

Outline



Reading Files

Writing Files

Methods

Scanner

- In our examples Scanner was reading input from System.in
- Example:

```
Scanner scan = new Scanner (System.in) ;  
int i = scan.nextInt() ;
```

This example reads a single int from System.in

We will now change the parameter, **System.in**

Reading File: The **File** Class

- How are we going to get a reference to a file on our disk?

```
// Need to import to use File class  
import java.io.*;
```

```
String myFilename = "example.txt";  
File myfile = new File(myFilename);
```

The Scanner class

Pass **the File** object as a parameter to the Scanner class

```
String myFilename = "example.txt ";  
File myFile = new File( myFilename );  
Scanner input = new Scanner( myFile );
```

Compiler Error with Files

- Would the following program compile?

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

        String myFilename = "data.txt";
        File myFile = new File(myFilename);
        Scanner input = new Scanner(myFile);

        String text = input.next();
        System.out.println(text);

        input.close();
    }
}
```

Compiler Error with Files

- The following program does not compile:

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

        String myFilename = "data.txt";
        File myFile = new File(myFilename);
        Scanner input = new Scanner(myFile);

        String text = input.next();
        System.out.println(text);

        input.close();
    }
}
```

- The following error occurs:

```
ReadFile.java:6: unreported exception java.io.FileNotFoundException;
must be caught or declared to be thrown
        Scanner input = new Scanner(new File("data.txt"));
                                ^
```

Exceptions

- **Exception:** An object representing a runtime error.
 - dividing an integer by 0
 - calling `charAt` on a `String` with a large index
 - trying to read the wrong type of value from a `Scanner`
 - trying to read a file that does not exist
- A program with an error "*throws*" an exception.
- It is also possible to "*catch*" (handle or fix) an exception.



The throws Clause

- **throws clause:** Keywords on a method's header that state that it may generate an exception.
- Like saying, "I hereby announce that this method might throw an exception, and I accept the consequences if it happens."

- **Syntax:**

```
public static type name (params) throws type {
```

- **Example:**

```
public class ReadFile {  
    public static void main(String[] args)  
        throws IOException{
```


Reading an Entire File

- Suppose we want our program to process the entire file; for instance find the sum of all numbers stored in a file regardless of the number of values stored in a file.
- How can we achieve this?

Testing for Valid Input: hasNext Methods

- Scanner hasNext Methods are useful to see what input is coming, and to avoid crashes

Method	Description
<code>boolean hasNext ()</code>	returns true if there are any more tokens of input to read <i>(always true for console input)</i>
<code>boolean hasNextInt ()</code>	returns true if there is a next token and it can be read as an int
<code>boolean hasNextDouble ()</code>	returns true if there is a next token and it can be read as a double
<code>boolean hasNextLine ()</code>	returns true if there is a next line of input

- **These methods do not consume input;** they just give information about the next token.

How to iterate until the end of file?

EchoFile2.java

EchoFileNumbers.txt

- Find the number of lines in a given input file
- See [FindNumberOfLines.java](#)

File Reading Example

Find the number of space (' '), comma (',') and dot ('.') in each line and in the file and report it to the user.

- Check out [CountPunctuationSpace.java](#)

Example

- Check this one at home: [SearchFile.java](#)

Outline

Reading Files



Writing Files

Methods

Writing Text Files

- We have to create `PrintWriter` objects for writing text to any file using `print`, `println` and `printf` methods.
- The `close()` method of the `PrintWriter` class must be used to close the file. If this method is not invoked the data may not be saved properly in the file.

Writing Text Files

Write a Java program that generates 10 random numbers per line for 10 lines and stores these numbers in a text file.

- See [TestData.java](#)

```

//*****
//  TestData.java          Author: Lewis/Loftus
//
//  Demonstrates I/O exceptions and the use of a character file
//  output stream.
//*****

import java.util.Random;
import java.io.*;

public class TestData
{
    //-----
    //  Creates a file of test data that consists of ten lines each
    //  containing ten integer values in the range 10 to 99.
    //-----
    public static void main (String[] args) throws IOException
    {
        final int MAX = 10;

        int value;
        String file = "ourtest.dat";

        Random rand = new Random();

```

continue

continue

```
PrintWriter outFile = new PrintWriter (file);

for (int line=1; line <= MAX; line++)
{
    for (int num=1; num <= MAX; num++)
    {
        value = rand.nextInt (90) + 10;
        outFile.print (value + "  ");
    }
    outFile.println ();
}

outFile.close();
System.out.println ("Output file has been created: " + file);
}
}
```

Output

Output file has been created: test.dat

continue

```
FileV  
PrintWriter outFile = new PrintWriter (fw);  
  
for (int line=1; line <= MAX; line++)  
{
```

