Lecture March 28 - Apartment Example and File IO

Bentley James Oakes

March 27, 2018

No Office Hours Today

- No office hours today

This Lecture

1. Apartment Example
2. Adding a Player
3. File Input/Output
4. IO Errors
5. Instance Creation
6. Writing to a File
7. Exception Handling
8. Exception Types
9. File IO Exceptions

Section 1

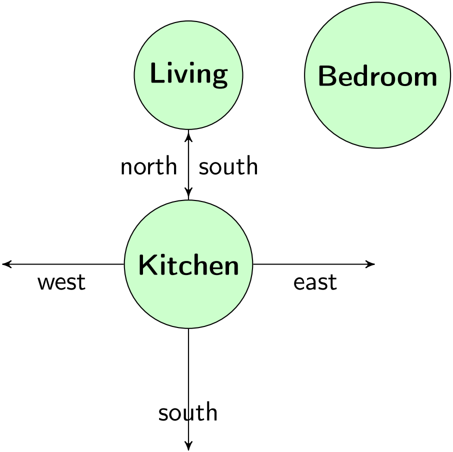
Apartment Example

Connected Rooms Example

- We're going to represent an apartment with a few rooms
- We'll have an `Apartment` class with `Rooms` connected to each other
    - For example, the living room will be to the north of the kitchen
- We'll store the length and width for each room, so that we can calculate the apartment's total area

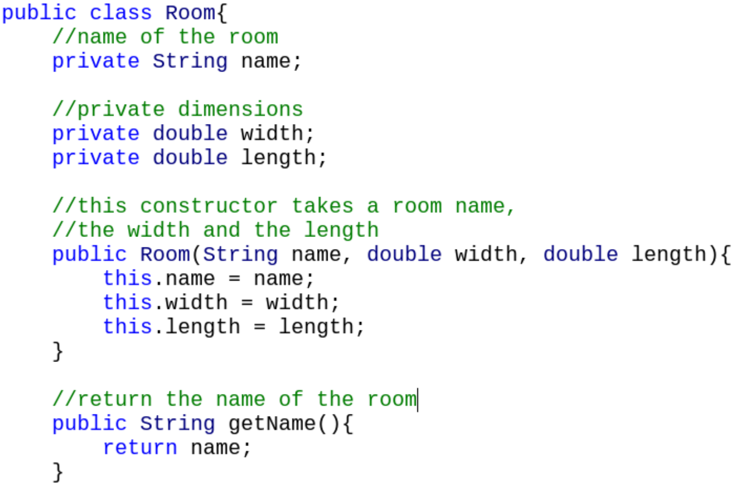
Room Connections

- We are going to connect instances of these rooms together.
- For example, we might place a living room to the north of the kitchen.
- In this case, we must store the address of the living room in the kitchen, and vice versa.



Room Start

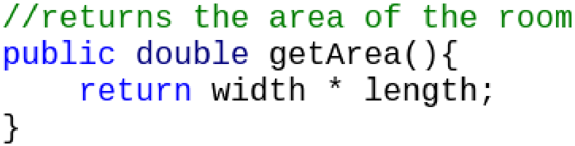
Here we define the room name, width and length, as well as the `Room` constructor.



```
public class Room{
    //name of the room
    private String name;

    //private dimensions
    private double width;
    private double length;

    //this constructor takes a room name,
    //the width and the length
    public Room(String name, double width, double length){
        this.name = name;
        this.width = width;
        this.length = length;
    }

    //return the name of the room
    public String getName(){
        return name;
    }
```

Room Area

This method returns the area of a room.

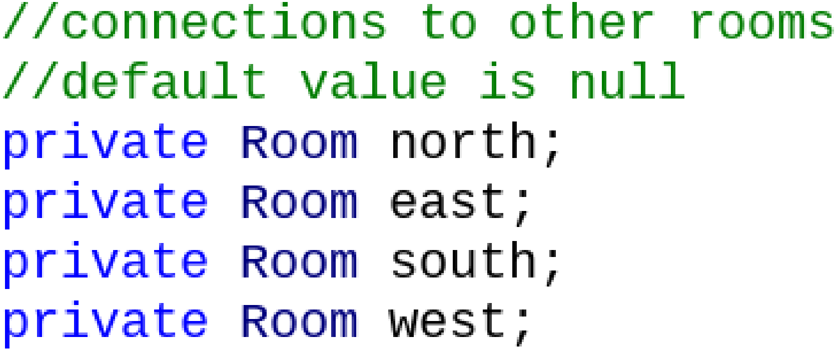


```
//returns the area of the room
public double getArea(){
    return width * length;
}
```

