



ORACLE

Academy



Java Foundations

3-5

Keyboard Input

ORACLE
Academy



Objectives

- This lesson covers the following objectives:
 - Understand user input
 - Create a JOptionPane to collect user input
 - Use a Scanner to collect input from the console
 - Use a Scanner to collect input from a file
 - Understand how a Scanner handles tokens and delimiters



Why Should You Get User Input?

- When you manually assign values to variables, this is known as hard-coding values:

```
String input = "This is a String";
```

- You can easily change hard-coded values because you have the source code and a Java IDE:

```
String input = "This is a different String";
```

- But when you distribute software, your users won't have the same luxury

Types of User Input

- Examples of user input include ...
 - Pressing a button on a game controller
 - Entering an address on a GPS
 - Entering numbers and functions into a calculator
 - Telling people your name
- But without user input ...
 - When will the game make your character jump?
 - Where will your GPS guide you?
 - What numbers will your calculator crunch?
 - What will people call you?

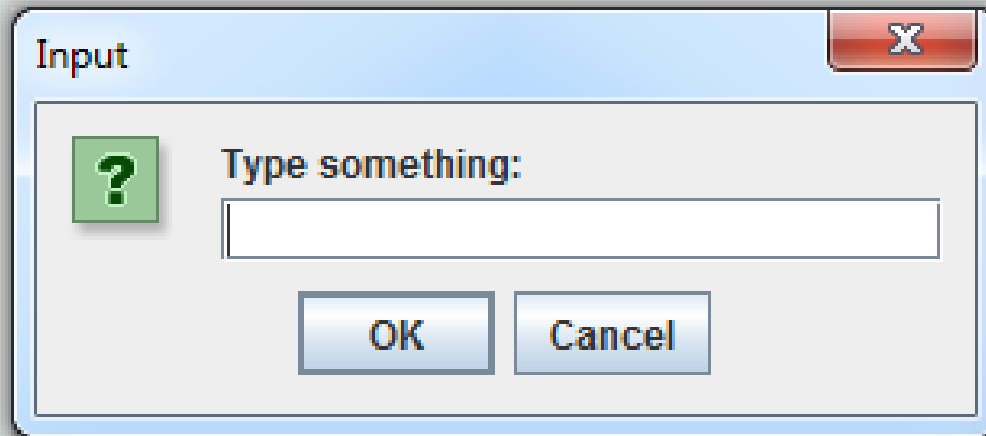
How to Get User Input

- There are many ways to get user input:
 - Buttons (physical or virtual)
 - Wheels and dials
 - Voice recognition
 - Text dialog boxes
 - Property files
- Java offers many ways of getting user input, including
 - ...
 - Swing JOptionPane
 - JavaFX (a successor of Swing, covered later)
 - Scanner

JOptionPane

- This is a simple way to get input from users:

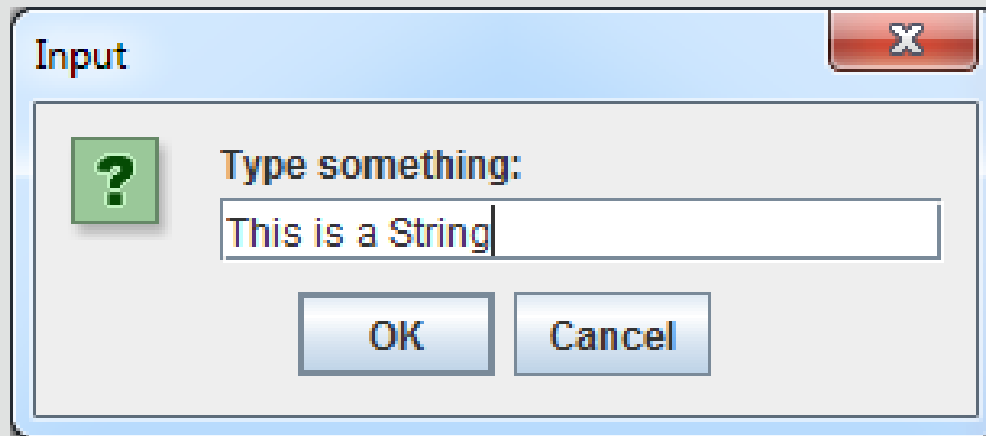
```
JOptionPane.showInputDialog("Type something:");
```



