Files

GEEN163

“It's hardware that makes a machine fast. It's software that makes a fast machine slow.”

Craig Bruce

Reading Files

- Java programs can read input from files as well as from the keyboard
- The Scanner method can be used to read files
- There are also many other methods that can be used to read a file

File Class

- The java.io.File class defines a file in Java
- The simple constructor for java.io.File takes a String containing the filename

```
String filename = "mydata.txt";
java.io.File  elephant = new java.io.File( filename );
```

- The filename can contain a full path, such as

```
"C:\\Users\\KenW\\Documents\\mydata.txt";
```

Escape Characters

- Remember that special characters in Java are represented by a backslash (\ ) and a letter
- The backslash and letter form one character

\t	tab
\n	new line (return)
\"	quote
\\	backslash

Scanner with Files

- When you create an object of the Scanner class, you can specify a File object instead of System.in

```
java.io.File  elephant = new java.io.File("mydata.txt");
java.util.Scanner pachyderm =
                              new java.util.Scanner( elephant );
        // read a number from the file
int mouse = pachyderm.nextInt();
```

Reading with Scanner

- Reading from a file with a Scanner is just like reading from the keyboard
- All the usual methods are available
- A useful method is hasNext() which is true if there is more data in the file

Useful Scanner Methods

- **nextInt**() – read an int
- **nextDouble**() – read a double
- **nextLine**() – read the whole line as a String
- **next**() – read the next word as a String
- **hasNext**() – true if there is more data
- **hasNextInt**() – true if there is another int
- **hasNextDouble**() – true if there is another double

Add Numbers in a File

- Read whole numbers from file.txt and display the sum

```
java.io.File  inputFile = new java.io.File("file.txt");
java.util.Scanner  fileIn = new java.util.Scanner(inputFile);
int num, sum = 0;
while ( fileIn.hasNextInt() ) {
        num = fileIn.nextInt();
        sum = sum + num;
}
System.out.println("The numbers total to "+sum);
```

Exceptions

- When a Java program encounters an error during execution it "throws an exception"
- Exceptions can be caused by many, many different errors
- When an exception occurs, the default action is to display an explanation of the error and a "stack trace"

Stack Trace

- A stack trace shows which method called which method

```
Exception in thread "main" java.io.FileNotFoundException:
at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.<init>(FileInputStream.java:38)
at java.util.Scanner.<init>(Scanner.java:656)
at ReadFile.main(ReadFile.java:9)
```

- On line 9 of my main method in my ReadFile class, my program called a Scanner method
- The Scanner method called a FileInputStream method
- FileInputStream detected an error in open method

Scanner File Exceptions

- When you create a Scanner object using a File object, it could throw java.io.FileNotFoundException
- Your program must handle the possible exception
- The easiest way to do this is to throw the exception

```
public static void main( String[] unused ) throws
                                    java.io.FileNotFoundException {
```

Throwing an Exception

- A method header may include a **throws** clause
- When a method throws an exception, the calling method is responsible for handling it
- When the main method throws an exception, the system will take the default action

```
void methA() {
    int dog = methB(); // must handle exception
}
int methB() throws Exception {
    …
}
```

Differences between reading from a file and the keyboard include

1. Create the Scanner
   using a File object
   not System.in
2. Throw an exception
3. All of the above
4. None of the above

```
/*  Example reading a file */
public class ReadFile {
   public static void main(String[] args) throws
                                        java.io.FileNotFoundException {
     java.util.Scanner keyboard = new
                                        java.util.Scanner(System.in);
     System.out.print("Enter the filename >");
     String filename = keyboard.next();

     java.io.File  inputFile = new java.io.File(filename);
     java.util.Scanner  fileIn = new java.util.Scanner(inputFile);
     String word;
     while ( fileIn.hasNext() ) {
        word = fileIn.nextLine();
        System.out.println( word );
     }
     fileIn.close();
   }
}
```

Closing Files

- After using a file, you should close it
- You can call the close() method on a Scanner object
- Closing a file releases the file and any program memory used by the file
- If your program was writing to the file, close will make sure everything is written to disk