- For your future programs, think about the units
- Is the area in metres? feet? yards?
- Need to write documentation so everyone uses the same units

Room Connections

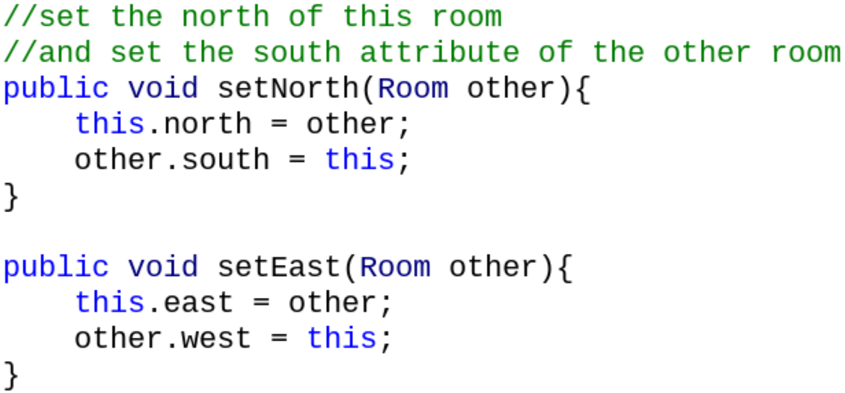
In the Room class, have attributes for the other rooms connected to this Room instance



```
//connections to other rooms
//default value is null
private Room north;
private Room east;
private Room south;
private Room west;
```

Room Connections

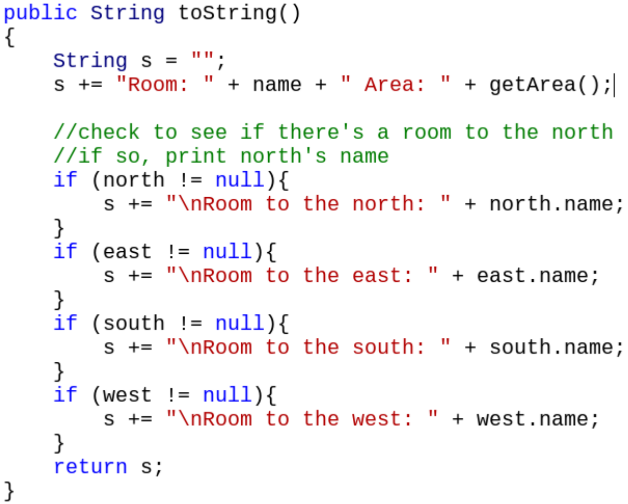
We also need methods to set the `Rooms` in the four directions



```
//set the north of this room
//and set the south attribute of the other room
public void setNorth(Room other){
    this.north = other;
    other.south = this;
}

public void setEast(Room other){
    this.east = other;
    other.west = this;
}
```

Room toString

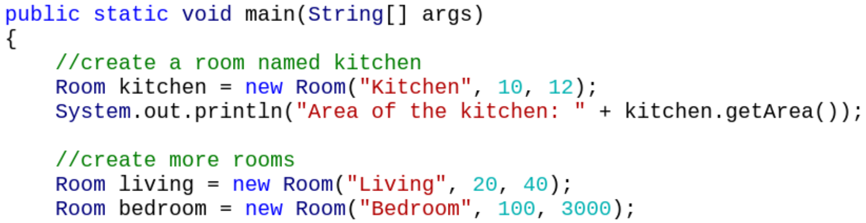
The toString method prints a few details about the room. As well, if there is a room in a direction (as in the room is not null), then that room's name is printed



```
public String toString()
{
    String s = "";
    s += "Room: " + name + " Area: " + getArea();

    //check to see if there's a room to the north
    //if so, print north's name
    if (north != null){
        s += "\nRoom to the north: " + north.name;
    }
    if (east != null){
        s += "\nRoom to the east: " + east.name;
    }
    if (south != null){
        s += "\nRoom to the south: " + south.name;
    }
    if (west != null){
        s += "\nRoom to the west: " + west.name;
    }
    return s;
}
```

