Object Oriented Programming

Text Input and String Processing

Dr. Seán Russell
sean.russell@ucd.ie

School of Computer Science,
University College Dublin

September XX, 2019

Learning outcomes

After this lecture and the related practical students should...

- understand the concept of input and output in programming
- be able to use the Scanner to read input from the user
- be able to give the user many attempts to enter the correct input
- be able to find and extract information from a string

Table of Contents

Text Input in Java

- Reading text input from the user is a little different in Java than it is in C
- Instead of just calling a function to read values, we have to use an object to perform the steps for us
- There are many different ways to do this provided by different classes in the Java libraries
- We will be using a class called `Scanner` from the package `java.util`.

The Scanner

- The scanner class can be used to create multiple scanner objects that are useful for breaking up text and converting it into useful formats for us
- The Scanner is designed to be reusable in many different situations, this means that we have to specify how it is to be used when we create it
- Additionally we will also have to import the class any time we want to use it
- E.g. `import java.util.Scanner;`

Constructing a Scanner

- Scanners can be used to read many types of input, so we must tell it **what** to read as a parameter to the constructor
- We will focus on the constructor that allows us to read from the command prompt

Scanner Constructor

`public Scanner(InputStream source)`

Constructs a new Scanner that produces values scanned from the specified input stream. Bytes from the stream are converted into characters using the underlying platform's default charset.

Parameters: source - An input stream to be scanned

Constructing a Scanner

- The constructor requires a single parameter, that is of the type `InputStream`
- We will learn more about what an input stream is an how to use them later when we study file IO
- For now we will always use the same parameter that is created for us but the JVM
- We will use an input stream created for us called `System.in`
- `Scanner input = new Scanner(System.in);`

Table of Contents

Using a Scanner

- Reading information using a scanner is different than using `scanf`
- When using a scanner we can only read information one piece at a time and in the order that it was entered
- The information that we read can vary from a single character up to an entire line of text
- The scanner class contains many methods that can be used to read information and each method reads a specific type of data

Reading Order

- A scanner will always read a group of characters
- These groups are known as **tokens**
- Tokens are separated by whitespace characters such as new line, space or tab characters
- Because the scanner reads tokens in order from first to last, all of the methods for reading information are named **nextXXX**
- This refers to the next token to be read, for example the method `nextInt` will try and read the next token as an int and `nextDouble` tries to read the next token as a double

Reading Order

- Each of the methods will read a token and often try to interpret it in some way, this may cause an error if we are not careful about what we type
- If we try to read a double using the method `nextDouble`, but the user types "abc", then the scanner will attempt to convert "abc" into a double
- This will cause our programs to stop working unless we are very careful

Common Scanner Methods

Method	type	Explanation
nextInt	int	Reads the next token as an int
nextDouble	double	Reads the next token as a double
nextLong	long	Reads the next token as a long
nextFloat	float	Reads the next token as a float
next	String	Reads the next token as a String
nextLine	String	Reads all of the tokens remaining on this line

Reading Order Example

- Lets say for example we have a number of lines of data containing the following information about a number of students;
  - An integer representing their grade in an assignment e.g. 100
  - Some characters representing their letter grade in an exam e.g. B+
  - A real number representing their grade in a programming exam e.g. 88.6
  - The students name e.g. Sean Russell
- All of this information is contained on a single line, what operations and in what order are required to read this information into our program in a way that is usable?

Reading Order Example

1. First we need to read the assignment grade token, to do this we call the `nextInt` method
2. Next we need to read the letter grade token, this is done by calling the `next` method
3. We also need to read in the programming exam token, this is done by calling the `nextDouble` method
4. Finally we need to read in the students name, as a name will most likely be multiple tokens we will call the `nextLine` method to read in all of the tokens remaining on the line.

Reading an int API

`public int nextInt()`

Scans the next token of the input as an int.
An invocation of this method of the form nextInt() behaves in exactly the same way as the invocation nextInt(radix), where radix is the default radix of this scanner.