Reading from the Web

- In Java you can read a file from a web server in almost the same way you read a file
- You need to create a **java.net.URL** object instead of a java.io.File object
- Instead of passing the File object to Scanner use **`openStream()`** method of a URL object
- You have to throw
  - java.net.MalformedURLException,
  - java.io.IOException

```
/*  Example reading a web file */
public class ReadWeb {
   public static void main(String[] args) throws
            java.net.MalformedURLException,
            java.io.IOException {
      java.util.Scanner keyboard = new
                  java.util.Scanner(System.in);
      System.out.print("Enter the URL >");
      String filename = keyboard.next();

      java.net.URL webFile = new java.net.URL(filename);
      java.util.Scanner  fileIn = new
            java.util.Scanner(webFile.openStream());
      String word;
      while ( fileIn.hasNext() ) {
          word = fileIn.nextLine();
          System.out.println( word );
      }
      fileIn.close();
   }
```

String Methods

- There are several methods for objects of the class String that can be useful
- We have used several String methods in our labs and homework
- It is not important to know the details of every String method. It is important to know that they exist and you could look up the details

Checking if Strings Are Equal

- Remember that you cannot use == to test if Strings are equal

- equals(String anotherString) is true if both strings are composed of the same letters
- equalsIgnoreCase(String anotherString) is true if this String is equal to another String ignoring case

Example Loop Using equals

- Read and display words until "done"

```
String word;
word = keyboard.next();
while (!word.equals("done")) { // word != "done"
        System.out.println(word);
        word = keyboard.next();
}
```

What is displayed?

```
String bird = "grade", dog = "Grade";
if (bird.equals( dog ) )
   System.out.print("E " );
else
   System.out.print("N ");
if (bird. equalsIgnoreCase( dog ))
   System.out.print("E" );
else
   System.out.print("N");
```

1. E E
2. E N
3. N E
4. N N

Comparing Strings

- mine.**compareTo**(String another) Compares two strings lexicographically
- mine.**compareToIgnoreCase**(String another) Compares two strings lexicographically, ignoring case differences

- These methods return zero if the strings are equal, less than zero if mine < another and greater than zero if mine > another

Searching Strings

- contains(String str) Returns true if and only if this string contains the specified String
- startsWith(String prefix) Tests if this string starts with the specified prefix
- endsWith(String suffix) Tests if this string ends with the specified suffix

Comparing Examples

```
String major = "Computer Engineering";
if ( major.contains( "put" ) ) {
        // This would be true
}
if ( major.endsWith( "ring" ) {
        // This would also be true
}
```

What is displayed?

```
String rat = "dysfunctional";
if ( rat.contains("FUN ") )
    System.out.print("F " );
else
    System.out.print("N ");
if (rat. endswith("nal" ))
   System.out.print("Y" );
else
    System.out.print("N");
```

1. F Y
2. F N
3. N Y
4. N N

Alphabetical Comparisons

- Find word earliest in the alphabet

```
Scanner scanIn = new Scanner( myFileObj );
String word, first;
first = scanIn.next();
while (scanIn.hasNext()) {
        word = scanIn.next();
        if (word.compareTo( first ) < 0) {
                first = word;
        }
}
System.out.println(first+" is first");
```

Searching Strings

- These methods not only tell you if a string contains something, but where it is
- indexOf(String str) Returns the index within this string of the first occurrence of the specified String
- indexOf(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

indexOf Example

- The position of a character in a string starts counting at zero

```
String major =   "Computer Science";
                //   0123456789012345
```

```
int where = major.indexOf( "put" );      // 3
```

Taking Strings Apart

- substring(int beginIndex, int endIndex) Returns a new string that is a substring of this string. The substring begins at beginIndex and extends to the character at index endIndex - 1. Thus the length of the substring is endIndex-beginIndex
- charAt(int index) Returns the char value at the specified index

substring Example

```
String major =   "Computer Engineering";
                 //   01234567890123456789
```

String result = major. substring(11, 14);

Separating Strings

- Separate the user name from an email address

```
String email, userid;
email = keyboard.next();      // read email address
int pos = email.indexOf('@');
if (pos > 0) {
        userid = email.substring(0, pos-1 );
}  else {
        userid = "unknown";
}
```

Changing Case

- toUpperCase() returns a copy of the String with all of the letters in upper case
- toLowerCase() returns a copy of the String with all of the letters in lower case

```
String gnu = “Today is 11/11/11”;
String wildebeest = gnu.toUpperCase();
System.out.println(gnu +”\n”+wildebeest);
```

```
Today is 11/11/11
TODAY IS 11/11/11
```

