

# Methods

Han-fen Hu

Tony Gaddis (2019) Starting Out with Java: From Control Structures through Data Structures, 4th Edition



# Outline

- ❑ Defining a Method
- ❑ Passing Arguments to a Method
- ❑ Returning a Value from a Method
- ❑ Problem Solving with Methods
- ❑ `DecimalFormat` Class

# Why Write Methods?

## □ Method

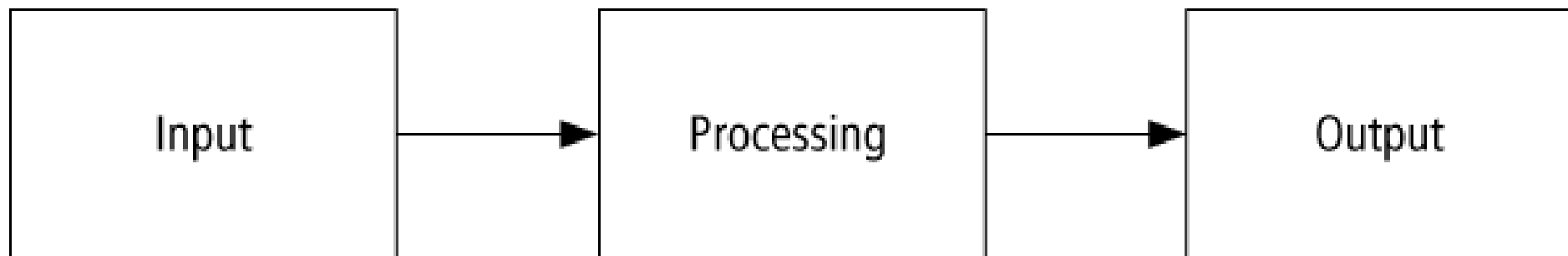
- Block of program code that performs specific task
- A program unit that performs a sub-task

## □ Why methods?

- Divide and conquer: Top-down design
- Methods simplify programs: break a problem down into small manageable pieces

# Design a Method with IPO (1)

- ❑ Determine Output
- ❑ Identify Input
- ❑ Determine process necessary to turn given Input into desired Output

