The Call Stack

• Execution begins in the main method
  – That method creates objects and invokes methods on them
    • When execution jumps to another method an entry is added to the call stack
      – The current method
      – Where the call occurred in that method
    • When a method finishes executing
      – The entry is removed from the call stack
      – And execution returns to the next line in that method
      – Until the main method finishes
The Call Stack

- Execution begins in the main method
  - The FIRST ENTRY in the call stack is WHERE the local variables of `main` live!

```java
public static void main(String[] a)
{
    int myIntVar;
    Picture myPictVar;
    House myHouseVar;
    for( myIntVar = 0;
        myIntVar<10; myIntVar++)
        { /*you can imagine*/ } 
}
```
FIGURE 12.2
Showing a call stack before a method returns and after.
FIGURE 12.2
Showing a call stack before a method returns and after.
2.2 call stack before a met
public static void main(String[] a) {
    int myIntVar;
    Picture myPictVar;
    House myHouseVar;
}
FIGURE 12.2
Showing a call stack before a method returns and after.
Suppose main calls another method, say method1

• The CALL STACK ENTRY pushed ON TOP OF main's is where that 2\textsuperscript{nd} method's local variables live!
FIGURE 12.2
Showing a call stack before a method returns and after.
Example Call Stack

• Remove the check for gradeArray == null in the getAverage method
  – And run the main method

• This says a null pointer exception occurred
  – at line 109 in the method getAverage in the Student class
    • Which was called from method toString at line 120

java.lang.NullPointerException:
at Student.getAverage(Student.java:109)
at Student.toString(Student.java:120)
at java.lang.String.valueOf(String.java:2131)
at java.io.PrintStream.print(PrintStream.java:462)
at java.io.PrintStream.println(PrintStream.java:599)
at Student.main(Student.java:129)
Learning Goals

• Computing concepts
  – What is a file?
  – What is the full path name of a file?
  – What is an exception?
  – What is a call stack?
• where the local vars are!
  – How to handle exceptions?
  – How to read from a file?
Files

• Files are named collections of bytes on your hard disk
  – Often have a base name and suffix
    • Like barbara.jpg

• Are grouped into directories
  – A directory can have other directories in it
  – There is often a root directory
    • Like the C: drive on Windows

• A path is the list of all the directories from the root to the file
  – And includes the file’s base name and suffix
Reading from a File

• When we read from a file
  – We copy data from disk into memory

• Things can go wrong
  – The file may not exist
  – The disk may go bad
  – The file may change while we are reading it

• In Java when things go wrong an java.lang.Exception object is created
Picture of a Path Tree

• Drawing a path yields an upside down tree
  – With the root at the top
  – And the leaves at the bottom

• C:\intro-prog-java\mediasources\640x480.jpg
Possible Exceptions

• What would happen if we try to read from a file that doesn’t exist?
  – We would get a FileNotFoundException
• What would happen if we try to read past the end of the file?
  – IOException
• What would happen if the file changes while we are reading it?
  – IOException
• The code won’t compile unless we
  – Either handle the exception with a try and catch
  – Or throw the exception
Generating Runtime Exceptions

• Try the following in the Interactions Pane
  – String test = null;
  – test.length();
    • What exception do you get?

• Try this
  – int sum = 95;
  – int num = 0;
  – System.out.println(sum/num);
    • What exception do you get?
Exceptions

• Exceptions are objects of the class java.lang.Exception
  – Or are objects of classes that inherit from Exception

• There are two types of exceptions
  – Checked and Unchecked
    • Checked exceptions must be caught or thrown
      – IOException and FileNotFoundException
    • Unchecked exceptions do not have to be caught or thrown
      – NullPointerException, ArrayIndexOutOfBoundsException
Exception Inheritance Tree

• All classes inherit from Object
• All Exception classes inherit from Exception
Importing Classes To Read From Files

• To read from a file we will use classes in the java.io package
  – Which means that we will need to use import statements
    • Or use the full names of classes
      – package.Class

• Import statements go before the class declaration in the file
  – import package.Class;
    • Allows the short name to be used for just the mentioned class
  – import package. * ;
    • Allows the short name to be used for any class in this package
Reading from a File

• To read from a character based file
  – Use a FileReader object
    • This class knows how to read character data from a file
  – With a BufferedReader object
    • To buffer the data as you read it from the disk
      – Into memory
    • Disks are much slower to read from than memory
      – So read a big chunk from disk into memory
        » And then read from the chunk in memory as needed
Using Try, Catch, and Finally Blocks

- Wrap all code that can cause a checked exception in try, catch (and optionally finally) blocks
  
  ```java
  try {
    // code that can cause an exception
  } catch (ExceptionClass ex) {
    // handle this exception
  } catch (ExceptionClass ex) {
    // handle this exception
  } finally {  // optional
    // do any required clean up
  }
  ```
public class SimpleReader
{
    /**
     * Method to read a file and print out the contents
     * @param fileName the name of the file to read from
     */
    public void readAndPrintFile(String fileName)
    {
        String line = null;