Sample ourtest.dat File

77	46	24	67	45	37	32	40	39
10								
90	91	71	64	82	80	68	18	83
89								
25	80	45	75	74	40	15	90	79
59								
44	43	95	85	93	61	15	20	52
86								
60	85	18	73	56	41	35	67	21
42								
93	25	89	47	13	27	51	94	76
13								
33	25	48	42	27	24	88	18	32
17								
71	10	90	88	60	19	89	54	21

File Read and Write

Modify the `CountPunctuationSpace.java` such that the output is written on file

- Check out [CountPunctuationSpace2.java](#)

Outline

Reading Files

Writing Files

 **Methods**

We used many methods up to now

`int lastIndexOf(char ch) or lastIndexOf(String str)`
Returns the index of the last match of the argument, or `-1` if none exists.

`boolean equalsIgnoreCase(String str)`
Returns `true` if this string and `str` are the same, ignoring differences in

`boolean startsWith(String str)`
Returns `true` if this string starts with `str`.

`boolean endsWith(String str)`
Returns `true` if this string starts with `str`.

`String replace(char c1, char c2)`
Returns a copy of this string with all instances of `c1` replaced by `c2`.

`String trim()`
Returns a copy of this string with leading and trailing whitespace

`String toLowerCase()`
Returns a copy of this string with all uppercase characters changed to

`String toUpperCase()`
Returns a copy of this string with all lowercase characters changed to
uppercase

Useful Methods in the `Math` Class

<code>Math.abs(x)</code>	Returns the absolute value of x
<code>Math.min(x, y)</code>	Returns the smaller of x and y
<code>Math.max(x, y)</code>	Returns the larger of x and y
<code>Math.sqrt(x)</code>	Returns the square root of x
<code>Math.log(x)</code>	Returns the natural logarithm of x ($\log_e x$)
<code>Math.exp(x)</code>	Returns the inverse logarithm of x (e^x)
<code>Math.pow(x, y)</code>	Returns the value of x raised to the y power (x^y)
<code>Math.sin(theta)</code>	Returns the sine of $theta$, measured in radians
<code>Math.cos(theta)</code>	Returns the cosine of $theta$
<code>Math.tan(theta)</code>	Returns the tangent of $theta$
<code>Math.asin(x)</code>	Returns the angle whose sine is x
<code>Math.acos(x)</code>	Returns the angle whose cosine is x
<code>Math.atan(x)</code>	Returns the angle whose tangent is x
<code>Math.toRadians(degrees)</code>	Converts an angle from degrees to radians
<code>Math.toDegrees(radians)</code>	Converts an angle from radians to degrees

Divide and Conquer

- Break large programs into a series of smaller methods:
 - Helps manage complexity
 - Makes it easier to build large programs
 - Makes is easier to debug programs
 - Reusability

Methods

- Local variables
 - Declared in method declaration
- Parameters
 - Communicates information into the methods
- Return value
 - Communicates information to the outside of the method

Methods

Write a java method, maximum instead of using
`Math.max(x,y)`

See [MyMax.java](#)

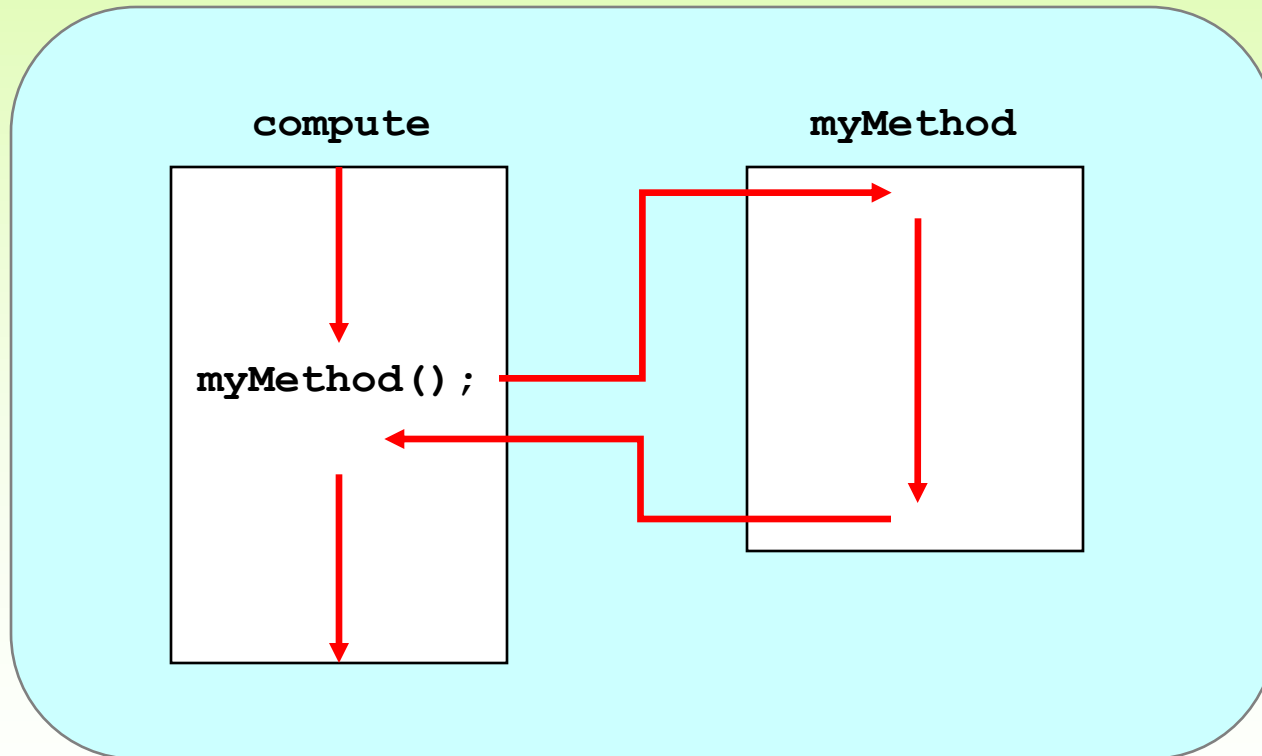
Now we'll learn how to write such methods.