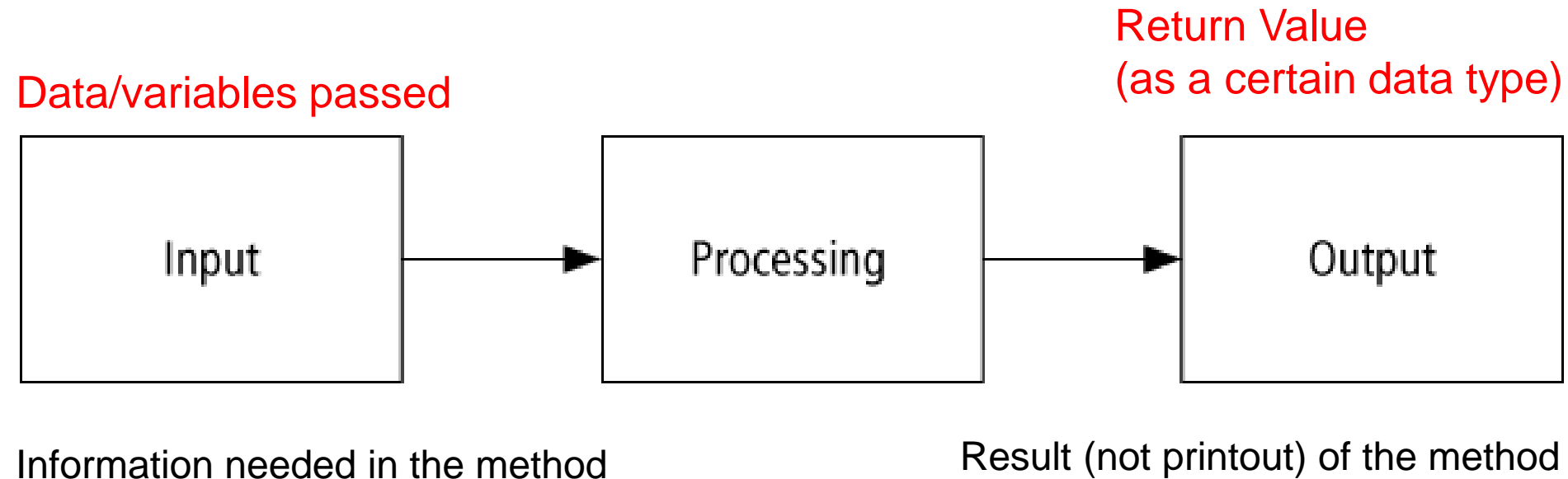


# Design a Method with IPO (2)

❑ Example: Design a method that converts hours and minutes to minutes

- Input: hours, minutes
- Output: minutes
- Process:
  - Calculate  $\text{minutes} = \text{hours} * 60 + \text{minutes}$
  - Return minutes
- Alternatives:
  - Defining 60 as a constant



# Design a Method with IPO (3)



\* Best Practice: Do not handle user interactions in methods.

# Design a Method with IPO (4)

❑ Example: Design a method that converts hours and minutes to minutes

- Input: hours, minutes  Parameters of the method.  
They both are integer values
- Output: minutes  Result of the method.  
Most likely is an integer value
- Process:
  - Calculate  $\text{minutes} = \text{hours} * 60 + \text{minutes}$
  - Return minutes
- Alternatives:
  - Defining 60 as a constant

# Design a Method with IPO (5)

Input variables, with data type

```
public static int convertToMinutes(int hr, int min) {  
    //result variable and calculation  
    int result = 60*hr + min;  
    return result;  
}
```

The data type of returned value



# Defining a Method (1)

```
public static void displayMesssage() {  
    System.out.println("Hello");  
}
```

Header

Body

# Defining a Method (2)

- ❑ To create a method, you must write a definition, which consists of a **header** and a **body**.
  - The method header, which appears at the beginning of a method definition, lists several important things about the method, including the method's name.
  - The method body is a collection of statements that are performed when the method is executed.

# Defining a Method (3)

Method  
Modifiers



Return  
Type



Method  
Name



Parentheses  
for arguments



```
public static void displayMessage () {  
    System.out.println("Hello");  
}
```

# Defining a Method (4)

## ❑ Method modifiers

- `public`—method is publicly available to code outside the class
- `static`—method belongs to an application class.

## ❑ Return type—`void` or the data type from a value-returning method

## ❑ Method name—name that is descriptive of what the method does

- Verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized

## ❑ Parentheses—contain nothing or a list of one or more variable declarations for receiving arguments.

# void Methods vs. Value-Returning Methods

- ❑ A `void` method is one that simply performs a task and then terminates.

```
System.out.println("Hi!");
```

- ❑ A value-returning method not only performs a task, but also sends a value back to the code that called it.

```
double salary;
```

```
salary = calcSalary(13.5, 43);
```

# Calling a Method

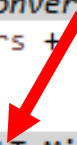
- ❑ A method executes when it is called.
- ❑ The **main** method is automatically called when a program starts, but other methods are executed by method call statements.

**displayMessage ( ) ;**

- ❑ Notice that the method modifiers and the **void** return type are not written in the method call statement. Those are only written in the method header.

# Calling a Method: Example

```
public static void main(String[] args) {  
    // variable declaration  
    int hours, minutes; // to be entered by the user  
    int convertedMinutes; // result to be calculated  
  
    // Scanner object for keyboard input  
    Scanner kb = new Scanner(System.in);  
  
    // get user input  
    System.out.print("Please enter the numbe of hours: ");  
    hours = kb.nextInt();  
    System.out.print("Please enter the number of minutes: ");  
    minutes= kb.nextInt();  
  
    convertedMinutes = convertToMinutes(hours, minutes);  
    System.out.print(hours + " hour(s) and "+minutes+" minute(s) is "+convertedMinutes+" minute(s).");  
}  
  
public static int convertToMinutes (int hours, int min) {  
    // result variable and calculation  
  
    int result = 60*hours+min;  
    return result;  
}
```



# Passing Arguments to a Method (1)

- ❑ Values that are sent into a method are called arguments.
  - variables passed to a method are the “input” for the method
- ❑ The data type of an argument in a method call must correspond to the parameter in method declaration.
  - The parameter is the variable that holds the value being passed into a method.