Room Main

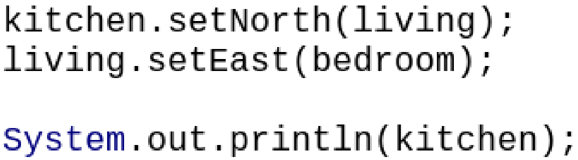
In a main method, let’s start by creating three rooms



```
public static void main(String[] args)
{
    //create a room named kitchen
    Room kitchen = new Room("Kitchen", 10, 12);
    System.out.println("Area of the kitchen: " + kitchen.getArea());

    //create more rooms
    Room living = new Room("Living", 20, 40);
    Room bedroom = new Room("Bedroom", 100, 3000);
```

Apartment Connect

Next, let's connect the rooms together
And to test, print out the kitchen

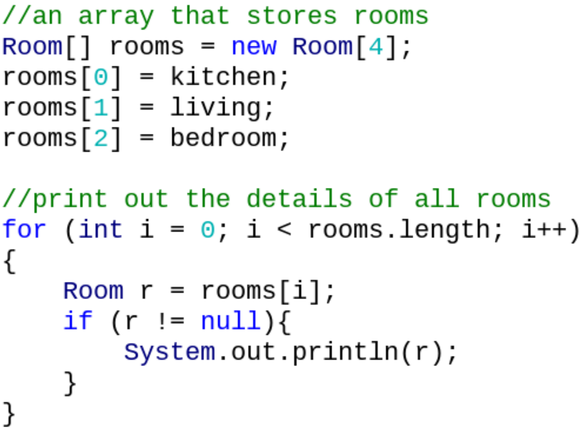


```
kitchen.setNorth(living);
living.setEast(bedroom);

System.out.println(kitchen);
```

Room: Kitchen Area: 120.0
Room to the north: Living

Placing Rooms in an Array

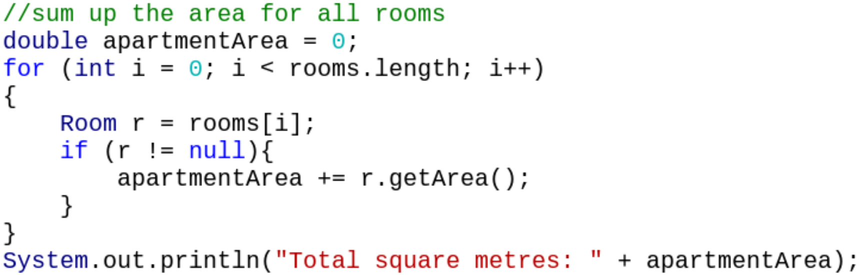
These rooms can then be placed in an array. We then iterate through the array and print out each room. Make sure to check to see if an entry in the array is *null*



```java
//an array that stores rooms
Room[] rooms = new Room[4];
rooms[0] = kitchen;
rooms[1] = living;
rooms[2] = bedroom;

//print out the details of all rooms
for (int i = 0; i < rooms.length; i++)
{
    Room r = rooms[i];
    if (r != null){
        System.out.println(r);
    }
}
```

Apartment Area

It is also easy to iterate through the array and calculate the total area for all the rooms



```
//sum up the area for all rooms
double apartmentArea = 0;
for (int i = 0; i < rooms.length; i++)
{
    Room r = rooms[i];
    if (r != null){
        apartmentArea += r.getArea();
    }
}
System.out.println("Total square metres: " + apartmentArea);
```

Section 2

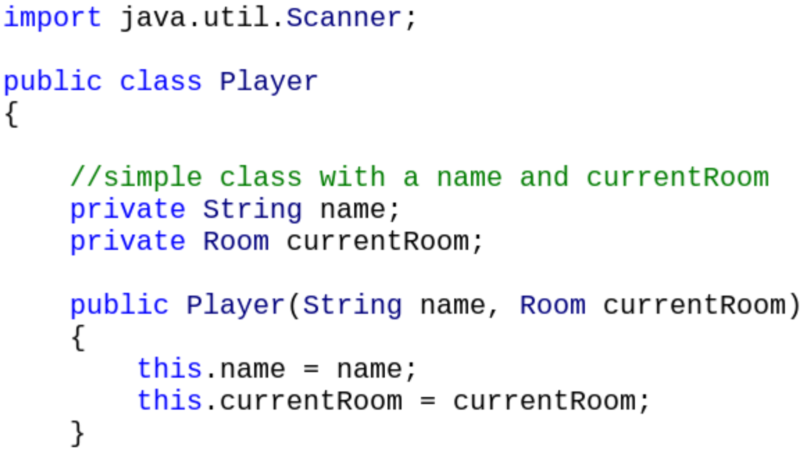
Adding a Player

Adding a Player

- To make our apartment more interesting, let's add a player to walk around
- The `Player` will have a current room
- The user will be able to input commands using a `Scanner`
- This will be very similar to Assignment 5

