Announcements

3:45 today, CS BLDG 151 - Advising meeting. Find out about CS major, minor, classes, etc.
187 next class; CS 575 = Math 513
CS 291W - Intro to web programming
Programs 5 (Sudoku), 6 (Scrabble), due 11/21, 24
Topics for today:

Quick review of recent material

Exceptions
public class Person{
    private String name;

    public Person(String who){ name = who;}

    public String getName(){return name;}

    public void personInfo(){
        System.out.println("I am " + name);
    }

    public boolean bigName(){
        if (name.length() > 6)
            return true;
        else return false;
    }
}
public class SizedPerson extends Person{

    private int height; // attribute only in SizedPerson

    public SizedPerson(String who, int ht){
        super(who);
        // build person part of SizedPerson
        height = ht;
    }

    public void personInfo(){
        super.personInfo(); // personInfo frm super class
        System.out.println("my height is " + height);
    }
}
public class PersonDriver{
    public static void main(String[] args){
        SizedPerson p = new SizedPerson("hilda",62);
        p.personInfo();
        System.out.println("big name? " + p.bigName());
    }
}

----------------

Output
----------------

I am hilda
my height is 62
big name? false
public class PersonDriver{
    public static void main(String[] args){
        SizedPerson p = new SizedPerson("hilda",62);
        p.personInfo();
        System.out.println("big name? " + p.bigName());
    }
}

----------------
Output
----------------
I am hilda
my height is 62
big name? false

just in the base class
personInfo from derived class
Person is the **base** class or the **super** class; SizedPerson is the **derived** or the **subclass** class. SizedPerson **extends** Person; SizedPerson **specializes** Person.

The **super** class is the **more general** class.
int height

String name

Person(…)

getName()

bigName()

personInfo()

Person

SizedPerson(…)

personInfo()

SizedPerson
Files

How to read from / write to them
Your computer's file system is - barring catastrophe - permanent

It's the repository of data (your programs, term papers, diary, emails, popular URLs, and so forth)

Actually, we'll distinguish between text files (blah.txt, Infant.java), for which the basic unit is a character, and binary files, for which the basic unit is the byte.

One other distinction: files can be read in sequential style - beginning to end (very inefficient for some applications); or in random access file (jump directly to a disk location - much more efficient).

Our focus here: sequential text files
Some Observations:

For many applications, what we do when we read text is done at the line level (e.g. here’s a line: capitalize it, count it, convert it to numbers, look for a word, etc.)

At the same time, the file-reading machinery is unchanging: get the name, make a FileReader, make a Scanner, give me the lines

We’ll separate these two aspects of DisplayFile, then use inheritance so that we never have to re-examine the unchanging part!
import java.util.Scanner;
import java.io.*;

public class Echo{
    String fileName; // external file name
    Scanner scan; // reads from external file

    public Echo(String f) throws IOException{
        fileName = f;
        scan =
            new Scanner(new FileReader(fileName));
    }
}
public void readLines()
{
    while(scan.hasNext()){
        processLine(scan.nextLine());
    }
    scan.close();
}

public void processLine(String line){
    System.out.println(line);
}
import java.util.Scanner;
import java.io.*;
public class ReadDriver{
    public static void main(String[] args) throws IOException{
        String fileName;
        Scanner nameReader = new Scanner(System.in);
        System.out.println("Enter a file name");
        fileName = nameReader.nextLine();
        Echo e = new Echo(fileName);
        e.readLine();
    }
}
import java.io.*;
public class LineCount extends Echo{
    private int count = 0;

    public LineCount(String f) throws IOException {
        super(f);
    }

    public void processLine(String line){count++;}

    public void reportLineCount(){
        System.out.println("Line count: "+ count);
    }
}
LineCount

```
int count = 0;

Echo
String filename
Echo(String f)
void readLines()
void processLine(String line){..}
```

```
void processLine(String line){count++;
void reportLineCount(){…}
```
NumberEcho

int lineNum = 0;

Echo
String filename
Echo(String f)
void readLines()
void processLine(String line){..}

void processLine(String line){lineNum++;
    S.o.p.(lineNum + " " + line);}
WordCount

```java
int wordCt = 0;

Echo
String filename
Echo(String f)
void readLines()
void processLine(String line){..}

void processLine(String line){chop up, count, update}
void reportCount(){S.o.p.(wordCt);}
Java has shadow system for handling errors: **Exceptions**

This is a system of classes and objects that employ special machinery for detecting errors and shifting responsibilities for handling these problems.

Java recognized two kinds of errors:

- **Unchecked exceptions**: errors that occur because of flaws in logical thinking on the part of the programmer (e.g. step off end of array)

- **Checked exceptions**: errors that can be expected to occur, and may occur despite your best efforts to avoid them (trying to read from a file that doesn’t exist)
The main class we’ll concern ourselves with:

The class **Exception**

It has many derived classes:

- **ArithmeticException** (unchecked)
- **ArrayOutOf BoundsException** (unchecked)
- **IOException** (checked)
- **NumberFormatException** (unchecked)
Exceptions are objects, either from the Exception class

or a derived class of the Exception class

Exceptions embody - yup - exceptional behavior

One attribute: a String message, (with a getMessage() method)

One constructor has a String parameter - the message
public class Except0{
    public static void main(String[] args){
        int k;  int a = 3;  int b = 0;
        k = a/b;
    }
}

Java reports:

Exception in thread "main"
java.lang.ArithmeticException: / by zero
at Except0.main(Except0.java:7)
Java has special machinery, the try/catch construction, for dealing with exceptions (handling exceptions)

An exception is handled (by you) if your code detects the error and takes some action in response -- as in the next example.
public static void main(String[] args){
    int k; int a = 3; int b = 0;
    try{
        k = a/b;
    }
    catch(ArithmeticException e)
    {
        System.out.println(e);
    }
}

Java reports:
java.lang.ArithmeticException: / by zero
The try/catch harness