# Passing Arguments to a Method (2): Example

```
displayValue(5) ;
```

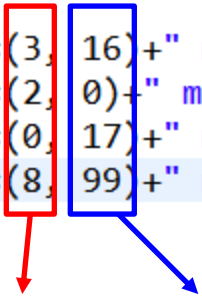
The argument 5 is copied into the parameter variable **num**.

```
public static void displayValue(int num) {  
    .....  
}
```



# Passing Arguments to a Method (3): Example

```
5 public class HourMinuteConverter {
6
7     public static void main(String[] args) {
8
9         System.out.println(convertToMinutes(3, 16)+" minutes(s).");
10        System.out.println(convertToMinutes(2, 0)+" minutes(s).");
11        System.out.println(convertToMinutes(0, 17)+" minutes(s).");
12        System.out.println(convertToMinutes(8, 99)+" minutes(s).");
13    }
14
15    public static int convertToMinutes(int hr, int min) {
16        //result variable and calculation
17        int result = 60*hr + min;
18        return result;
19    }
20
21 }
22
```



**NOTE: Order matters!**

Problems @ Javadoc Declaration Console

<terminated> HourMinuteConverter [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86

```
196 minutes(s).
120 minutes(s).
17 minutes(s).
579 minutes(s).
```

# Passing Arguments to a Method (4)

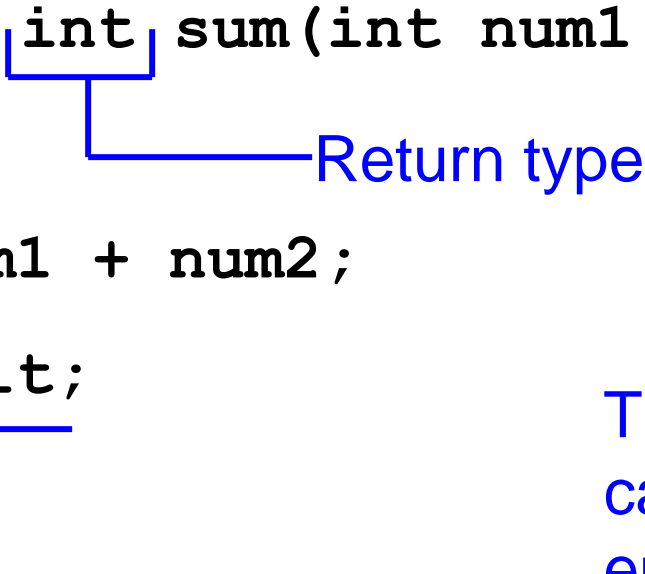
- ❑ When you pass an argument to a method, be sure that the argument's data type is **compatible** with the parameter variable's data type.
- ❑ Java will automatically perform widening conversions but narrowing conversions will cause a compiler error.

```
double d = 1.0;  
displayValue(d);
```

← **Error! Can't  
convert double to  
int**

# Returning a Value from a Method (1)

```
public static int sum(int num1, int num2) {  
    int result;  
    result = num1 + num2;  
    return result;  
}
```



The diagram illustrates the return type and the return statement in the provided code. A blue bracket underlines the word 'int' in the method signature, with a line extending to the text 'Return type'. Another blue bracket underlines the word 'result' in the return statement, with an arrow pointing down to the explanatory text below.

Return type

This expression must be of the same data type as the return type

The **return** statement causes the method to end execution and it returns a value back to the statement that called the method.

# Returning a Value from a Method (2)

```
total = sum(value1, value2);
```

```
public static int sum(int num1, int num2) {  
    60 int result;  
    result = num1 + num2;  
    return result;  
}
```

# Returning a Value from a Method (3)

```
,  
private static double calcTrainingRate(int age, int restingRate) {  
    // use the formula to calculate the target training heart rate  
    return 0.6* (220-age)+restingRate;  
}
```

# Why Pass and Return Variables?

## ❑ Methods are independent

- They communicate through passing variables and return value

## ❑ Passing variables

- The receiving method is not given access to the variable in memory (from the calling method)
- It cannot change the value stored inside the variable in the calling method

## ❑ Return value

- Output of a method
- The operation result of the receiving method is sent back to the calling method