Miscellaneous String Methods

- trim() Returns a copy of the string, with leading and trailing whitespace omitted
- length() Returns the length of this string

```
String hair = "  What ever    ";
              // 01234567890123
String bald = hair.trim();
// bald is  "What ever";
```

Multiple Methods

- Methods can be strung together

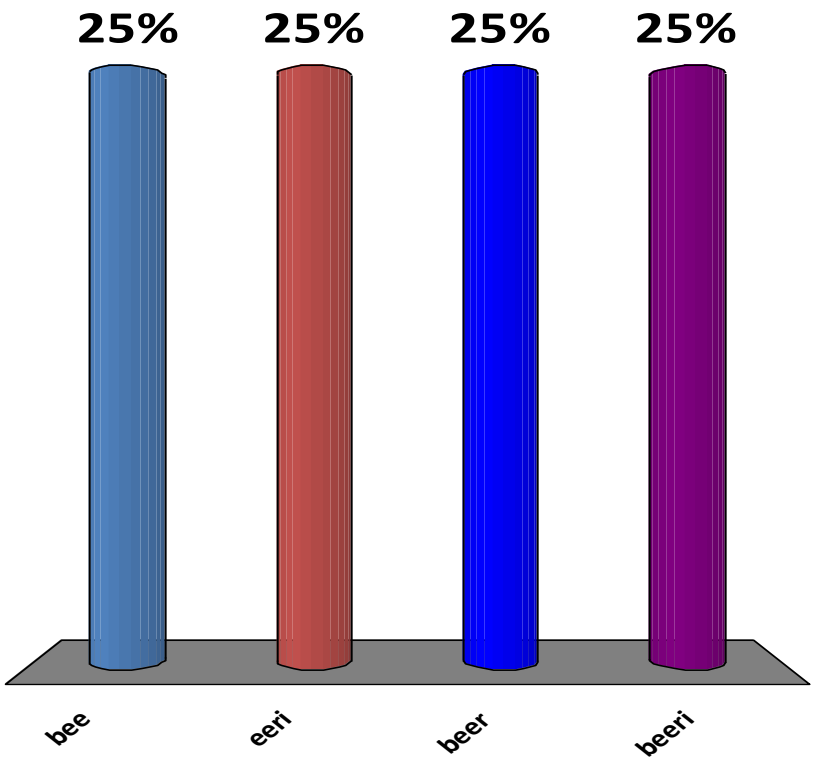
```
String hair = "  What ever    ";
                // 01234567890123
String cut = hair.trim().toUpperCase();
int poodle = cut.length();  // poodle = 9
//  cut is "WHAT EVER"
```

What is the value of result?

```
String major =      "Computer Engineering";
                //   01234567890123456789
```

String result = “b” + major. substring(14, 17);

1. bee
2. eeri
3. beer
4. beeri



Solution

```
String major =      "Computer Engineering";
                //   01234567890123456789
```

String result = “b” + major. substring(14, 17);

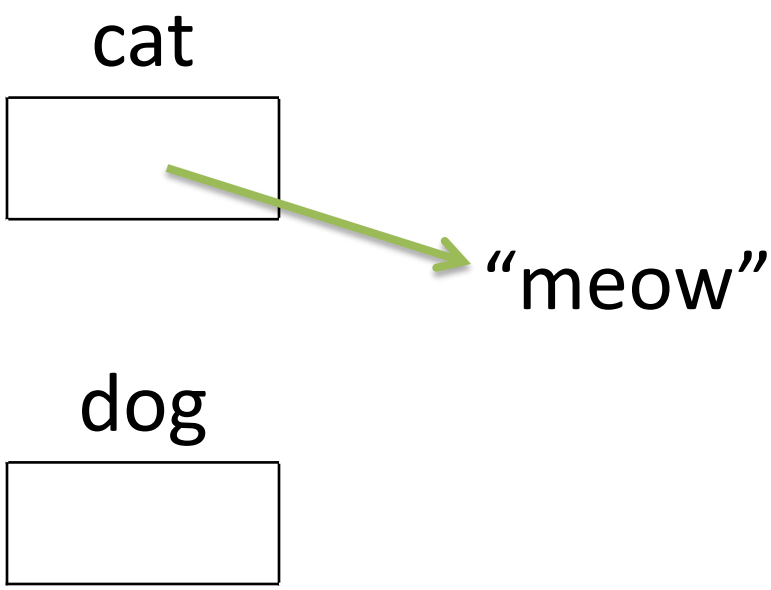
- result is **“beer”**

Strings Do Not Change

- A String never changes but a String variable can reference a different String
- Strings are said to be **immutable**

