Chapter 2 Solutions

1. Identify all errors in the following program:

```
/*

      Program Exercise1

      Attempting to display a frame window

// ← 1
import swing.JFrame; ← 2

class Exercise 1 { ← 3
      public void Main() { ← 4
            JFrame frame; ← 5
            frame.setVisible(TRUE) ← 6
      }
}
```

1. *Should be* `*/` *to terminate the multi-line comment.*
2. *Correct statement is:*
   ```
   import javax.swing.JFrame;
   ```
   or
   ```
   import javax.swing.*;
   ```
3. *The class name is not a valid identifier. It should be* **`Exercise1`**.
4. *The correct declaration for the main method is*
   ```
   public static void main(String[] args)
   ```
5. *Should include an object creation statement*
   ```
   JFrame frame = new JFrame();
   ```
6. *Semicolon is missing after the statement. Also,* TRUE *is not valid syntax in JAVA. The correct statement is:*
   ```
   frame.setVisible(true);
   ```

2. Identify all errors in the following program:

```
// ← 1
      Program Exercise2

      Attempting to display a frame of the size 300 by 200
pixels
// ← 2
import Javax.Swing.*; ← 3

class two {

      public static void main method() { ← 4
            myFrame JFrame; ← 5
            myFrame = new JFrame();
            myFrame.setSize(300, 200);
            myFrame.setVisible(); ← 6
```

```
    }
}
```

1. *Should be* `/*` *to begin a multi-line comment*
2. *Should be* `*/` *to terminate a multi-line comment*
3. *The correct package name is* **`javax.swing`** *Java is case sensitive.*
4. *The correct declaration for the main method is*
   public static void main(String[] args)
5. *When declaring a variable, the type comes before the variable name. The correct statement is:*
   ```
   JFrame myFrame;
   ```
6. *The* **`setVisible`** *method requires a boolean parameter. The correct statement is:*
   ```
   myFrame.setVisible(true);
   ```

3. Identify all errors in the following program:

```
/*
      Program Exercise3

      Attempting to display the number of characters
      in a given input.
 */
← 1
class three {
      public static void main( ) { ← 2
            String input;
            input = JOptionPane("input:"); ← 3

            JOptionPane.showMessageDialog( null,"Input has "+
                                           input.length() +
                                           " characters");
      }
}
```

1. **`JOptionPane`** *is defined in the* **`javax.swing`** *package. In order to use* **`JOptionPane`** *we must import* **`javax.swing`**. *We must insert the following statement*
   ```
   import javax.swing.*;
   ```
   or
   ```
   import javax.swing.JOptionPane;
   ```
2. *The correct declaration for the main method is*
   ```
   public static void main(String[] args)
   ```
3. *In order to get user input from a* **`JOptionPane`**, *we must use the* **`showInputDialog`** *method. The correct statement is:*
   ```
   input = JOptionPane.showInputDialog(null, "input:");
   ```

4. Describe the purpose of comments. Name the types of comments available. Can you include comment markers inside a comment?

Purposes:

To describe the program, methods, data values, and other components
To explain the meaning of code
To provide other helpful information to improve the readability

Types:

Multi-line comments
Single-line comments
Javadoc comments

*In some cases **yes** (like including // in a multi-line comment), but in other cases no.*

5. What is the purpose of the import statement? Does a Java program always have to include an import statement?

   Imports allow us to use or refer to classes in the imported package without having to use the fully qualified class name. Import statements are not always required. For example, any classes in the **java.lang** *package can be used without importing. If we use fully qualified names in our program, the import statements are not necessary.*

6. Show the syntax for importing one class and all classes in a package.

   One class:

   ```
   import <package name>.<class name>;
   ```

   All classes:

   ```
   import <package name>.*;
   ```

7. Describe the class that must be included in any Java application.

   Every java application requires one class to be designated as its main class. The designated main class must include the **main** *method.*

8. What is a reserved word? List all the Java reserved words mentioned in this chapter.

   A reserved word is an identifier that is reserved by a programming language and used for a specific purpose. It can be used to designate a special operator such as **new** *or syntactical element of the language such as* **class** *and* **public**.

   Reserved words introduced in this chapter are **new, import, class, public, static, void**, **true**, **false**, *and* **null**.