Method Declarations

- *A method declaration specifies the code that will be executed when the method is invoked (called)*
- When a method is invoked, the flow of control jumps to the method and executes its code
- When complete, the flow returns to the place where the method was called and continues
- The invocation may or may not return a value, depending on how the method is defined

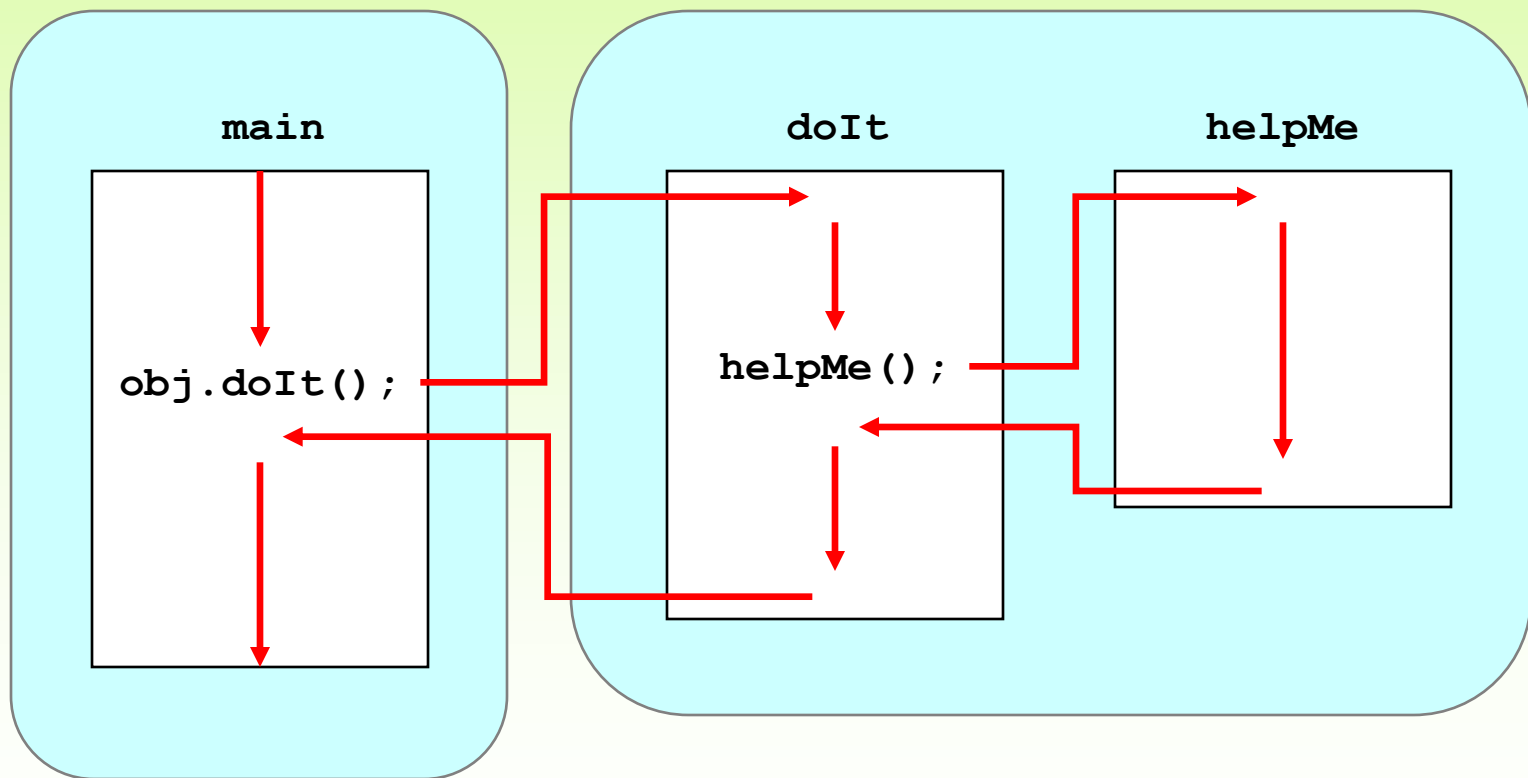
Control Flow on Invoking a Method

- If the called method is in the same class, only the method name is needed



Control Flow on Invoking a Method

- The called method is often part of another class or object



Methods Header

- What information can you learn about a method from its header?

```
public static void calc (int num1, String message)
```

Method Body

- The method header is followed by the *method body*

```
public static char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = '*';


    if (sum >= 0 && sum < message.length() )
        result = message.charAt (sum);

    return result;
}
```

sum **and** result
are local data

**They are created
each time the
method is called, and
are destroyed when
it finishes executing**

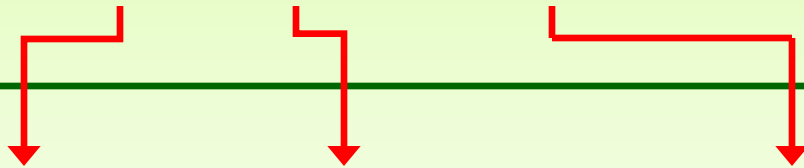
**The return expression
must be consistent with
the return type**



Parameters

- When a method is called, the *actual parameters* in the invocation are copied into the *formal parameters* in the method header

```
ch = obj.calc (25, count, "Hello");
```



```
char calc (int num1, int num2, String message)
{
.
.
.
}
```

Parameter Data Types

- You can pass as many parameters as you like.
- To pass more than one parameter, you need to separate the parameters with commas.

```
public static int maximum (int x, int y)
{
    /*body*/
}
```

No Parameters

- You can also have a method that accepts no parameters. The parameter list is empty:

```
public static int rollDie ()
```

```
public static void printIntro ()
```

The **return** Statement

- The *return type* of a method indicates the type of value that the method sends back to the calling location
- A method that does not return a value has a `void` return type
- A *return statement* specifies the value that will be returned
`return expression;`
- **Its expression must conform to the return type**

Return Value Types

- You **may have multiple return statement expressions** (through if, nested if, switch etc)
- You can only return **at most one value** from a method.