Returns

the int scanned from the input

Reading an int Example

```
1  import java.util.Scanner;
2
3  public class ScannerExamples {
4    public static void main(String[] args) {
5      Scanner in = new Scanner(System.in);
6
7      System.out.println("Enter an int: ");
8
9      int x = in.nextInt();
10
11     System.out.println("You entered "+ x);
12   }
13 }
```

Example Explained

- Here we can see that there are a number of steps;
  1. On line 1 the scanner class is imported so we can use it
  2. On line 5 the scanner named in is constructed
  3. On line 7 a message is printed telling the user what to do
  4. On line 9 we declare a variable named x and use it to store the result of calling the method nextInt on the object in
  5. On line 11 we print out the number

Reading a double API

`public double nextDouble()`

Scans the next token of the input as a double. This method will throw `InputMismatchException` if the next token cannot be translated into a valid double value. If the translation is successful, the scanner advances past the input that matched.

Returns

the double scanned from the input

Reading a double Example

```
1  public class ScannerExamples {
2    public static void main(String[] args) {
3      Scanner in = new Scanner(System.in);
4
5      System.out.println("Enter a double: ");
6
7      double x = in.nextDouble();
8
9      System.out.println("You entered "+ x);
10   }
11 }
```

Reading an word API

`public String next()`

Finds and returns the next complete token from this scanner. A complete token is preceded and followed by input that matches the delimiter pattern. This method may block while waiting for input to scan, even if a previous invocation of hasNext() returned true.

Returns

the next token

Reading a word Example

```
1  public class ScannerExamples {
2    public static void main(String[] args) {
3      Scanner in = new Scanner(System.in);
4
5      System.out.println("Enter a String: ");
6
7      String x = in.next();
8
9    System.out.println("You entered <"+x+">");
10   }
11 }
```

Reading an line API

`public String nextLine()`

Advances this scanner past the current line and returns the input that was skipped. This method returns the rest of the current line, excluding any line separator at the end. The position is set to the beginning of the next line. Since this method continues to search through the input looking for a line separator, it may buffer all of the input searching for the line to skip if no line separators are present.

Returns

the line that was skipped

Reading a Line Example

```
1  public class ScannerExamples {
2    public static void main(String[] args) {
3      Scanner in = new Scanner(System.in);
4
5      System.out.println("Enter a String: ");
6
7      String x = in.nextLine();
8
9    System.out.println("You entered <"+x+">");
10   }
11 }
```

Reading Order Example

- The scanner itself only needs to be created a single time, but it can be used many times to read information from the user
- For example, the code to read the student information described above would be:

Student Information Implemented

```
1 Scanner in = new Scanner(System.in);
2 System.out.println("Enter the students information");
3 int assignmentGrade = in.nextInt();
4 String examGrade = in.next();
5 double pExamResult = in.nextDouble();
6 String name = in.nextLine();
7 System.out.println(name + " got results " + examGrade
    + ", " + pExamResult + " and " + pExamResult);
```

Table of Contents

User Prompts

- In the API excerpt for the next method, it notes that the method may **block** while waiting for input
- This means that the scanner will wait until the user has typed something
- When this is happening, it can often be confused for the program freezing or entering an infinite loop
- It is always important that when asking the user for input that there is a message to tell them what input is expected
- These messages should be relatively specific, so that the user has a good idea of the type of data that is to be entered

Table of Contents

Mismatch Errors

- If we are expecting an integer, but anything else is entered, this will cause a mismatch error and probably crash our program
- This can only happen if we are trying to read a specific type of data, such as an int, long or double
- There can **never** be a mismatch error when we are trying to read a string
- This is because, the rules allow any type of character to be stored in a String

Mismatch Errors

Example of mismatch error output

```
1 Enter an int:
2 abc
3 Exception in thread "main"
      java.util.InputMismatchException
4   at java.util.Scanner.throwFor(Unknown Source)
5   at java.util.Scanner.next(Unknown Source)
6   at java.util.Scanner.nextInt(Unknown Source)
7   at java.util.Scanner.nextInt(Unknown Source)
8   at ScannerExamples.main(ScannerExamples.java:9)
```

- This is an example of the output you will see when a mismatch error happens

Mismatch Errors

- There are two possible solutions to this problems
  1. We write our programs more robustly to recover from these errors
  2. We check that the next token is the type we correct and ignore it if it is not
- The first solution we will learn about when we study exceptions
- The second solution we will look at now

What is Coming Next?