Player Start

Let's have the `Player` with a name and their current room

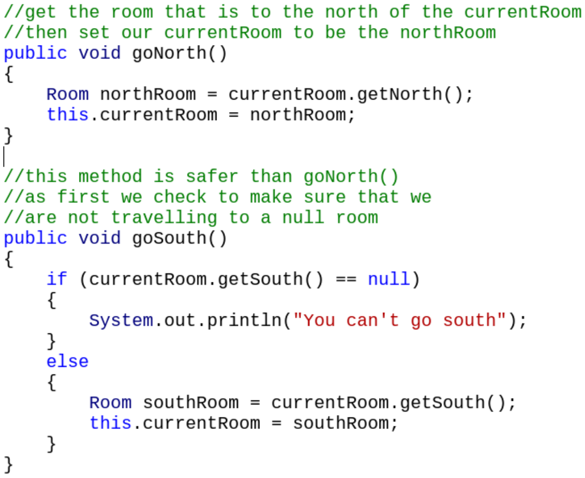


```
import java.util.Scanner;

public class Player
{

    //simple class with a name and currentRoom
    private String name;
    private Room currentRoom;

    public Player(String name, Room currentRoom)
    {
        this.name = name;
        this.currentRoom = currentRoom;
    }
```

Player Move

Let's add methods to travel between rooms

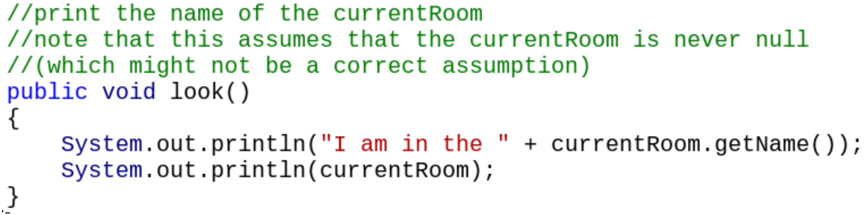
We'll need getter methods in our `Room` class to access `north, south, east, west`



```java
//get the room that is to the north of the currentRoom
//then set our currentRoom to be the northRoom
public void goNorth()
{
    Room northRoom = currentRoom.getNorth();
    this.currentRoom = northRoom;
}

//this method is safer than goNorth()
//as first we check to make sure that we
//are not travelling to a null room
public void goSouth()
{
    if (currentRoom.getSouth() == null)
    {
        System.out.println("You can't go south");
    }
    else
    {
        Room southRoom = currentRoom.getSouth();
        this.currentRoom = southRoom;
    }
}
```

Player Look

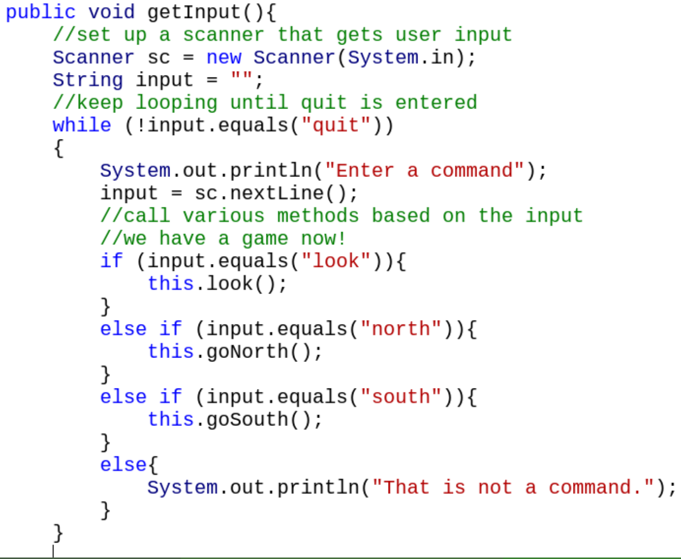
Let's add a `look` command, to see which room we're currently in



```java
//print the name of the currentRoom
//note that this assumes that the currentRoom is never null
//(which might not be a correct assumption)
public void look()
{
    System.out.println("I am in the " + currentRoom.getName());
    System.out.println(currentRoom);
}
```