Modify MyMax.java program to find the maximum among 3 numbers.

- See [MyMax3.java](#)

- Exercise: Modify the [Palindrome.java](#) such that it includes a `isPalindrome` method
- See [Palindrome2.java](#)

Returning `void`

- `void`: means nothing
- A method that returns `void` returns nothing.

```
public static void printIntro (int n)
```
- `void` methods can optionally use a return statement with no value:
 - `return;`
 - There is no need for the optional return statement. But occasionally you might need it to force the program for an early exit from the method.

Methods That Don't Return Anything

```
public static void printIntro() {  
    System.out.println("Welcome to CS114");  
    System.out.println("It's the best part of my day :P");  
}
```

Local Data

- Local variables can be declared inside a method
- The formal parameters of a method create *automatic local variables* when the method is invoked
- When the method finishes, all local variables are destroyed (including the formal parameters)

The static Modifier

- We declare static using the `static` modifier
- Static method is one that can be **invoked through its class name as opposed to an object of the class**
- Static methods are sometimes called *class methods*
- For example, the methods of the `Math` class are static:

```
result = Math.sqrt(25)
```

Static Methods

- There is no need to instantiate an object of the class in order to invoke the method.

`ClassName.methodName(args)`
↓ ↙ ↖
`Math.sqrt(28);`

- All the methods of the Math class are static.

Static Methods

```
public static int max(int val1, int val2) {  
  
    if (val1 > val2) {  
        return val1;  
    }  
    else {  
        return val2;  
    }  
}
```

Let's say this belongs to a class named Calculator

```
int myMax = Calculator.max(9, 2);
```

Static Methods

```
public class Helper
{
    public static int cube (int num)
    {
        return num * num * num;
    }
}
```

- Because it is declared as static, the cube method can be invoked through the class name:

```
value = Helper.cube(4);
```

- By convention visibility modifiers come first (private, or public determines visibility)

Main Method

- Recall that the `main` method is static – it is invoked by the Java interpreter without creating an object

```
public static void main (String[] args )
```

Exercise

- Modify the following code according to the comments
- See [Circle.java](#)