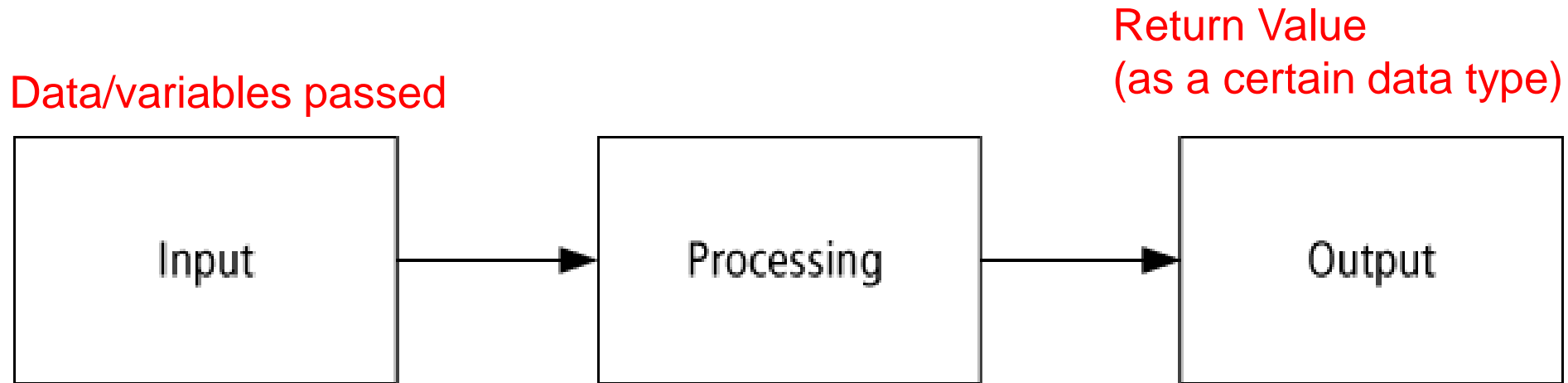
# Lab (1)

## ❑ CreditCardApproval

- In this program, the user will enter the salary and the credit rating, the program will then show the application is approved or not. If the salary is no less than \$20,000 and the credit rating is no less than 7, the application can be approved.
- In this example, we will create a method, and call the methods from the main method.



# Designing Methods



What are needed to  
make the decision?

amount of salary  
credit rating

Data type?

double amount  
int rating

What is the expected  
outcome?

approved or not

Data type?

boolean

# Top-Down Design (1)

- ❑ Each method should be cohesive
  - Do a single well-defined task.
- ❑ Each method should be as reusable as possible
  - Keep the task simple
- ❑ Pass a variable only when it is used in the receiving method