9. Which of the following are invalid Java identifiers?

   a. R2D2
   b. Whatchamacallit
   g. 3CPO
   h. This is okay.

c. HowAboutThis?
d. Java
e. GoodChoice
f. 12345
i. thisIsReallyOkay
j. DEFAULT_AMT
k. Bad-Choice
l. A12345

*Invalid identifiers are **c** (no question mark is allowed), **f** (the first character must be a non-digit), **g** (the first character must be a non-digit), **h** (no space is allowed), and **k** (no hyphen is allowed).*

10. Describe the steps you take to run a Java application and the tools you use in each step. What are source files and bytecode files? What different types of errors are detected at each step?

The cycle is Edit-Compile-Run. In the Edit state, you enter a program using any text editor. The file you create is called the source file, which is in human readable form and contains the Java statements that make up the program. In the Compile stage, you compile the source file to create a bytecode file. The tool (another piece of software) to compile the Java source file is called **javac**. *In the Run stage, you execute the bytecode file by using the Java interpreter called* **java**. *Note: The Java interpreter is called the **Java Virtual Machine (JVM)**.*

Syntax and semantic errors are caught by the compiler. A compiler cannot catch logic errors. Logic errors are detected when the program is actually executed.

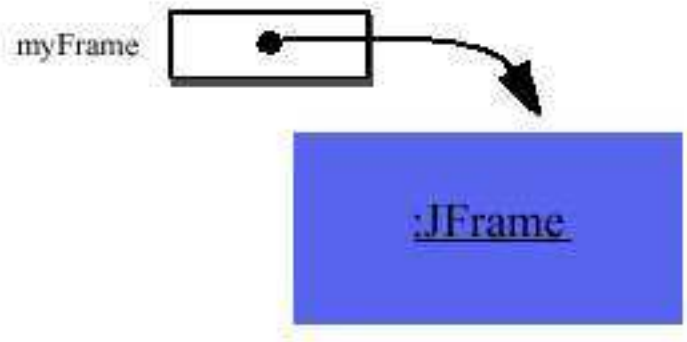
11. Describe the difference between object declaration and object creation. Use a state-of-memory diagram to illustrate the difference.

Object declaration associates an object name to a designated class. It also reserves a small amount (like four bytes) of memory to store a reference to an object.

```
JFrame myFrame;
```

Object creation (provided that the object is already declared) allocates the memory space for the object itself and stores the reference (arrow) to this object in the object name.

```
myFrame = new JFrame();
```



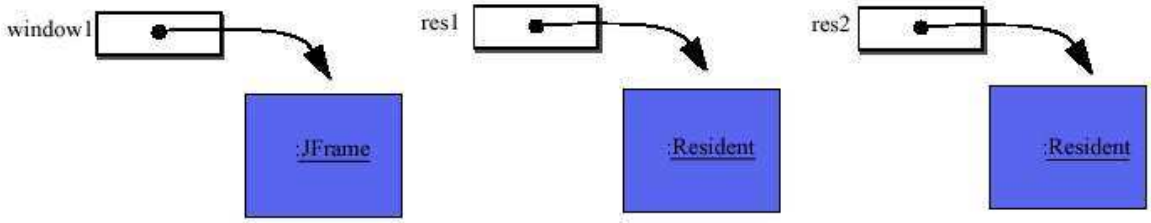
12. Show a state-of-memory diagram after each of these statements is executed:

```
JFrame          window1;
Resident   res1, res2;

window1    = new JFrame();
res1       = new Resident();
res2       = new Resident();
```

After 3 objects are declared:

After 3 new operations are executed:

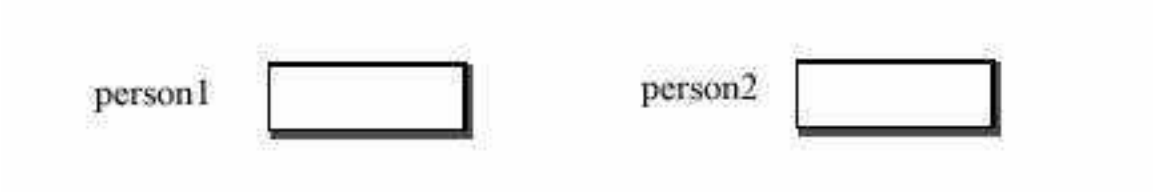


13. Show a state-of-memory diagram after each of these statements is executed:

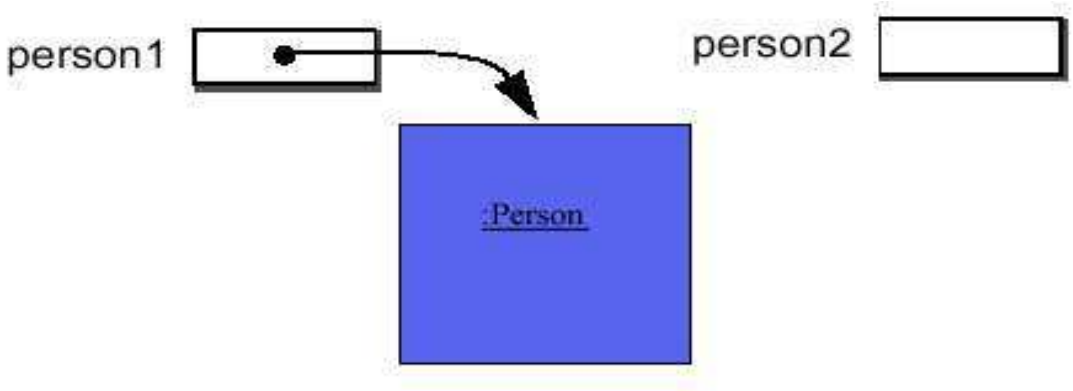
```
Person          person1, person2;
```

```
person1    = new Person();
person2    = new Person();
person2    = new Person();
```

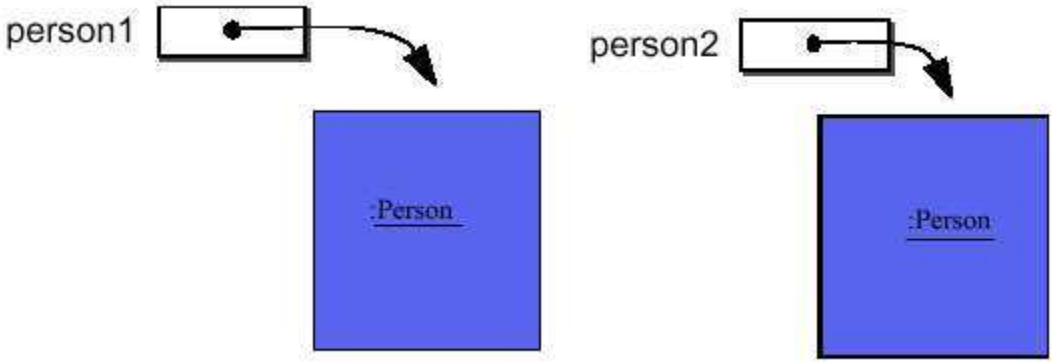
After two objects are declared:



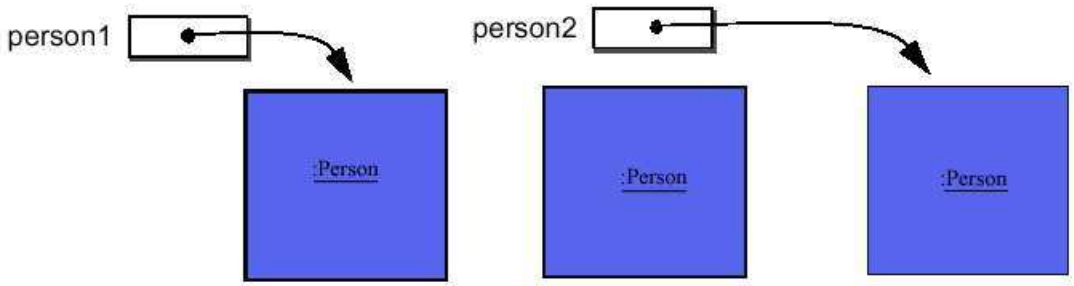
After the first new operation is executed:



After the second new operation is executed:



After the third new operation is executed:



14. Which of these identifiers violate the naming convention for class names?

a. r2D2	e. CPO
b. whatchamacallit	f. ThisIsReallyOkay
c. Java	g. java
d. GoodName	h. badName

*The convention is to use an uppercase letter for the first character. The first character of a new word is also capitalized. All other letters are lowercase. The identifiers **a**, **b**, **g**, and **h** violate the convention. The identifier CPO should be considered as valid if it is interpreted as some kind of acronym.*

15. Which of these identifiers violate the naming convention for object names?

a. R2D2	e. 3CPO
b. isthisokay?	f. ThisIsReallyOkay
c. Java	g. java
d. goodName	h. anotherbadone