        // try to do the following
        try {
            // create the buffered reader
            BufferedReader reader =
                new BufferedReader(new FileReader(fileName));

            //...
// Loop while there is more data
while((line = reader.readLine()) != null)
{
    // print the current line
    System.out.println(line);
}

// close the reader
reader.close();
catch(FileNameNotFoundException ex) {
    SimpleOutput.showError("Couldn't find " + fileName + " please pick it.");
    fileName = FileChooser.pickAFile();
    readAndPrintFile(fileName);
}

public static void main(String[] args) {
    SimpleReader reader = new SimpleReader();
    reader.readAndPrintFile("test.txt");
}
Key Points

• Notice that we put all ‘normal’ code in the try block
  – This handles the case when everything goes right
• We can catch more than one exception
  – Here we caught FileNotFoundException
    • And used the FileChooser to have the user pick the file
      – And then called the method again
  – Catching Exception will catch all children of Exception as well
    • So make it the last Exception you catch
• Finally blocks are not required
  – But they will always execute if there is an exception or not
Summary

• A file is a named collection of bytes on your disk.
  – beach.jpg
• The full path name of a file is a list of all the directories from the root to the file
  – C:\intro-prog-java\mediasources\beach.jpg
• Exception is short for exceptional event
  – When things go wrong
• Handle checked exceptions with try and catch (and optionally finally) blocks
• To read from a file use a BufferedReader and a FileReader
  – And catch Exceptions
Creating and Modifying Text
part 2
READ THIS IN Chapter 12!

Barb Ericson
Georgia Institute of Technology
Oct 2005
BASE for Lecture 25
S. Chaiken
Dec. 2012
Univ. at Albany.
Georgia Institute of Technology
The Call Stack

• Execution begins in the main method
  – That method creates objects and invokes methods on them
    • When execution jumps to another method an entry is added to the call stack
      – The current method
      – Where the call occurred in that method
    • When a method finishes executing
      – The entry is removed from the call stack
      – And execution returns to the next line in that method
      – Until the main method finishes
The Call Stack

• Execution begins in the main method
  – The FIRST ENTRY in the call stack is WHERE the local variables of main live!

```java
public static void main(String[] a)
{
    int myIntVar;
    Picture myPictVar;
    House myHouseVar;
    for( myIntVar = 0;
        myIntVar<10; myIntVar++)
    {
        /*you can imagine*/
    }
}
```
FIGURE 12.2
Showing a call stack before a method returns and after.
FIGURE 12.2
Showing a call stack before a method returns and after.
2.2

call stack before a met
THESE ARE LOCAL VARIABLES!!

```java
public static void main(String[] a) {
    int myIntVar;
    Picture myPictVar;
    House myHouseVar;
}
```
FIGURE 12.2
Showing a call stack before a method returns and after.
Suppose main calls another method, say \texttt{method1}.

- The CALL STACK ENTRY pushed ON TOP OF main's is where that 2\textsuperscript{nd} method's local variables live!
FIGURE 12.2
Showing a call stack before a method returns and after.
Example Call Stack

- Remove the check for gradeArray == null in the getAverage method
  - And run the main method
- This says a null pointer exception occurred
  - at line 109 in the method getAverage in the Student class
    - Which was called from method toString at line 120

java.lang.NullPointerException:
at Student.getAverage(Student.java:109)
at Student.toString(Student.java:120)
at java.lang.String.valueOf(String.java:2131)
at java.io.PrintStream.print(PrintStream.java:462)
at java.io.PrintStream.println(PrintStream.java:599)
at Student.main(Student.java:129)

This exception was caused by removing the check to see if the gradeArray was null. And then creating a new student with passing in just the student name.
Learning Goals

• Computing concepts
  – What is a file?
  – What is the full path name of a file?
  – What is an exception?

  What is a call stack?
  • where the local vars are!
  – How to handle exceptions?
  – How to read from a file?
Files

- Files are named collections of bytes on your hard disk
  - Often have a base name and suffix
    - Like barbara.jpg
- Are grouped into directories
  - A directory can have other directories in it
  - There is often a root directory
    - Like the C: drive on Windows
- A path is the list of all the directories from the root to the file
  - And includes the file’s base name and suffix
Reading from a File

• When we read from a file
  – We copy data from disk into memory

• Things can go wrong
  – The file may not exist
  – The disk may go bad
  – The file may change while we are reading it

• In Java when things go wrong an java.lang.Exception object is created
Each directory is a branch node. There is only one root node. There can be many leaf nodes. A leaf node is any node that doesn’t have an arrow to another node from it (it can have one pointing to it).
Possible Exceptions

• What would happen if we try to read from a file that doesn’t exist?
  – We would get a FileNotFoundException
• What would happen if we try to read past the end of the file?
  – IOException
• What would happen if the file changes while we are reading it?
  – IOException
• The code won’t compile unless we
  – Either handle the exception with a try and catch
  – Or throw the exception
Generating Runtime Exceptions

• Try the following in the Interactions Pane
  – String test = null;
  – test.length();
    • What exception do you get?