# Top-Down Design (2): Example

```
public static void main(String[] args) {
    // variable declaration
    int age; // age to be entered by user
    double height; // height to be entered
    double weight; // weight to be entered
    int restingHeartRate; // the heart rate with the user is resting. to be entered.

    double trainingHeartRate; // the target heart rate for the user. to be calculated

    // Scanner object for input
    Scanner kb = new Scanner(System.in);

    System.out.print("Please enter your age: ");
    age = kb.nextInt();
    System.out.print("Please enter your weight in kilogram: ");
    weight = kb.nextDouble();
    System.out.print("Please enter your height in cetimeter: ");
    height = kb.nextDouble();

    if(isOverweight(height, weight) && age >=18) {
        System.out.print("You can exercise to improve your health condition.");
        System.out.print("Please enter your resting heart rate: ");
        restingHeartRate = kb.nextInt();
        trainingHeartRate = calcTrainingRate(age,restingHeartRate);

        System.out.print("Please exercise to reach the training heart rate of "+
            trainingHeartRate+" per minute.");
    }
}

private static boolean isOverweight(double height, double weight) {
    boolean isOverweight = true; // the individual is overweight or not
    double bmi; // BMI to be calculated

    // Calculate the BMI
    bmi= weight/ (height/100*height/100);

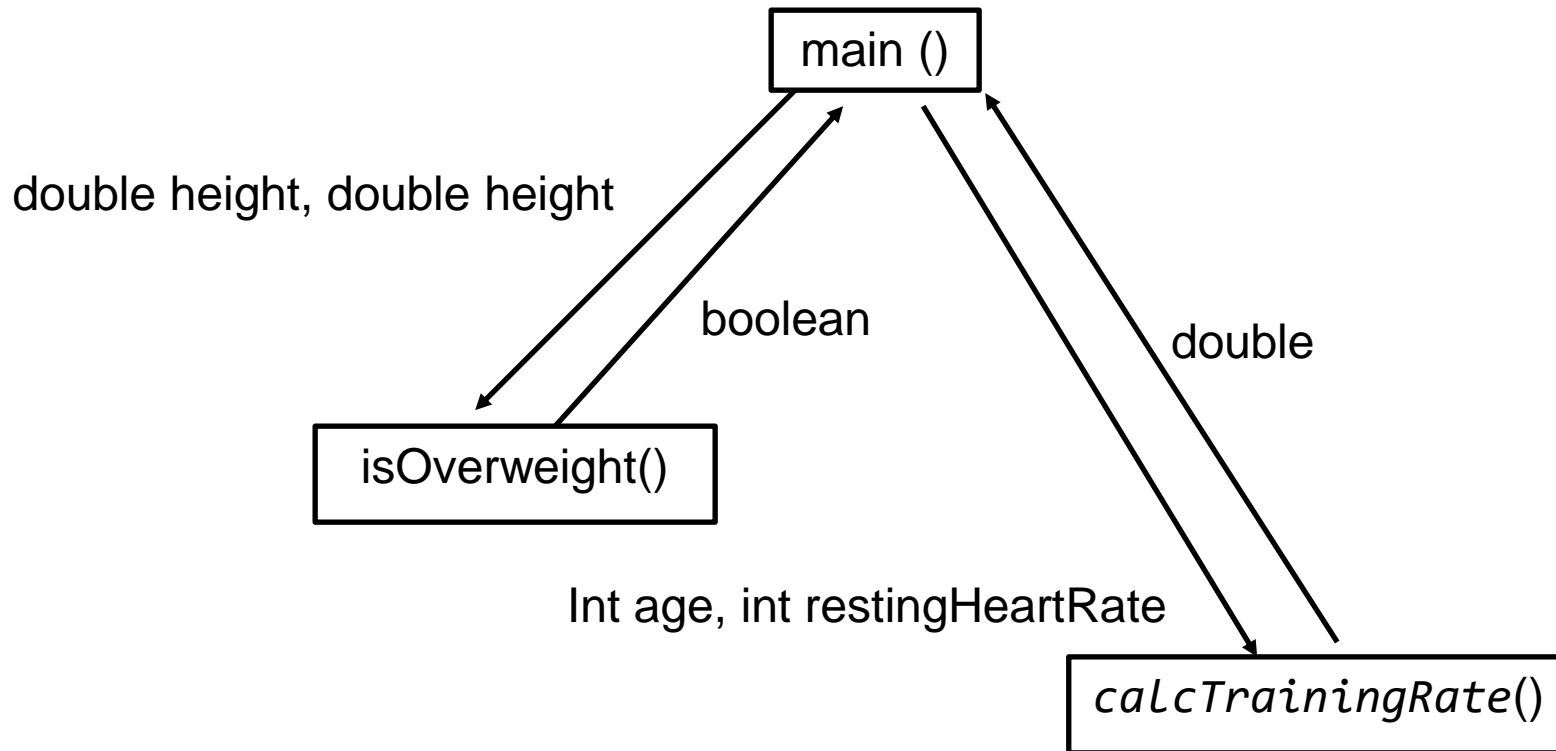
    if(bmi<=25) // determine whether the person is overweight
        isOverweight = false;

    // return the result
    return isOverweight;
}

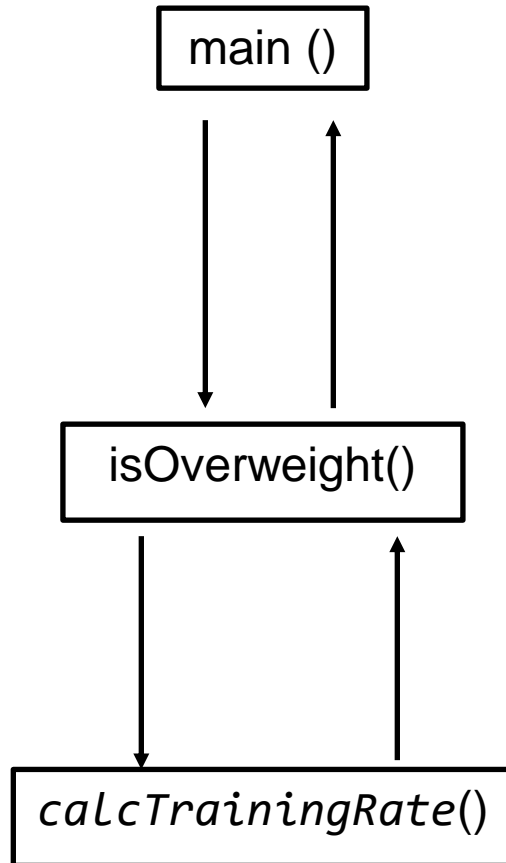
private static double calcTrainingRate(int age, int restingRate) {
    // user the formula to calculate the target training heart rate
    return 0.6* (220-age)+restingRate;
}
```

See ExerciseTargetSetter.java  
in the lab files for a clearer view.