*The convention for object names is almost identical to the one for class names, except the first character is lowercase. The identifiers **a**, **b**, **c**, **e**, **f**, and **h** violate the convention. Notice that **b** and **e** are not even valid identifiers.*

16. For each of these expressions determine its result. Assume the value of **text** is a string **Java Programming**.

```
String text = "Java Programming";
```

```
a. text.substring(0, 4)
b. text.length( )
c. text.substring(8, 12)
d. text.substring(0, 1) + text.substring(7, 9)
e. text.substring(5, 6) + text.substring(text.length() - 3,
   text.length())
```

a. Java
b. 16

c. gram
d. Jog
e. Ping

17. Write a Java application that displays today's date in this format: Sunday November 10, 2002.

    See Exercise2_17.java

18. Write a Java application that displays a frame window 300 pixels wide and 200 pixels high with the title **My First Frame**. Place the frame so that its top, left corner is at a position 50 pixels from the top of the screen and 100 pixels from the left of the screen. To position a window at a specified location, you use the **setLocation** method, as in

    ```
    //assume mainWindow is declared and created
    frame.setLocation( 50, 50 );
    ```

    Through experimentation, determine how the two arguments in the **setLocation** method affects the positioning of the window.

    See Exercise2_18.java

19. Write a Java application that displays the two messages **I Can Design** and **And I Can Program**, using two separate dialogs.

    See Exercise2_19.java

20. Write a Java application that displays the two messages **I Can Design** and **And I Can Program**, using one dialog but in two separate lines.

    See Exercise2_20.java

21. Write a Java application that displays a very long message. Try a message that is wider than the display of your computer screen, and see what happens.

    See Exercise2_21.java

22. Because today's computers are very fast, you will probably not notice any discernable difference on the screen between the code

    ```
    JFrame myWindow;
    myWindow = new JFrame( );
    myWindow.setVisible( true );
    ```

    and

```
JFrame myWindow;
myWindow = new JFrame( );
myWindow.setVisible( true );
myWindow.setVisible( false );
myWindow.setVisible( true );
```

One way to see this disappearance and reappearance of the window is to put a delay between the successive **setVisible** messages. To put a delay, you can use a **Clock** object from the **javabook** package. Here's a simple usage of the **Clock** class:

```
import javabook.*;
...
Clock myClock;
myClock = new Clock();

// put statement X here

myClock.pause( 2 );

// put statement Y here
```

The unit for the argument you pass in the **pause** message is seconds. If you want a 0.5-s delay, for example then you pass 0.5 as an argument. Using the **Clock** class, write a program that makes a **JFrame** object appear, disappear, and appear again. The window remains visible for 5 s when it appears for the first time, and once it disappears, it won't reappear for 3 s.

See Exercise2_22.java

23. *At the textbook website* or at the author's website, you will find a Java package called **galapagos** (**www.drcaffeine.com/packages**). The **galapagos** package includes a **Turtle** class that is modeled after Seymour Papert's logo. This **Turtle** has a pen, and when you move the **Turtle**, its pen will trace the movement. So by moving a **Turtle** object, you can draw many different kinds of geometric shapes. For example, this program commands a **Turtle** to draw a square:

```
import galapagos.*;

class Square {
     public static void main( String[] args ) {
          Turtle turtle;
          turtle = new Turtle();

          turtle.move( 50 );  //move 50 pixels
          turtle.turn( 90 );    //turn 90 deg
counterclockwise

          turtle.move( 50 );
          turtle.turn( 90 );
```

```
            turtle.move( 50 );
            turtle.turn( 90 );

            turtle.move( 50 );
        }
    }
```

Write a program to draw a triangle. Read the documentation and see if you can find a way to draw the square in a different color and line thickness.