• Try this
  – int sum = 95;
  – int num = 0;
  – System.out.println(sum/num);
    • What exception do you get?

Exceptions are an indication that you tried to do something the computer can’t
do or that an error occurred. We can’t get the length of an object when there
isn’t any object (as indicated by the keyword null). We can’t divide by 0.
Exceptions

- Exceptions are objects of the class java.lang.Exception
  - Or are objects of classes that inherit from Exception
- There are two types of exceptions
  - Checked and Unchecked
    - Checked exceptions must be caught or thrown
      - IOException and FileNotFoundException
    - Unchecked exceptions do not have to be caught or thrown
      - NullPointerException, ArrayIndexOutOfBoundsException

We don’t have to catch unchecked exceptions because to do so would add lots of code to our methods. However we can catch these exceptions if we want to.
Exception Inheritance Tree

- All classes inherit from Object
- All Exception classes inherit from Exception

[Diagram of exception inheritance tree]
Importing Classes To Read From Files

• To read from a file we will use classes in the java.io package
  – Which means that we will need to use import statements
    • Or use the full names of classes
      – package.Class

• Import statements go before the class declaration in the file
  – import package.Class;
    • Allows the short name to be used for just the mentioned class
  – import package. *;
    • Allows the short name to be used for any class in this package

Importing a class just tells Java where to find it. Each time Java encounters a new class it loads the file that defines the class. Importing many classes doesn't make the code any bigger like including code does in C.
**Reading from a File**

- To read from a character based file
  - Use a FileReader object
    - This class knows how to read character data from a file
  - With a BufferedReader object
    - To buffer the data as you read it from the disk
      - Into memory
    - Disks are much slower to read from than memory
      - So read a big chunk from disk into memory
        » And then read from the chunk in memory as needed
Using Try, Catch, and Finally Blocks

- Wrap all code that can cause a checked exception in try, catch (and optionally finally) blocks
  
  ```java
  try {
      // code that can cause an exception
  }
  catch (ExceptionClass ex) {
      // handle this exception
  }
  catch (ExceptionClass ex) {
      // handle this exception
  }
  finally {  // optional
      // do any required clean up
  }
  ```

You can put the `{ on the next line instead of following the try but this is standard practice.
SimpleReader - Example Class

class SimpleReader
{
   /**
    * Method to read a file and print out the contents
    * @param fileName the name of the file to read from
    */
   public void readAndPrintFile(String fileName)
   {
      String line = null;

      // try to do the following
      try {

         // create the buffered reader
         BufferedReader reader =
            new BufferedReader(new FileReader(fileName));

      } catch (IOException e) {
         // handle the exception
      }
   }
}
Simple Reader - Continued

// Loop while there is more data
while((line = reader.readLine()) != null)
{
    // print the current line
    System.out.println(line);
}

// close the reader
reader.close();
Simple Reader - Continued

```java
} catch(NoSuchFieldException ex) {
    SimpleOutput.showWarning("Field name is incorrect");
    return;
} catch(FileNotFoundException ex) {
    SimpleOutput.showError("Couldn't find " + fileName +
                          " please pick it.");
    fileName = FileChooser.pickAFile();
    readAndPrintFile(fileName);
} catch(Exception ex) {
    SimpleOutput.showError("Error reading file " + fileName);
    ex.printStackTrace();
}

public static void main(String[] args)
{
    SimpleReader reader = new SimpleReader();
    reader.readAndPrintFile("test.txt");
}
```
Key Points

• Notice that we put all ‘normal’ code in the try block
  – This handles the case when everything goes right
• We can catch more than one exception
  – Here we caught FileNotFoundException
    • And used the FileChooser to have the user pick the file
      – And then called the method again
  – Catching Exception will catch all children of Exception as well
    • So make it the last Exception you catch
• Finally blocks are not required
  – But they will always execute if there is an exception or not

Try changing the order of the Exceptions to catch Exception first and then FileNotFoundException. It won’t even compile because we will never get to the code that catches the FileNotFoundException. A FileNotFoundException is a child of Exception and will be caught when you catch Exception.
Summary

- A file is a named collection of bytes on your disk.
  - beach.jpg
- The full path name of a file is a list of all the directories from the root to the file
  - C:\intro-prog-java\mediasources\beach.jpg
- Exception is short for exceptional event
  - When things go wrong
- Handle checked exceptions with try and catch (and optionally finally) blocks
- To read from a file use a BufferedReader and a FileReader
  - And catch Exceptions