Road Map

- Method abstraction
- Divide and conquer
- Examine parameters
- Examine return types
- Argument promotion

- Reading:
  - Liang: 5.1-5.4
Benefits of Methods

- Experience has shown that the best way to develop and maintain large programs is to build it from smaller components or modules.
- In Java, one such module is the method.
- Each module is generally simpler and more manageable than the entire program.
- This concept is known as *Divide and Conquer*, also with *Abstraction* in the mix.
Divide and Conquer

• Divide and Conquer: break large programs into a series of smaller methods.
  – Helps manage complexity
  – Makes it easier to build large programs
  – Makes it easier to debug programs
Abstraction

• *Abstraction*: most of the time, you need to know what a method does, but not how it actually does it.
  – Also helps manage complexity.
  – You can use other people’s code, without understanding how it works.

• If you have code that would appear more than once, put it in a method and call that method multiple times
  – Reduces length and complexity of code
  – Reduces likelihood of errors in code
User defined Method Declarations

• General format of method declaration:

    modifiers return-value-type  method-name( parameter1, …, parameterN )
    {
        declarations and statements
    }

• Method can also return values:

    return expression;
Naming your methods

• As with variables naming methods is important
• You should give your methods names which clearly describe what the function is doing
  – helps debugging
  – helps others read your code
• Same rules as naming variables
  – E.g. public static double calculateTax( int sale )
• When you write about a method in an explanation use the parenthesis to indicate you are referencing a method (as opposed to a regular variable):
  – E.g. //call squareInteger() to calculate the square
Good programming with methods

• A method should do *one and only one* useful action
  – If you see names for your method that suggest multiple actions then it’s time to break it up into separate functions; for example,
    ```java
calculateTaxAndPrintReturnAndSaveFile();  // ugh
```
• If you do something more than once in a program, you should write a method for that action.
• If you have written a method to do something in one project, and you need to do the same action in another project, you should reuse the method.
  - In Java this is usually accomplished by using classes which we will not cover this semester.
Return Value Types

• You can only return one value from a method.

• **Returning void**
  - `void`: means “nothing”
  - `void` methods can optionally use a return statement with no value:
    ```java
    return;
    ```
  - There is no need for the optional return statement. But using one can force early exit from the method.
  - Example:
    ```java
    public static void printIntro (int n);
    ```
Parameter Data Types

• Unlike return values, you can pass as many parameters as you like.
• To pass more than one parameter, you need to separate the parameters with commas.

```java
public static int maximum (int x, int y)
{
    /*body*/
}
```
Warning

• Unlike declaring variables, you must specifically state the type for multiple variables
  – For example
    
    ```java
    takeTwoDoubles(double x, y )
    ```

    **is incorrect**
  – Instead you must write
    
    ```java
    takeTwoDoubles(double x, double y)
    ```
No parameters

• You can also have a method that accepts no parameters. In such case, you would just have an empty parameter list.

E.g.

```java
public static int rollDie ()

public static void printIntro ()
```
Argument Promotion

• Coercion of arguments
  – Forcing arguments to appropriate type to pass to method
    • e.g., `System.out.println( Math.sqrt( 4 ) );`
      – Evaluates `Math.sqrt( 4 )`
      – Then evaluates `System.out.println()`

• Promotion rules
  – Specify how to convert types without data loss
<table>
<thead>
<tr>
<th>Type</th>
<th>Valid promotions</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>None</td>
</tr>
<tr>
<td>float</td>
<td>double</td>
</tr>
<tr>
<td>long</td>
<td>float or double</td>
</tr>
<tr>
<td>int</td>
<td>long, float or double</td>
</tr>
<tr>
<td>char</td>
<td>int, long, float or double</td>
</tr>
<tr>
<td>short</td>
<td>int, long, float or double</td>
</tr>
<tr>
<td>byte</td>
<td>short, int, long, float or double</td>
</tr>
<tr>
<td>boolean</td>
<td>None (boolean values are not considered to be numbers in Java)</td>
</tr>
</tbody>
</table>

**Fig. 6.5** Allowed promotions for primitive types.
Exercise 5.24

• Implement the following methods
  – Method `convertToFahrenheit` returns the Fahrenheit equivalent of a Celsius temperature (`9.0 / 5.0 * cTemp + 32`)
  – Method `convertToCelsius` returns the Celsius equivalent of a Fahrenheit temperature (`( 5.0 / 9.0 * ( fTemp - 32 ) )`)
  – Use these methods to write a program that prints charts showing the Fahrenheit equivalent of all Celsius temperatures from 0 to 100 degrees, and the Celsius equivalents of all Fahrenheit temperatures from 32 to 212 degrees. Print the output in a neat tabular format.
Exercise 2

• Write a method that will print a line of stars. The method should take one parameter which will determine the number of stars to print. Write a main() method which calls the star printing method 5 times using random numbers from 3 - 13.
Exercise 3

• Write a method that will raise an integer to a given power. (Do not use Math.pow.)