JOptionPane Returns Strings

- The input can be stored as a String:

```
String input = JOptionPane.showInputDialog("Type something:");
```



- This is equivalent to writing:

```
String input = "This is a String";
```


Exercise 1, Part 1

- Create a new project and add the `Input01.java` file to the project
- Create a `JOptionPane`:
 - NetBeans will complain
 - Follow the NetBeans suggestion of importing `javax.swing.JOptionPane`
 - We'll cover importing in another section

Exercise 1, Part 2

- Store this input as a String
- Print the String variable
- Parse the String as a separate int variable
 - You'll need to input a value that can be parsed
 - Print this value +1
- Try creating a dialog box, parsing it, and initializing an int in a single line
- You should have only one semicolon (;)

Condensed Code

- You could spread your input, parsing and calculating across several lines:

```
String inputString = JOptionPane.showInputDialog("??");  
int input = Integer.parseInt(inputString);  
input++;
```

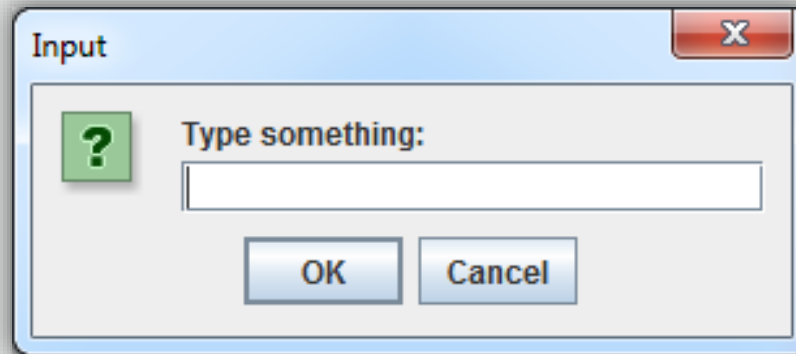
- Or condense this into a single line:

```
int input = Integer.parseInt(JOptionPane.showInputDialog("??")) +1;
```

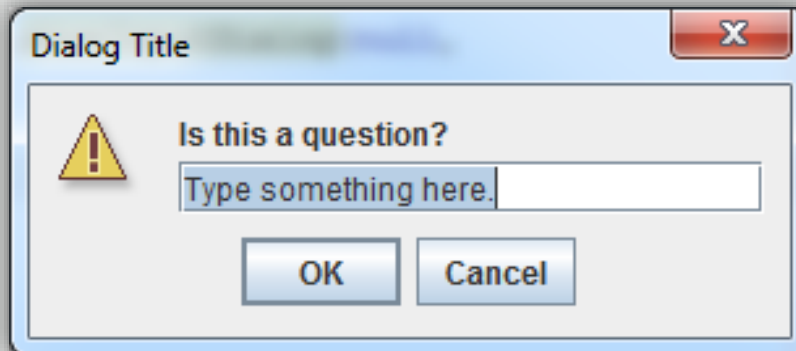
- This choice is a matter of personal preference
 - But if you need to reference certain values again later, it would be helpful to store these values in a variable

Different InputDialogs

- We created a simple InputDialog:



- With more complicated code, we can customize the InputDialog more:

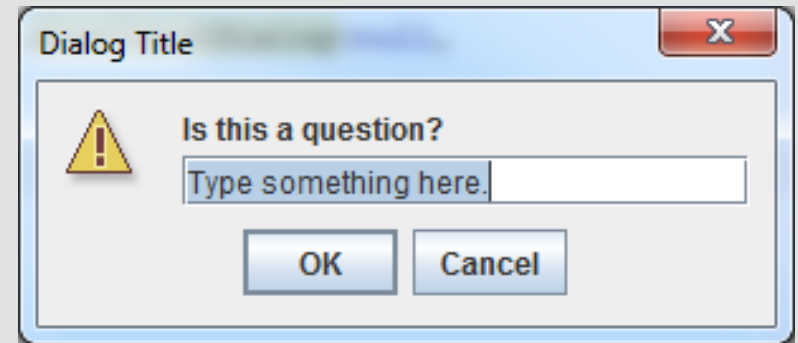


More Options with InputDialogs

- This version of an InputDialog doesn't return a String
- The result must be cast to a String to be usable:

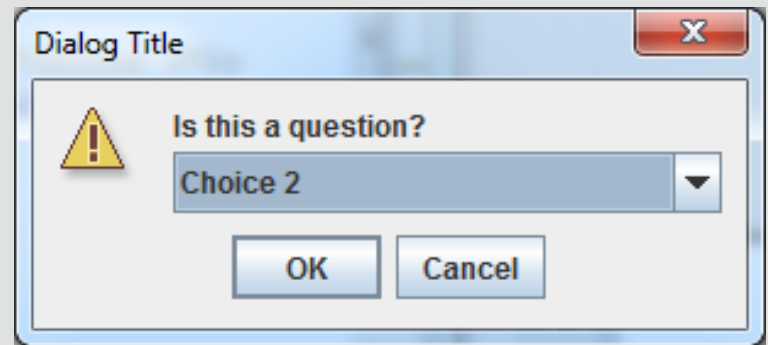
Casting

```
String input = (String)JOptionPane.showInputDialog(null,  
"Is this a question?",  
"Dialog Title",  
2,  
null,  
null,  
"Type something here.");
```



More Options with InputDialogs

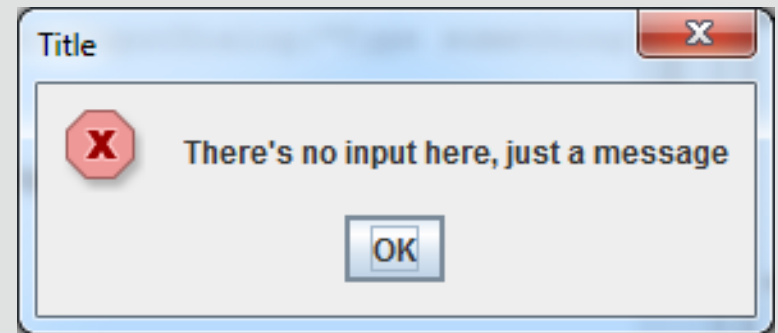
- To avoid unwanted input, it's possible to provide only acceptable values to users
- Some of this syntax is discussed in greater detail in Section 8



```
String[] acceptableValues = {"Choice 1", "Choice 2", "Choice 3"};  
Input2 string= (String)JOptionPane.showInputDialog(null,  
    "Is this a question?",  
    "Dialog Title",  
    2,  
    null,  
    acceptableValues,  
    acceptableValues[1]);
```

showMessageDialog


- A showMessageDialog doesn't provide a field for input
- There are many other variations of JOptionPane



```
JOptionPane.showMessageDialog(  
    null,  
    "There's no input here, just a message",  
    "Title",  
    0);
```



Exercise 2

- Create a new project and add the `Input02.java` file to the project
- Experiment with the code and try to change ...
 - The message title
 - The message
 - Any default input text 
 - The dialog box's icon
- Parse, manipulate, and print any input