String cat = “meow”;

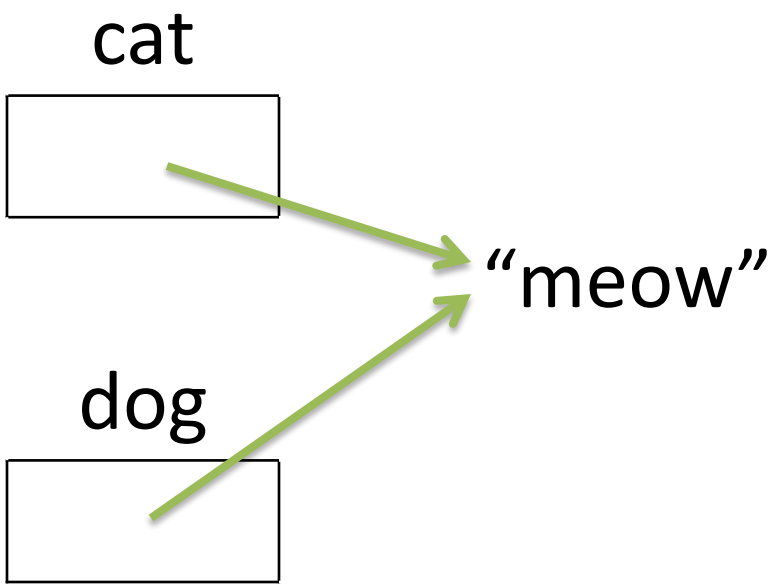
String dog;



Setting One String to Another

- When you set one string to another, it makes the second string reference the string data

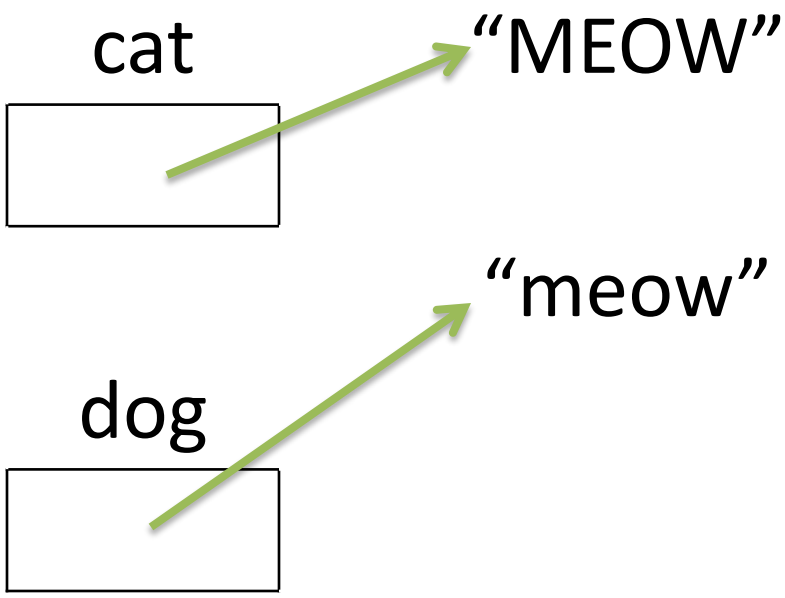
```
String cat = “meow”;
String dog;
dog = cat;
```



Changing a String

- A String never changes but a String variable can reference a different String

```
String cat = “meow”;
String dog;
dog = cat;
cat = cat.toUpperCase();
```



Second Exam

- The second exam will be in lecture on Friday, October 19
- The exam will cover everything since the first exam
- Read the textbook chapters 5, 6 and 9.7
- You can have one 8½ by 11" page of notes

Tutorial System

- Answer any 25 questions from chapter 6 of TuringsCraft tutoring system
- 4 points for each correct answer and 1 point for each incorrect answer
- Due Friday, October 19, 2012 by 9:00am

Programming Homework

- Write a program to read a file and count the number of words and sentences
- Due by 5:00pm on Friday, October 19, 2012