

## Java Classes

- Java programs are made up of *classes*
- So far, we have *written* only the “main program” class
- But we have *used* several other classes:
  - Scanner
  - String
  - FileIOHelper
  - Student

## Two Kinds of Classes

- Classes can either provide a new *type* with corresponding *methods* to manipulate the values of that type, e.g.,
  - Scanner
  - String
  - Student
- Or they simply can provide a bunch of methods that perform some useful task, e.g.,
  - FileIOHelper

## Two Kinds of Methods

- You may have noticed a difference in the syntax of method invocation
- We write
  - `str.length()`
  - `in.nextInt()`
- But we also write
  - `FileIOHelper.numberOfStudents(fn)`
  - `FileIOHelper.getNextStudent()`

## Sending Output to a (Text) File

```
import java.util.Scanner;
import java.io.*;
public class TextFileOutputDemol
{
    public static void main(String[] args) throws IOException
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        PrintWriter outputFile = new PrintWriter("out.txt");
        outputFile.println(name);
        outputFile.close();
    }
}
```

## Creating a File for Text Output

- We need a new class:
  - `PrintWriter`
- To gain access to it we need to import it with:
  - `import java.io.*;`
- To create and open the file:
  - `PrintWriter outFile =  
new PrintWriter(fileName);`

## Writing Output to a Text File

- The `PrintWriter` class has output methods you are already familiar with:
  - `print`
  - `println`
- Instead of calling `System.out.print(...)` or `System.out.println(...)`, you invoke:
  - `outFile.print(...)`
  - `outFile.println(...)`

## Closing the File

- It is important to remember to explicitly close a file you are done writing to:
  - `outFile.close()`

## Your Turn

- Write a program that asks the user for a file name and then outputs the numbers 1 through 10, one per line, to the corresponding file.

# Output Numbers

## What If an Error Occurs?

- What could go wrong when we try to create a file?
- We might not be able to create it!
- In that case an *exception* (error) is generated/raised by the program.
- What can we do about it?

# Exceptions

- There are a variety of errors that can occur in a Java program that result in an exception being raised
- Java forces us to deal with some of them by
  - either explicitly stating that such an exception might occur in our program
  - or by catching the exception and dealing with it in our code

## Stating That Exception Might Occur

```
import java.util.Scanner;
import java.io.*;
public class TextFileOutputDemo1
{
    public static void main(String[] args) throws IOException
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        PrintWriter outputFile = new PrintWriter("out.txt");
        outputFile.println(name);
        outputFile.close();
    }
}
```

## Dealing With Exception If It Occurs

```
import java.util.Scanner;
import java.io.*;
public class TextFileOutputDemo2
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        try {
            PrintWriter outputFile = new PrintWriter("out.txt");
            outputFile.println(name);
            outputFile.close();
        } catch (IOException e) {
            System.out.println("Error opening the file out.txt");
        }
    }
}
```

## Try-Catch

- You can think of it as a control structure in that it affects the flow of execution of the program when an exception occurs.

```
try {
    // statements possibly raising exception
}
catch (exception declaration)
{
    // statements dealing with exception
}
```

## Reading Input from a (Text) File

```
import java.util.Scanner;
import java.io.*;

public class TextFileInputDemo1
{
    public static void main(String[] args)
        throws IOException
    {
        File file = new File("in.txt");
        Scanner inputFile = new Scanner(file);
        String line = inputFile.nextLine();
        inputFile.close();
        System.out.println(line);
    }
}
```

## Opening a File for Text Input

- We need a new class:
  - File
- To gain access to it we need to import it with:
  - `import java.io.*;`
- To open an existing file:
  - `File file = new File(fileName);`  
`Scanner inFile = new Scanner(file);`

## Reading Input from a Text File

- The Scanner class has input methods you are already familiar with:
  - `nextLine`, `nextInt`, `nextDouble`, etc.
- Now we also need a way to check when we reach the end of the file:
  - `boolean hasNext()`
  - It takes no parameters and returns `true` if there is more data to read, and `false` otherwise

## Closing the File

- It is important to remember to explicitly close a file you are done reading from:
  - `inFile.close()`

## Your Turn

- Complete the following program that asks the user for a file name, opens the file, reads one line at a time and outputs the line to the screen, until it reaches the end of the file.
- Make sure you catch the possible IOException in a try-catch statement.

## Read File

```
import java.util.Scanner;
import java.io.*;
public class ReadFile
{
    public static void main(String[] args)
    {

    }
}
```

## Your Turn

- Complete the following program that asks the user for a file name, opens the file, reads a bunch of integer values, one per line, until it reaches the end of the file. The program computes and outputs the average of the integers to the screen.
- Make sure you catch the possible IOException in a try-catch statement.

## Average Numbers

```
import java.util.Scanner;
import java.io.*;

public class AverageNumbers
{
    public static void main(String[] args)
    {

    }
}
```

## Passing Arguments To Main

```
public static void main(String[] args) {...}
```

- The formal parameter `args` is an array of `String`
- How do we pass actual arguments to method `main`?

## Command Line Arguments

- When you invoke a Java program from a terminal, anything you type after the name of the program is passed to the `main` method as the array of `String`, e.g.,

```
java SomeProgram arg1 arg2 arg3
```

# Example

```
public class CommandLineArgsTest
{
    public static void main(String[] args)
    {
        System.out.println(
            "Number of arguments: " + args.length);
        for (int i = 0; i < args.length; i++)
        {
            System.out.println(args[i]);
        }
    }
}
```

# Your Turn

- What will the following program invocations output to the screen?
  - `java CommandLineArgsTest this is a test`
  - `java CommandLineArgsTest 1 2 3 4 5`
  - `java CommandLineArgsTest to b || ! 2 be`
  - `java CommandLineArgsTest`