- The Scanner class provides methods that allow us to check if the next token can be converted to the type we want
- Before we read an integer, we ask the scanner if the next item is an integer
- Before we read a double, we ask if the next item is a double
- If it is, everything is OK
- If it is not we can read the token **as a string** and try again

Is the Next Token an int

hasNextInt

`public boolean hasNextInt()`

Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the `nextInt()` method. The scanner does not advance past any intput.

Returns

`true` if and only if the scanner's next token is a valid int value

Is the Next Token an int

```
1  public static void main(String[] args) {
2     Scanner in = new Scanner(System.in);
3
4     System.out.println("Enter an int: ");
5     if (in.hasNextInt()) {
6        int x = in.nextInt();
7        System.out.println("You entered " + x);
8     } else {
9        System.out.println("Not an int");
10    }
11 }
```

Table of Contents

More Robust Checking

- The previous example is not a very good solution, it does not offer the user a chance to try again
- Ideally, we would want to print an error message to the user and then have them attempt to enter the information again
- But how do we get rid of the token they entered?
- We call the `next` method
- What happens if they get it wrong again?
- We should give them another chance, and another, and another....

More Attempts

- Because the user may make many mistakes when entering input, we will want to give them many attempts to get it correct
- The easiest way to do this is using a loop
- The loop should execute when the user is incorrect
- If we ask the user to enter an int we would add a loop that removes a token if it is **not** an integer
- If we are using a scanner named `scan` the code `scan.hasNextInt()` will return true if the token is an integer
- This means we use `!scan.hasNextInt()` instead (! is the boolean not operator)

More Attempts Explained

- What this means is that when we want to read an integer value, the loop will execute as long as the token is not an integer value
- Inside the loop then we need to remove the token from the scanner by calling the `next` method
- The only way for the loop to end is if the user enters an integer value
- Once the loop is completed, we can read the value we want
- Because of the position of the loop, just before, we know that the token we are about to read must be an integer

More Attempts

```
1  public static void main(String[] args) {
2     Scanner in = new Scanner(System.in);
3
4     System.out.println("Enter an int: ");
5     while (!in.hasNextInt()) {
6     String temp = in.next();
7       System.out.println("You entered <"+
       temp+"> try again")
8     }
9     int x = in.nextInt();
10    System.out.println("You entered " + x);
11 }
```

Table of Contents

Verifying Input

- Sometimes it is necessary to make sure that **values** are within the correct range
- If we were reading an assignment grade from the user, we should check that it was between 0 and 100
- This is more complicated that simply checking that the type is correct because we must also check that the value is correct
- This will require **two** loops
- The code we have written already is correct, but it needs to be inside another loop that makes sure that the value is within the correct range
- We must have all of the code repeated as long as the value is not in the correct range

Verifying Input Example

```
1  System.out.println("Enter an integer grade: ");
2  int grade = -1;
3  while (grade < 0 || grade > 100) {
4    while (!in.hasNextInt()) {
5      String temp = in.next();
6      System.out.println("You entered <" + temp + "> try
       again");
7    }
8    grade = in.nextInt();
9    System.out.println("You entered " + grade);
10   if(grade < 0 || grade > 100){
11     System.out.println("The value must be between 0
       and 100, try again");
12   }
13 }
```

Table of Contents

Dialogs

- When developing graphical applications, generally the user is not expected to type into the console
- When creating these types of applications, we may want to display pieces of information to the user or read small pieces of information from the user
- We can use dialog boxes to do this
- There are a number of class methods in the class `JOptionPane` in the package `java.swing` that can be used to easily create dialogs
- We will very briefly study some of these methods
- There are different types, but we will only look at the message dialog and the input dialog

Message Dialog

- A message dialog is simply, a small box containing a message for the user
- The box will stay on the screen until the user clicks the OK button