# Top-Down Design (3)



# An Undesirable Design



`isOverweight()` is not an independent, cohesive method !!

# Exercise (1)

## □ Average Test Score

- Please write a program that asks the user to enter three test scores. The program should display the average of the test score and the letter grade for average test score, using the following grading scheme:

Score	Grade
90-100	A
80-89	B
70-79	C
Below 70	F

- The program should use a method (other than the main method) in this program. Please follow the top-down design pattern in this program

# Documenting Methods (1)

- ❑ A method should always be documented by writing comments that appear just before the method's definition.
- ❑ The comments should provide a brief explanation of the method's purpose.
- ❑ The documentation comments begin with `/ * *` and end with `* /`.

# Documenting Methods (2)

- ❑ You can provide a description of each parameter in your documentation comments by using the `@param` tag.

- ❑ General format

```
@param parameterName Description
```

- ❑ All `@param` tags in a method's documentation comment must appear after the general description.
- ❑ The description can span several lines.



# Documenting Methods (3)

- ❑ You can provide a description of the return value in your documentation comments by using the `@return` tag.
- ❑ General format

`@return` Description

- ❑ The `@return` tag in a method's documentation comment must appear after the general description. The description can span several lines.

# Documenting Methods (4)

- These comments can be used later on to generate the documentation for your program using the [Javadoc](#) Tools (It will be covered later on)

# Lab (2)

## □ CreditCardApproval

- Please add comments to document the **determineQualification()** method

# Exercise (2)

## □ Average Test Score

- Please add method comments to the methods you define in the program

# Unit Test (1)

- ❑ Unit testing is the process of individually testing a small part or unit of a program, typically a method
  - Testing a large program can be hard, and multiple bugs may interact.
  - Good practice is to test small parts of the program individually, before testing the entire program, which can more readily support finding and fixing bugs.

# Unit Test (2)

- ❑ Unit test is typically conducted by creating a separate program to check that a method returns correct output values for a variety of input values.
- ❑ Each unique set of input values is known as a **test vector**.

# Unit Test Example (1)

```
public static int convertToMinutes(int hr, int min) {  
    //result variable and calculation  
    int result = 60*hr + min;  
    return result;  
}
```

# Unit Test Example (2)

```
7- public static void main(String[] args) {
8     System.out.println("Testing started");
9
10    System.out.println("0:0, expecting 0, get "+convertToMinutes(0, 0));
11    System.out.println("0:1, expecting 1, get "+convertToMinutes(0, 1));
12    System.out.println("0:99, expecting 99, get "+convertToMinutes(0, 99));
13    System.out.println("1:0, expecting 60, get "+convertToMinutes(1, 0));
14    System.out.println("5:0, expecting 300, get "+convertToMinutes(5, 0));
15    System.out.println("2:30 expecting 150, get "+convertToMinutes(2, 30));
16
17    System.out.println("Testing completed");
18
19 }
20
21- public static int convertToMinutes(int hr, int min) {
22     //result variable and calculation
23     int result = 60*hr + min;
24     return min;
25 }
```

Problems @ Javadoc Declaration Console X

<terminated> HourMinuteConverter [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspc

```
Testing started
0:0, expecting 0, get 0
0:1, expecting 1, get 1
0:99, expecting 99, get 99
1:0, expecting 60, get 0
5:0, expecting 300, get 0
2:30 expecting 150, get 30
Testing completed
```

Anything wrong with the result?



# Problem Solving with Methods

- ❑ A large, complex problem can be solved a piece at a time by methods.
- ❑ The process of breaking a problem down into smaller pieces is called **functional decomposition**.