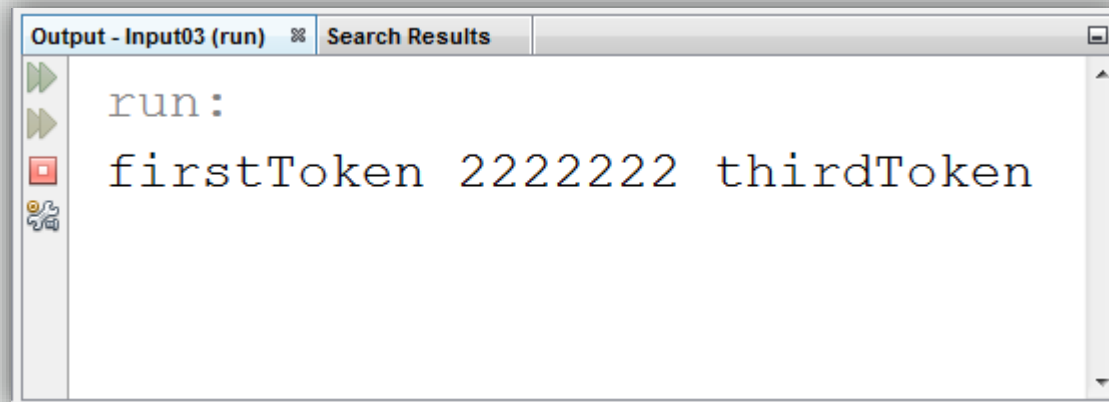
Getting Input with a Scanner

- A Scanner object opens a stream for collecting input:
 - System.in readies Scanner to collect input from the console
 - Type your input in your IDE output window
 - It's also possible to use Scanner without an IDE
- It's best practice to close the Scanner stream when you're finished

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    sc.close();  
} //end method main
```

Reading Input with a Scanner

- The Scanner searches for tokens
- Tokens are separated by a delimiter
 - The default delimiter is a space

A screenshot of an IDE's output window. The window has a title bar with "Output - Input03 (run)" and a search icon. Below the title bar is a tab labeled "Search Results". The main area of the window displays the output of a Java program. It shows "run:" on the first line and "firstToken 2222222 thirdToken" on the second line. The text is in a monospaced font. On the left side of the output area, there are four icons: a green play button, a green right-pointing arrow, a red square, and a magnifying glass over a document icon.

```
run:
firstToken 2222222 thirdToken
```

The Scanner Class

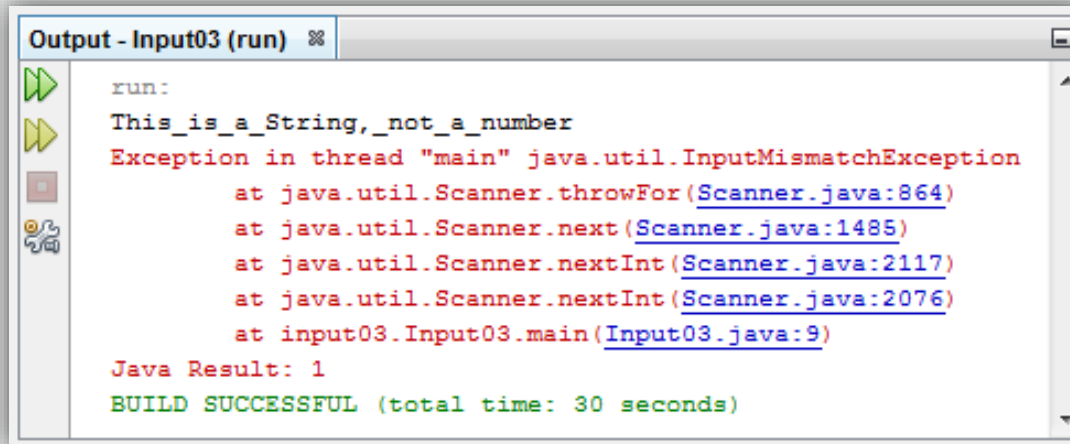
- Scanner, like any other class, has fields and methods
- A few useful Scanner methods ...
 - `nextInt()` reads the next token as an int
 - `nextDouble()` reads the next token as a double
 - `next()` reads the next token as a String

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int    x = sc.nextInt();  
    double y = sc.nextDouble();  
    String z = sc.next();  
    sc.close();  
} //end method main
```