```java
try{
    blah    // this is normal execution
    blah
}

catch(Exception1 e){   }

catch(Exception2 e){   }
```

public class Except2{
    public static void main(String[] args){
        String s = "98.6";
        int n;
        try{
            n = Integer.parseInt(s);
            System.out.println(n*n); }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
What happens if:

catch (NumberFormatException e) { }?
catch (ArrayOutOf BoundsException e) { }?
A more realistic example - you are supposed to enter an integer, representing your age (the program then reports your age for next year..)

Some interaction:

enter your age
39e
Bad input. 39e is not an integer. You must input an integer

enter your age
39
next year you will be 40
Aside:
Still a contrived example
Illustrates an extremely important aspect of computer programming:
“bullet-proofing” / “error trapping” / “idiot proofing”
import java.io.*;
import java.util.*;

public class IntegerInput{
public static void main(String[] args){
    int n = -1;
    String userInput;
    Scanner scan = new Scanner(System.in);
while (n < 0) {
    System.out.println("enter your age");
    try {
        if (scan.hasNextInt()) // false if non-int next
            n = scan.nextInt();
        else { // non integer submitted
            userInput = scan.next(); // get actual input
            throw new Exception("Bad input. " + userInput + " is not an integer. You must input an integer");
        }
    } // end try
    catch (Exception e){ System.out.println(e.getMessage()); } // end while
    System.out.println("next year you will be " + (n + 1));
}

What does `throw` do? It “presents” an exception object of the indicated kind at a new location in the program.
What if you enter a negative number?

Not handled very well
(No message)..

enter your age
-39
enter your age
-4
enter your age
One way to handle this:
make a new kind of Exception..

```java
public class NegativeException extends Exception{
    public NegativeException() { }
    public NegativeException(String msg){
        super(msg);
    }
}
```

And then:
while (n < 0) {
    System.out.println("enter your age");
    try {
        if (scan.hasNextInt())
            n = scan.nextInt();
        else {
            String userInput = scan.next();
            throw new Exception("Bad input. "+ userInput + " not an integer. You must input an int");
        }
    }
    if (n < 0) throw new NegativeException();
}
catch (NegativeException e)
    { System.out.println("age must be >= 0"); }
catch (Exception e)
    { System.out.println(e.getMessage()); }
} // end while
System.out.println("next year you will be " + (n + 1));

Notice: two kinds of exceptions, more specific one first
One more example

You have a data file of numerical data, integers, one per line
It may contain errors (deep space info?)

You want to compute the average value of the entries - but you’re happy just to skip over the bad data
Here is nums.txt

50
60
3e
10

Here’s the output:
Bad Entry: 3e
40.0  // note: the average of 50, 60, 10
import java.io.*;
import java.util.*;
public class NumberReader extends Echo {

    int sum = 0;
    int count = 0;

    public NumberReader(String f) throws IOException {
        fileName = f;
    }

    This says: if there is a file-reading problem here, I'm not going to handle it locally. I'll leave for the calling method (here: main in the driver) to handle it.
public void processLine(String line){
    try{
        int num = Integer.parseInt(line);
        sum = sum + num;
        count++;
    }
    catch(Exception e)
    {
        System.out.println("Bad Entry: " + line);
    }
}

public void printResult()
{
    System.out.println((double)sum/count);
}
}
public class NumDriver{
    public static void main(String[] args){
        try{
            NumberReader num =
                new NumberReader("nums.txt");
            num.readLine();
            num.printResult();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
This is considered tacky:

```java
public class ReadDriver{
    public static void main(String[] args)
        throws IOException
    {
        String fileName;
        Scanner nameReader = new Scanner(System.in);
        System.out.println("Enter a file name");
        fileName = nameReader.nextLine();
        Echo e = new Echo(fileName);
        e.readLine();
    }
}
```
public class ReadDriver{
public static void main(String[] args) {
try{
    String fileName;
    Scanner nameReader = new Scanner(System.in);
    System.out.println("Enter a file name");
    fileName = nameReader.nextLine();
    Echo e = new Echo(fileName);
    e.readLine();
} 
catch(IOException e){System.out.println(e);}
} 
}