# Example: Sales Report (1)

- ❑ The program reads 30 days of sales amount from a file, and then displays the total sales and average daily sales
  - Ask the user to enter the name of the file
  - Get the total of the sales amounts in the file
  - Calculate the average daily sales
  - Display the total and average daily sales

# Example: Sales Report (2)

❑ Instead of writing the entire program in the **main** method, the algorithm was broken down into the following method

— getTotalSales

- Accepts the file name as an argument
- Open file, read the sales amount
- Accumulate the sales amount
- Return the total as a **double**

# Example: Sales Report (3)

- ❑ If a method calls another method that has a **throws** clause in its header, then the calling method should have the same **throws** clause.
  - Note that the `main` and `getTotalSales` methods in `SalesReport.java` have a **throws IOException** clause.
- ❑ All methods that use a **Scanner** object to open a file must throw or handle **IOException**.

# Lab (3)

## □ Sales Report

- The program reads 30 days of sales amount from a file (MonthlySales.txt), and then displays the total sales and average daily sales.
- Move the text file (MonthlySales.txt) to the project folder for easier operation.

# DecimalFormat Class (1)

□ We can also format the output number using **DecimalFormat** class

- The **DecimalFormat** class can be used to format **double** and **float** values.
- In order to use the **DecimalFormat** class, the following **import** statement must be used at the top of the program:

```
import java.text.DecimalFormat;
```

# DecimalFormat Class (2)

## ❑ Define the formatter string object

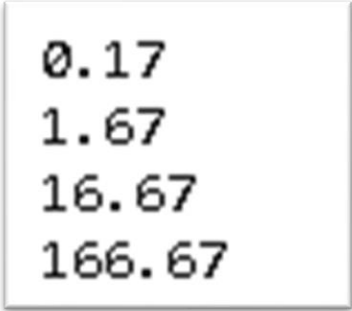
Symbol	Usage
0	A digit - always displayed
#	A digit, leading zeroes are omitted.
,	thousand separator
%	Multiplies by 100 and shows number as percentage

## ❑ Format string

- Use the format() method of the object

# DecimalFormat Example (1)

```
double number1 = 0.1666666666666667;  
double number2 = 1.6666666666666667;  
double number3 = 16.666666666666667;  
double number4 = 166.66666666666667;  
  
// Create a DecimalFormat object.  
DecimalFormat formatter = new DecimalFormat("#0.00");  
  
// Display the formatted variable contents.  
System.out.println(formatter.format(number1));  
System.out.println(formatter.format(number2));  
System.out.println(formatter.format(number3));  
System.out.println(formatter.format(number4));
```

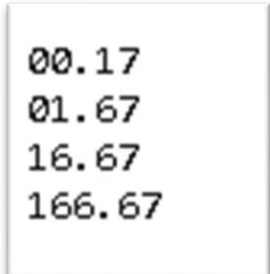


0.17  
1.67  
16.67  
166.67



# DecimalFormat Example (2)

```
double number1 = 0.166666666666667;  
double number2 = 1.666666666666667;  
double number3 = 16.666666666666667;  
double number4 = 166.66666666666667;  
  
// Create a DecimalFormat object.  
DecimalFormat formatter = new DecimalFormat("00.00");  
  
// Display the formatted variable contents.  
System.out.println(formatter.format(number1));  
System.out.println(formatter.format(number2));  
System.out.println(formatter.format(number3));  
System.out.println(formatter.format(number4));
```



```
00.17  
01.67  
16.67  
166.67
```

# DecimalFormat Example (3)

```
double number1 = 123.899;  
double number2 = 1233.899;  
double number3 = 12345.899;  
double number4 = 123456.899;  
double number5 = 1234567.899;  
  
// Create a DecimalFormat object.  
DecimalFormat formatter = new DecimalFormat("#,##0.00");  
  
// Display the formatted variable contents.  
System.out.println(formatter.format(number1));  
System.out.println(formatter.format(number2));  
System.out.println(formatter.format(number3));  
System.out.println(formatter.format(number4));  
System.out.println(formatter.format(number5));
```

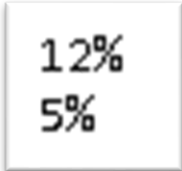
```
123.90  
1,233.90  
12,345.90  
123,456.90  
1,234,567.90
```

# DecimalFormat Example (4)

```
double number1 = 0.12;
double number2 = 0.05;

// Create a DecimalFormat object.
DecimalFormat formatter = new DecimalFormat("#0%");

// Display the formatted variable contents.
System.out.println(formatter.format(number1));
System.out.println(formatter.format(number2));
```



12%  
5%