Exercise 3

- Create a new project and add the `Input03.java` file to the project
- Create a Scanner:
 - Your IDE will complain
 - Follow your IDE's suggestion of importing `java.util.Scanner`
 - Remember to close the Scanner
- Use Scanner and `System.in` to write a program that ...
 - Finds and prints the sum of three integers entered by the user
- Try entering less than three tokens
- Try entering a token that can't be parsed as an int

Exceptions: InputMismatchException

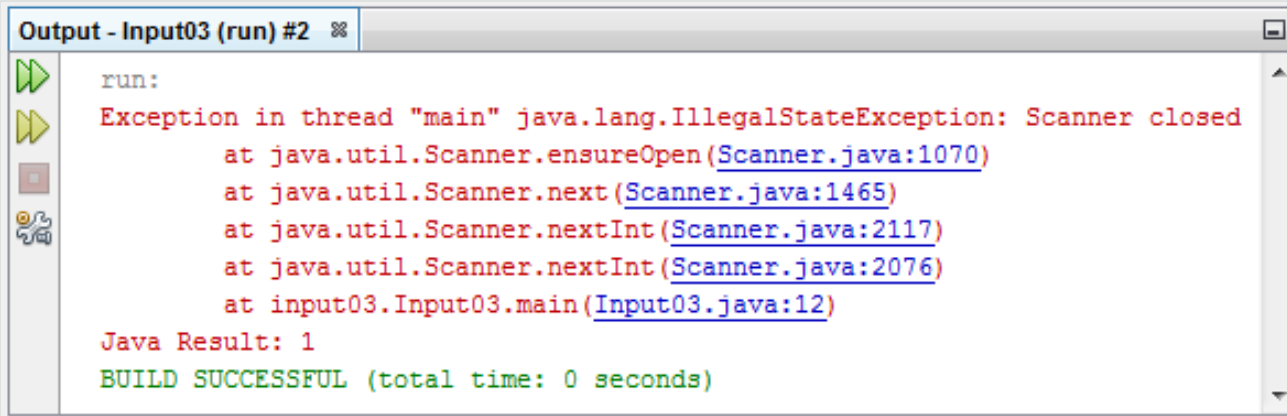
A screenshot of an IDE's output window titled "Output - Input03 (run)". The window shows the execution of a Java program. The output starts with "run:" followed by the input string "This_is_a_String,_not_a_number". Then, a red exception message is displayed: "Exception in thread \"main\" java.util.InputMismatchException". Below this, the stack trace is shown in red text, listing the following locations: "at java.util.Scanner.throwFor(Scanner.java:864)", "at java.util.Scanner.next(Scanner.java:1485)", "at java.util.Scanner.nextInt(Scanner.java:2117)", "at java.util.Scanner.nextInt(Scanner.java:2076)", and "at input03.Input03.main(Input03.java:9)". The output concludes with "Java Result: 1" and "BUILD SUCCESSFUL (total time: 30 seconds)".

```
run:
This_is_a_String,_not_a_number
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at input03.Input03.main(Input03.java:9)
Java Result: 1
BUILD SUCCESSFUL (total time: 30 seconds)
```

- Occurs because the input cannot be parsed as the expected type:

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println(sc.nextInt());
    sc.close();
} //end method main
```

Exceptions: IllegalStateException

A screenshot of an IDE's output window titled "Output - Input03 (run) #2". The window shows the following text: "run:", "Exception in thread \"main\" java.lang.IllegalStateException: Scanner closed", followed by a stack trace: "at java.util.Scanner.ensureOpen(Scanner.java:1070)", "at java.util.Scanner.next(Scanner.java:1465)", "at java.util.Scanner.nextInt(Scanner.java:2117)", "at java.util.Scanner.nextInt(Scanner.java:2076)", and "at input03.Input03.main(Input03.java:12)". Below the stack trace, it says "Java Result: 1" and "BUILD SUCCESSFUL (total time: 0 seconds)".

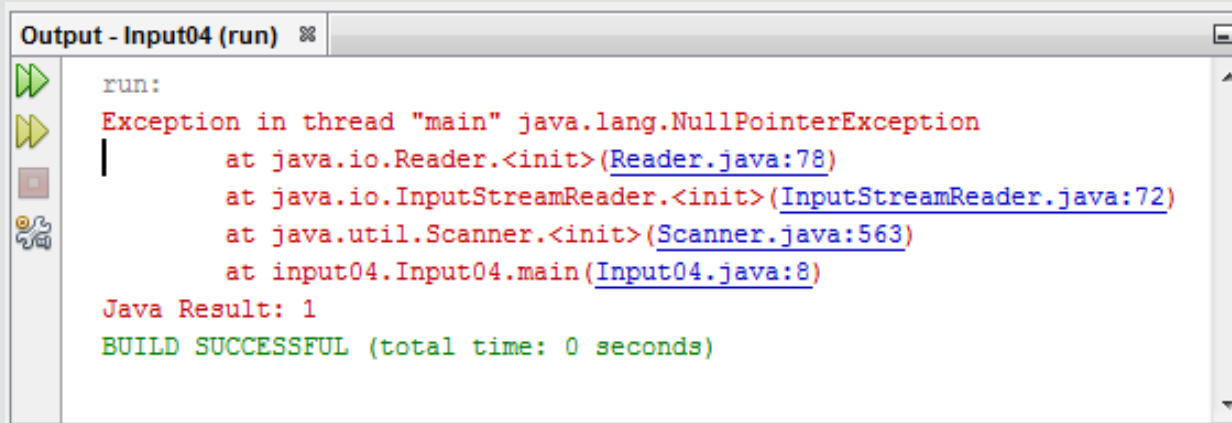
```
run:
Exception in thread "main" java.lang.IllegalStateException: Scanner closed
    at java.util.Scanner.ensureOpen(Scanner.java:1070)
    at java.util.Scanner.next(Scanner.java:1465)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at input03.Input03.main(Input03.java:12)

Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Occurs because the stream is accessed after it's been closed:

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    sc.close();
    System.out.println(sc.nextInt());
} //end method main
```

Exceptions: NullPointerException



```
Output - Input04 (run)
run:
Exception in thread "main" java.lang.NullPointerException
    at java.io.Reader.<init>(Reader.java:78)
    at java.io.InputStreamReader.<init>(InputStreamReader.java:72)
    at java.util.Scanner.<init>(Scanner.java:563)
    at input04.Input04.main(Input04.java:8)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Occurs because "fakeFile.txt" doesn't exist, it's also a common error to forget the .txt extension

```
public static void main(String[] args) {
    Scanner sc = new Scanner(
        Input04.class.getResourceAsStream("fakeFile.txt"));
    sc.close();
} //end method main
```

Remember the extension

Reading from a File

- Java offers several way to read files
- More useful Scanner methods include:
 - `nextLine()` advances this Scanner past the current line and returns the input that was skipped
 - `findInLine("StringToFind")` Attempts to find the next occurrence of a pattern constructed from the specified String, ignoring delimiters

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(  
        Input04.class.getResourceAsStream("fakeFile.txt"));  
    int x = sc.nextInt();  
    String entireLine = sc.nextLine();  
    sc.close();  
} //end method main
```


Exercise 4, Part 1

- Create a new project and add the `Input04.java` file to the project
- Run the code and examine the output
- Read through each next line until you find "BlueBumper"
- The two numbers following "**BlueBumper**" are the object's `xPosition` and `yPosition`. Store these coordinates as integers and print them
- Examine `input04text.txt`, if necessary

Exercise 4, Part 2

- Examine `Level05.txt` if you're curious:
 - This is how level data is stored for Java Puzzle Ball
 - Reading and parsing level data is slightly more complicated than what you've done in this exercise
 - But if you finished this exercise, you're close to understanding how it's done

Summary

- In this lesson, you should have learned how to:
 - Understand user input
 - Create a JOptionPane to collect user input
 - Use a Scanner to collect input from the console
 - Use a Scanner to collect input from a file
 - Understand how a Scanner handles tokens and delimiters





ORACLE

Academy