See Exercise2_23.java

You can use the **penSize** *and* **penColor** *methods to change the thickness and color of the* **Turtle's** *pen.*

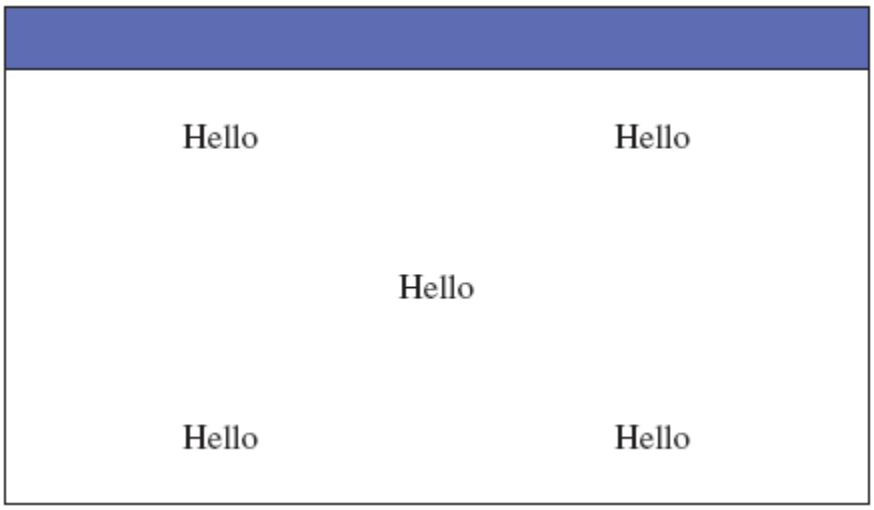
24. Write a program to draw a star, using a **Turtle** from Exercise 23.

    See Exercise2_24.java

25. Write a program to draw a big letter J, using a **Turtle** from Exercise 23.

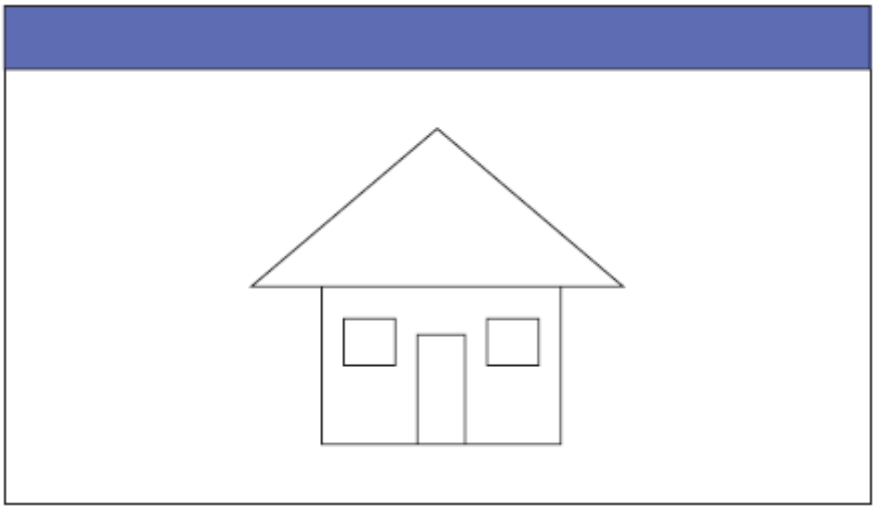
    See Exercise2_25.java

26. Using a **Turtle** from Exercise 23, write a Java application that displays the text **Hello** as illustrated here:



    See Exercise2_26.java

27. Using a **Turtle** from Exercise 23 and employing the incremental development steps, build a Java application that draws a house.



See Exercise2_27.java

28. Add the moon and a tree to the house you drew in Exercise 27.

    See Exercise2_28.java

29. Follow the incremental development methodology explained in this chapter to implement a program for the following problem statement. You must clearly write down the program tasks, create a design document with class descriptions, and draw the program diagram. Identify the development steps. State any assumptions you must make about the input. Articulate any design alternatives and justify your selection. Be sure to perform adequate testing at the end of each development step.

    Problem Statement: Write an application that asks the user for his or her birth date and replies with the day of the week on which he or she was born.

    We learned in this chapter that we can create a **Date** object for today's date by writing

    ```
    import java.util.*;
    …
    Date today = new Date();
    ```

    To create a **Date** object for a date other than today, we can use the **Date** class from the **java.sql** package. (A more general and flexible way to deal with a date by using the **GregorianCalendar** class is introduced in Chapter 3.) Notice that there are two distinct classes with the same name **Date**, but from different packages – one from **java.util** and another from **java.sql**. To distinguish the two we will use the fully qualified names. To create a new **java.util.Date** object, we can call the class method **valueOf** of the **java.sql.Date** class with the string representation of a date. The string representation must be in the format yyyy-MM-dd. For example, to create a **java.util.Date** object for July 4, 1776, we write

```
java.util.Date bdate = java.sql.Date.valueOf("1776-07-04");
```

Notice that **valueOf** is a class method of the Data class in the **java.sql** package. Calling it with a correct argument will return a **java.util.Date** object for the specified date.