`showMessageDialog`

`public static void showMessageDialog(Component parentComponent, Object message)`
Brings up an information-message dialog titled "Message".
Parameters:
parentComponent - determines the Frame in which the dialog is displayed; if null, or if the parentComponent has no Frame, a default Frame is used

message - the Object to display

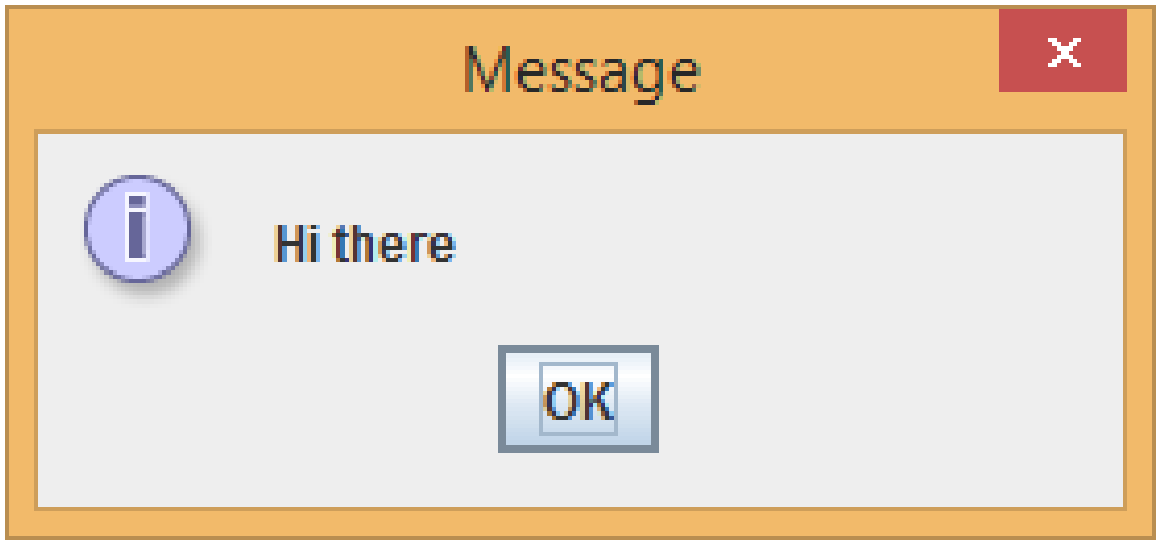
Message Dialog

- The method requires two parameters, the first is a frame and the second is the message
- We can just pass the value `null` for the first parameter
- For the second parameter, we pass the message that we want to be displayed as a String
- The `showMessageDialog` method is a class method, this means that we do not need to create an instance of the `JOptionPane` class
- It is important to remember that the execution of the program will wait until the user has clicked OK

Message Dialog Result

- For example, if we wanted to show the message "Hi there", we would use:

```
JOptionPane.showMessageDialog(null, "Hi there");
```



Input Dialog

- An input dialog is similar to a message dialog, except that the user can enter some information
- An input dialog contains a message to the user, a box for them to enter some text and two buttons OK and Cancel

showInputDialog

`public static String showInputDialog(Component parentComponent, Object message)`
Shows a question-message dialog requesting input from the user parented to parentComponent. The dialog is displayed on top of the Component's frame, and is usually positioned below the Component.
Parameters:
parentComponent - the parent Component for the dialog

