

## JOptionPane:

Up until now you have been using the **Scanner** class for input and displayed your output to the **command prompt**.

Java uses different techniques for input and output purposes. In this lesson you will learn about the **JOptionPane** class in the **javax.swing** package for input and output purposes. Remember to always import the **javax.swing** package when you want to use the **JOptionPane** class.

This will be your first introduction to GUI's, since **JOptionPane** is a GUI.

Just like you only used input with the **Scanner** class and output to the command prompt in front-end applications, you will also use **JOptionPane** in front-end applications.

The methods of the **JOptionPane** class will allow you to create dialog boxes to:

- Display information to the user
- Request information from the user
- Give a user a choice with buttons

While the **JOptionPane** class may appear complex because of the large number of methods, almost all uses of this class are **one-line** calls to one of the static **showXxxDialog** methods shown below:

Method Name	Description
showConfirmDialog	Asks a confirming question, like yes/no/cancel.
showInputDialog	Prompt for some input.
showMessageDialog	Tell the user about something that has happened.

Show the students the complete API of the **JOptionPane** class.

## Output:

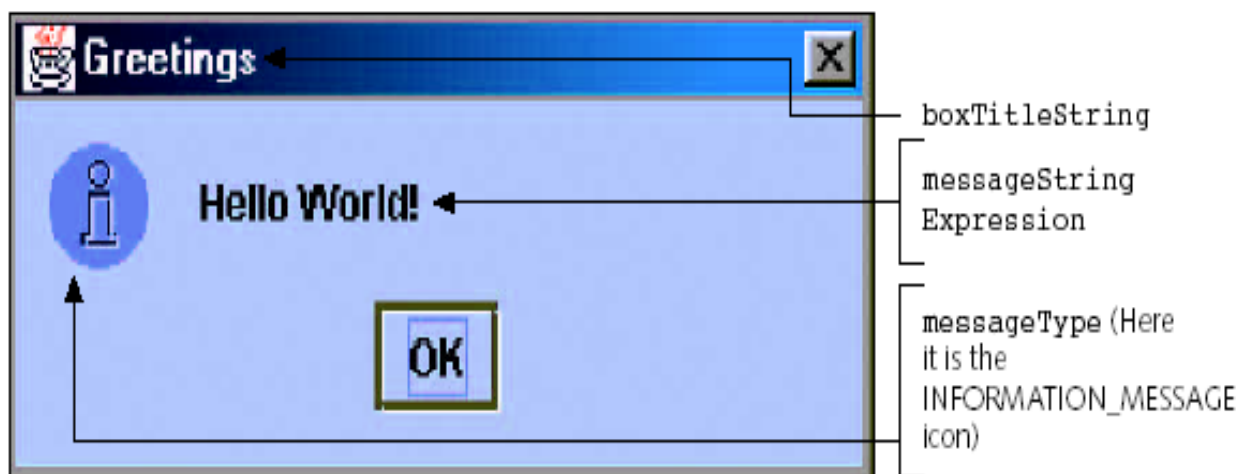
To output information to the screen with the **JOptionPane** class we use the **showMessageDialog()** method.

You will note in the API that the `showMessageDialog()` method is of type static and returns no value. It will return no value, since it is only used for output purposes. And because it is of type static, we will use the class name (in this case `JOptionPane`) to reference this method.

Here is an example:

```
JOptionPane.showMessageDialog(null, "Hallo World!", "Greetings,  
JOptionPane.INFORMATION_MESSAGE);
```

Below is the output of this message box:



The method must receive four parameters:

Parameter	Description
parentComponent	This is an object that represents the parent of the dialog box. For now, we will specify the <code>parentComponent</code> to be <code>null</code> , in which case the program uses a default component that causes the dialog box to appear in the middle of the screen. Note that <code>null</code> is a reserved word in Java.
messageStringExpression	The <code>messageStringExpression</code> is evaluated and its value appears in the dialog box.
boxTitleString	The <code>boxTitleString</code> represents the title of the dialog box.
messageType	An <code>int</code> value representing the type of icon that will appear in the dialog box. Alternatively, you can use certain <code>JOptionPane</code> options described below.

The message type can be one of the following:

messageType	Description
JOptionPane.ERROR_MESSAGE	The error icon is displayed in the dialog box.
JOptionPane.INFORMATION_MESSAGE	The information icon is displayed in the dialog box.
JOptionPane.PLAIN_MESSAGE	No icon appears in the dialog box.
JOptionPane.QUESTION_MESSAGE	The question icon, question mark, is displayed in the dialog box.
JOptionPane.WARNING_MESSAGE	The warning icon, the exclamation point, is displayed in the dialog box.