See Exercise2_29.java

Design Document: Birthday Program

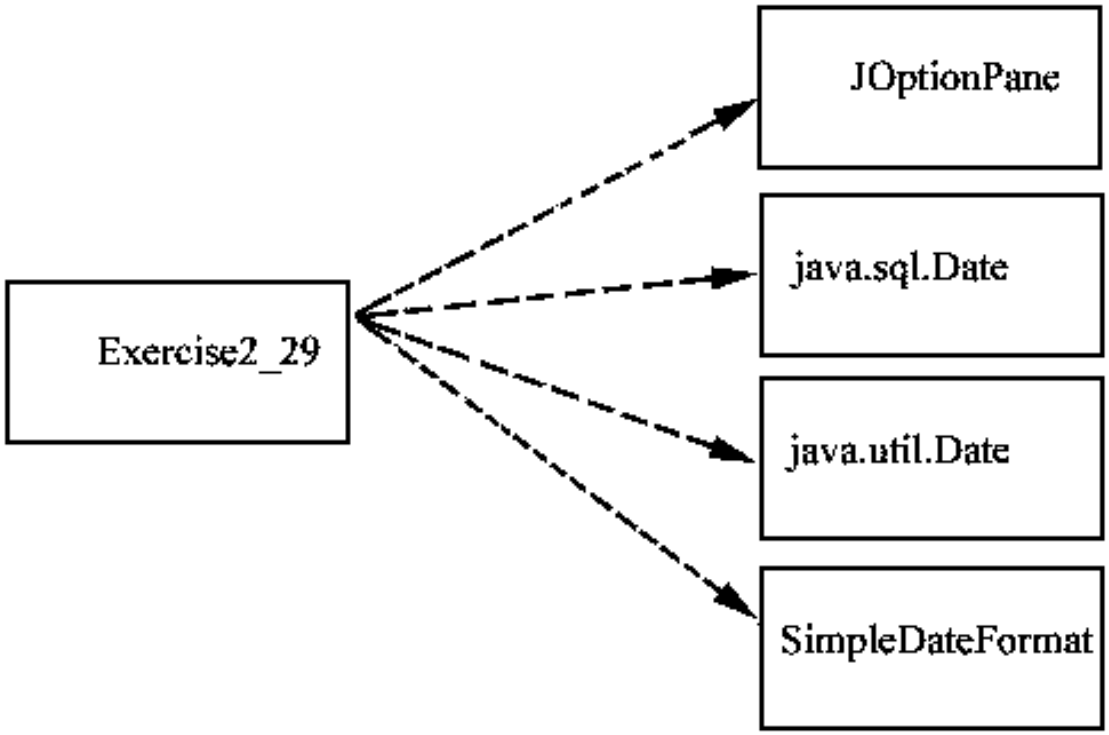
Class	***Purpose***
Exercise2_29	*The main class of the program.*
JOptionPane	*The* **showInputDialog** *method is used for getting the birth date. The* **showMessageDialog** *method is used for displaying the day of the week.*
java.util.Date	*Used to hold the date the user was born*
java.sql.Date	*Used to generate the java.util.Date version of the user's birth date.*
SimpleDateFormat	*Used to display the day of the week to the user.*

Assumptions: The user enters valid input in the form yyyy-mm-dd

Development Step 1: Getting Input

Alternative: Get the month, date, and year separately

Development Step 2: Computing and Displaying the Day of the Week



30. Repeat Exercise 29 for this problem statement:

Problem Statement: Write an application that asks the user for her or his full name in the format

first middle last

and replies with the name in the format

last, first middle-initial.

where the last name is followed by a comma and the middle initial is followed by a period.

For example, if the input is

Decafe Chai Latte

then the output is

Latte, Decafe C.

See Exercise2_30.java

Design Document: Name Manipulation

Class	***Purpose***
Exercise2_30	*The main class of the program.*
JOptionPane	*The* **showInputDialog** *method is used for getting the user's full name. The* **showMessageDialog** *method is used for displaying the result.*
String	The class is used for String manipulation, extracting the first name, last name, and middle initial.

Assumptions: The user enters valid input in the form first middle last

Development Step 1: Getting Input

Alternative: Get the first, middle, and last names separately

Development Step 2: Computing and Displaying the reformatted name