message - the Object to display

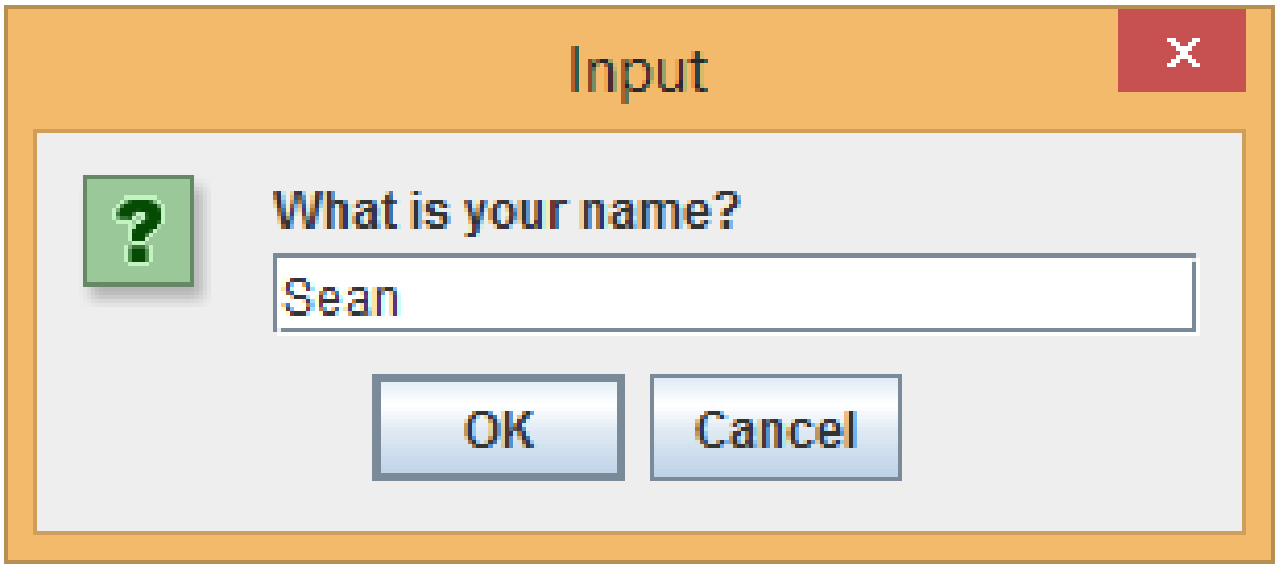
Input Dialog Example

- We can see that the method requires two parameters again, these are pretty much exactly the same as before
- The difference here is the return type of the method. This method returns a String as a result
- If the user clicks OK, the content of the text box is returned as a String, and if the user clicks cancel then null is returned
- Again, because this is a class method, we can call it directly without creating a JOptionPane object

Input Dialog Example

- For example, if we wanted to ask the user their name, we would use:

```
String name = JOptionPane.showInputDialog(null,
    "What is your name?");
```



Blocking

- The input dialog causes the code to stop executing until the OK or Cancel button has been clicked
- The input dialog can only be used to read a string from the user
- So if we wanted to read the users age, we would need to convert the string into an integer after it was read from the input dialog
- This can be done using methods such as `parseInt` from the `Integer` class

```
1 String ageString = JOptionPane.showInputDialog(null,
    "What is your age?");
2 int age = Integer.parseInt(ageString);
3 System.out.println(age);
```

Table of Contents

String Processing

- A Scanner is good for read information piece by piece
- But what if we already have all the information in one string?
- There are 2 simple solutions
  1. Create a scanner to consume the string
  2. Use the method in the string class to get the information

1. Using a Scanner

- The Scanner class has many constructors
- One of them takes a String as a parameter
- We can create a new scanner and pass it the string and use the same methods as above to get the information

2. Using String Methods

- There are many methods in the String class
- Some of the most useful ones are;
  - `public int indexOf(String s)`
  - `public String[] split(String s)`
  - `public String subString(int b, int e)`
  - `public char charAt(int index)`
  - `public boolean equalsIgnoreCase(String s)`

Searching a String

- `indexOf` returns the index within this a of the first occurrence of a specified substring
- E.g. `"hello".indexOf("l")` would return 2
- We can use this to search for the information we need

Splitting a String

- `split` breaks a single string into an array of strings
- The strings are split every time a particular string is found
- E.g. `"Sean is great".split(" ")` would return `{ "Sean", "is", "great"}`
- E.g. `"hello".split("l")` would return `{ "he", "o"}`
- Note the string we use to split is removed from the result

Substrings

- `substring` allows you to get just a piece of a string indicated by two numbers (start and end index)
- E.g. `"Sean is great".substring(4,8)` would return `" is "`
- E.g. `"hello".substring(0,2)` would return `"he"`
- All of the characters starting from the first index and up to but not including the second index are returned

Selecting a Single Character

- We can also get a single character from a string using `chatAt`
- `"Sean".charAt(2)` would return `'a'`
- Remember indexes start at 0 and go to length - 1

Are Two Strings the Same

- We can use `equals` or `equalsIgnoreCase` to compare 2 strings
- `"Sean".equals("sean")` will return `false`
- `"Sean".equalsIgnoreCase("sean")` will return `true`
- This is necessary because == compares the memory addresses of two strings
  - This can work sometimes...