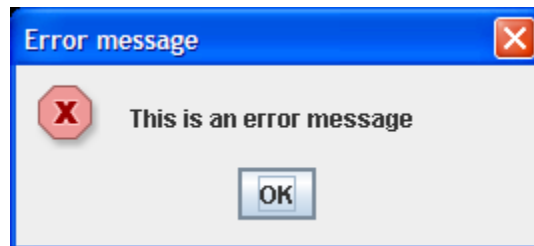
Note the overloaded showMessageDialog() methods that can be used:

static void	<code>showMessageDialog(Component parentComponent, Object message)</code>
static void	<code>showMessageDialog(Component parentComponent, Object message, String title, int messageType)</code>

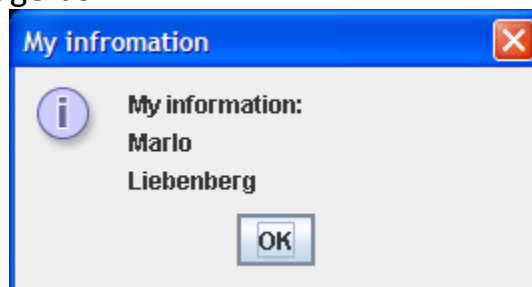
Exercises:

Create the following Message boxes:

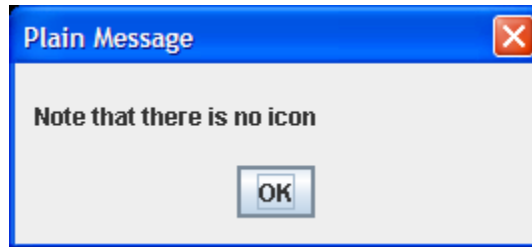
1. Error message box:



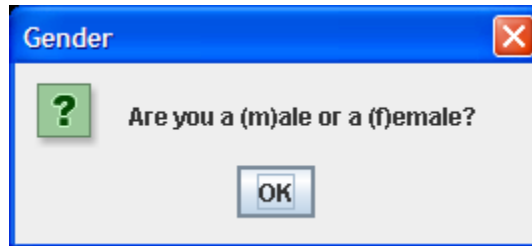
2. Information message box:



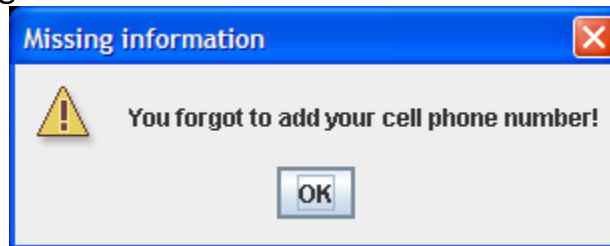
3. Plain message box:



4. Question message box:



5. Warning message box:



### Input:

We used the **Scanner** class to get input from the user. Now we will use the **showInputDialog()** method.

You will note that the method is of type static, which means we have to use the class name to reference the method, just like the **showMessageDialog()** method.

Unlike the **Scanner** class, which we had a method to read the different data types and **String** values, the **showMessageDialog()** method will always return a **String** value.

As you learned previously to parse numeric string values to numeric data types, you must always parse the input string to the numeric data type you asked the user to enter. Remember that the parse methods of the different

data type classes are all static method, so we use the class name to reference the method.

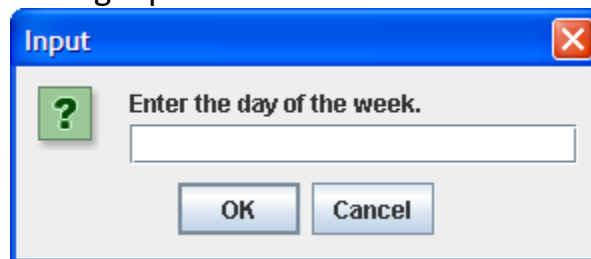
Here is the syntax:

**String <variable name> = JOptionPane.showInputDialog(<Message>);**

Here is an example where we ask the user to enter day of the week:

**String input = JOptionPane.showInputDialog("Enter the day of the week.");**

It will display the following input box:



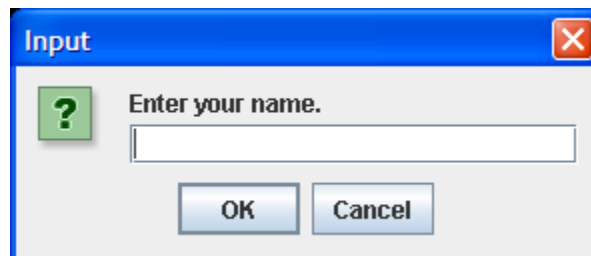
Note the overloaded showInputDialog() methods that can be used:

static <b>String</b>	<b>showInputDialog</b> ( <b>Component</b> parentComponent, <b>Object</b> message)
static <b>String</b>	<b>showInputDialog</b> ( <b>Component</b> parentComponent, <b>Object</b> message, <b>String</b> title, int messageType)
static <b>String</b>	<b>showInputDialog</b> ( <b>Object</b> message)

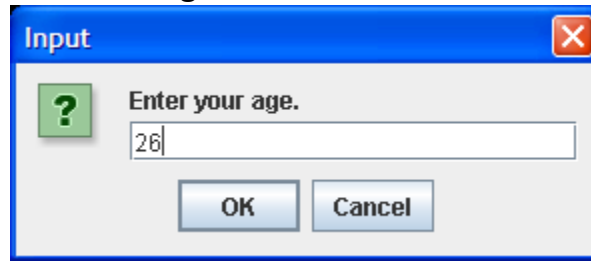
Exercises:

Create the following input dialog boxes. If it asks for a specific data type, parse it to the correct data type.

