling
  2. Issues with methods:
     - Passing parameters and values
     - Overloading
     - Scope of variables
  3. Useful tools:
     - The Math class
     - Wrapper classes
     - Enumerated Types
  4. Designing Programs with Methods

==========

(1)
- Basics

  - Why Methods
    - Modular Programming (Procedural Programming)
    - Code more readable
    - Code more maintainable (modified by others)
    - Code reuse

  - How to define a method
    - Syntax:

```
public static <Return_Type> <Method_Name>( <Parameter_List> )
{
    <body>
    return <value>
}
```

      where
        1. "public" and "static" are reserved keywords
        2. <Return_Type> can be int, double, float, char, ..., void
             void -> nothing to return
                     (no data to be output from this method)
        3. Parameter_List
           - void or a list of declarations for variables called parameters
        4. <body>
            statement(s)
        5. return <value>
           - output a value and terminates this method
           - may appear in any place and can appear more than once in <body>
    - Examples:

```
public static double distance(double x, double y)
{
    return Math.sqrt(x * x + y * y);
}
public static void hello ( )
{
    System.out.println( "Hello" );
}
```

    - Note:
      - variables declared inside the method body are called
        local variables and are only known within the method

  - How to use a method -> calling
    - Syntax:

```
<Method_Name>( Argument_List );
```

    - Examples:

```
double dist = distance(1.0,2.0);
hello();
```

    - Note:

```
      - MUST understand the step-by-step procedure when calling a method
      - A method is executed when it is called.
        Control flow goes back to the place where the method is called after
        - return statement is executed in the method
        - or the last statement of the method is executed


==============================================

(2)
- Issues with methods:

  - Passing parameters and values
    - Note:
      Communications between a method and the calling body
      is done through parameters and the return value of a method
    - Two terms:
      - Parameters or (Formal parameters)
        - used when defining methods
      - Arguments or (Actual parameters)
        - the values used when calling the method
    - Pass by value
      - refers to passing parameter values into a called method

  - Overloading
    - Signature of a method:
      - number of parameters
      - type of each parameter
    - Overloading:
      - allow the creation of more than one methods with *same name*
        - doing similar tasks
        - but differ only in the signature

  - Scope of variables
    - Every variable has scope and duration
      - Scope:
        the section of code that it can be used
      - Duration:
        its life time in memory
    - More terminologies:
      - Local Variables
        - variables declared within a method (local to method
      - Block scope
        - the variable is visible until the end of the block
          containing its definition,
        - e.g., local variables and method's formal parameters
      - Class scope
        - the variable is visible from the beginning of a class
          definition until the end of the class definition
        - e.g., class variables with "static"
                (constants in a class)
      - Static Duration
        - the variable exists throughout program execution
      - Automatic Duration
        - the variable only exists within a block where the
          variable is defined
    - Note:
      - Java -> All variables with class scope have static duration
      - if we declare a variable in the initialization of a for statement,
        the variable will be local to the for structure


==============================================

(3)
- Useful tools:

  - The Math class
    - From java.lang package
```

```
    - provides a set of mathematical functions such as
      abs(), sqrt(), pow(), sin(), cos(), etc.
    - random()
      - Returns a random number within [0,1)
        (excluding 1.0)

  - Wrapper classes
    - wraps a primitive data type into an object like a container/box
      e.g.
      Integer intObject = new Integer (100);
    - provide methods for managing the associated primitive data types:
      e.g.,
      int value = intObject.intValue();
    - provide various useful methods:
      - e.g., from String to int
          int num = Integer.parseInt(str);
      - e.g. toBinaryString(), tohexString(), toOctalString(), etc.
    - Autoboxing: give value to a wrapper object
        Integer numObject1;
        int num1 = 888;
        numObject1 = num1;
    - Unboxing: extract the value from a wrapper object
        Integer numObject2 = new Integer(888);
        int num2;
        num2 = numObject2;

  - Enumerated Types
    - define symbolic names to represent the type of a variable
    - keyword: enum
    - three steps:
      - step 1: declare the enumerated type
          enum Day {Sun, Mon, Tue, Wed, Thu, Fri, Sat};
      - step 2: create a variable of this type
          Day today;
      - step 3: use the available values
          today = Day.Tue ;
    - Note:
      - enumeration values are represented internally by an integer
        number starting from zero, e.g.,
          Sun=0 Mon=1 Tue=2 Wed=3 Thu=4 Fri=5 Sat=6
      - In Java, an enumerated type is a class and the variables of
        an enumerated type are objects
      - ordinal() Returns integer value, e.g,
          today.ordinal() -> returns 2 for Tue
      - name() Returns the name, e.g,
          today.name() -> returns "Tue"
      - Advantage: program more readable


==========

(4)
- Designing Programs with Methods

  - Top-Down Approach
      Big Problem (or big software system)
        -> Subproblems
           -> Subsubproblems
              -> ......

  - Top-down stepwise refinement means to
    1. start with the high level description of the program
    2. decompose the program (the main() method) into
       successively smaller methods until we arrive at
       suitably sized methods
    3. then design the code for the individual method using
       stepwise refinement.
    4. at each level we are only concerned with what the
       lower level methods will do, but not how.
```

- Advantages:
  - programs are easier to write and debug
  - reduce program development time and enhance program reliability
  - code reuse