1. Ask the user to enter their name.

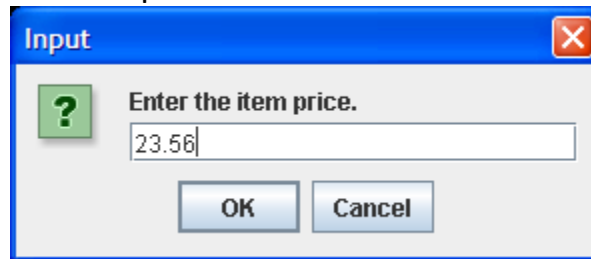


2. Ask the user to enter their age.



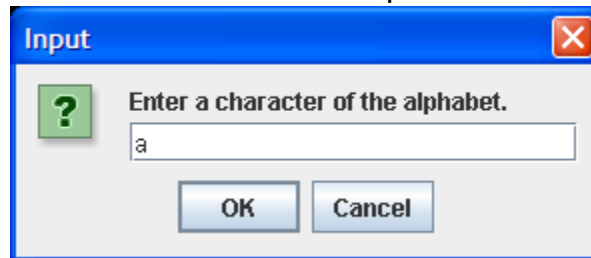
A screenshot of a Windows-style 'Input' dialog box. The title bar is blue with the word 'Input' and a red close button. The dialog has a light gray background. On the left is a green square icon with a white question mark. To its right is the text 'Enter your age.' Below this is a text input field containing the number '26'. At the bottom are two buttons: 'OK' and 'Cancel'.

3. Ask the user to enter the price of the item.



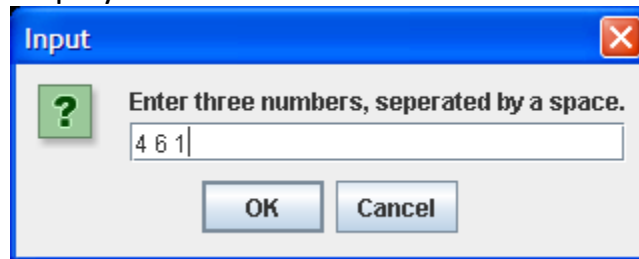
A screenshot of a Windows-style 'Input' dialog box. The title bar is blue with the word 'Input' and a red close button. The dialog has a light gray background. On the left is a green square icon with a white question mark. To its right is the text 'Enter the item price.' Below this is a text input field containing the number '23.56'. At the bottom are two buttons: 'OK' and 'Cancel'.

4. Ask the user to enter a character of the alphabet.

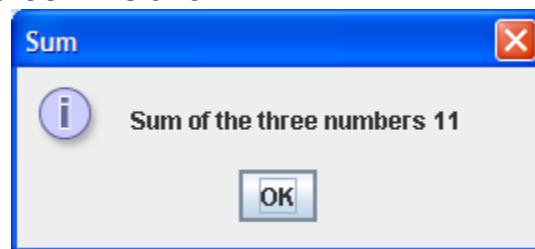


A screenshot of a Windows-style 'Input' dialog box. The title bar is blue with the word 'Input' and a red close button. The dialog has a light gray background. On the left is a green square icon with a white question mark. To its right is the text 'Enter a character of the alphabet.' Below this is a text input field containing the letter 'a'. At the bottom are two buttons: 'OK' and 'Cancel'.

5. Ask the user to enter three numbers separated by a space. Use the `split()` method of the `String` class to retrieve the three entered numbers. Then calculate and display the sum of the three numbers.



The output must look like this.

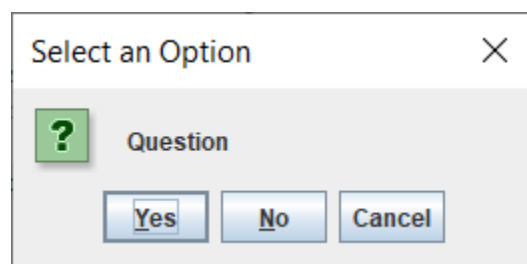


### Confirmation Input:

The **`showConfirmDialog()`** method is commonly used when you provide the user with a question that yield a YES, NO, or CANCEL answer.

A confirmation dialog box can be created using the following statement:

```
int option = JOptionPane.showConfirmDialog(null, "Question");
```



When a button is clicked, the method returns an option value. The value is **`JOptionPane.YES_OPTION (0)`** for the **Yes** button, **`JOptionPane.NO_OPTION (1)`** for the **No** button, and **`JOptionPane.CANCEL_OPTION (2)`** for the **Cancel** button.