Player Input

A non-static method within the `Player` class that asks for commands, and executes them



```java
public void getInput(){
    //set up a scanner that gets user input
    Scanner sc = new Scanner(System.in);
    String input = "";
    //keep looping until quit is entered
    while (!input.equals("quit"))
    {
        System.out.println("Enter a command");
        input = sc.nextLine();
        //call various methods based on the input
        //we have a game now!
        if (input.equals("look")){
            this.look();
        }
        else if (input.equals("north")){
            this.goNorth();
        }
        else if (input.equals("south")){
            this.goSouth();
        }
        else{
            System.out.println("That is not a command.");
        }
    }
```

Ways to Extend This

- Many ways to extend this example
- Making a game is a great way to learn
- Things to try:
  - Add items to the room that you can pick up and use
  - Example: Read and write notes that can be dropped in each room
  - Have a monster to fight, where both characters deal randomly-calculated damage
  - Have the player head through a maze, where they have to visit different locations before they win

Section 3

File Input/Output

File IO

- Lots of programs need to read data from a file, or write data to a file

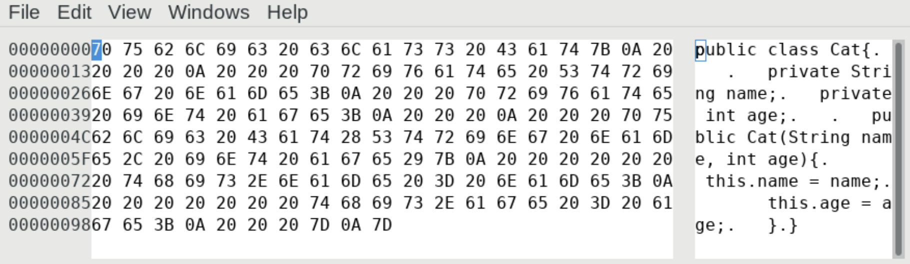
Let's briefly talk about what a file contains

File Definition

What is a file?

A file is a named collection of 1s and 0s.

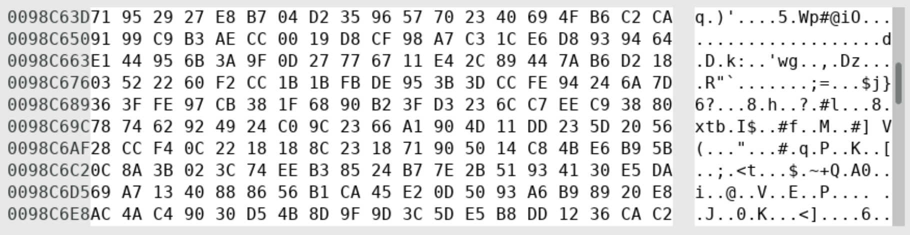
For example, we have 1s and 0s stored in the file `Cat.java`.
Here's the file seen in hex:



```
0000000070 75 62 6C 69 63 20 63 6C 61 73 73 20 43 61 74 7B 0A 20   public class Cat{.
0000001320 20 20 0A 20 20 20 70 72 69 76 61 74 65 20 53 74 72 69      .   private Stri
000000266E 67 20 6E 61 6D 65 3B 0A 20 20 20 70 72 69 76 61 74 65   ng name;.   private
0000003920 69 6E 74 20 61 67 65 3B 0A 20 20 20 0A 20 20 20 70 75    int age;.   .   pu
0000004C62 6C 69 63 20 43 61 74 28 53 74 72 69 6E 67 20 6E 61 6D   blic Cat(String nam
0000005F65 2C 20 69 6E 74 20 61 67 65 29 7B 0A 20 20 20 20 20 20   e, int age){.
0000007220 74 68 69 73 2E 6E 61 6D 65 20 3D 20 6E 61 6D 65 3B 0A    this.name = name;.
0000008520 20 20 20 20 20 20 74 68 69 73 2E 61 67 65 20 3D 20 61          this.age = a
0000009867 65 3B 0A 20 20 20 7D 0A 7D                               ge;.   }.}
```

Opening this File

Every file is just 1s and 0s. Programs interpret these values differently. Here's an example of an MP3 file. An audio program knows how to interpret this file.



```
0098C63D71 95 29 27 E8 B7 04 D2 35 96 57 70 23 40 69 4F B6 C2 CA  q.)'....5.Wp#@iO...
0098C65091 99 C9 B3 AE CC 00 19 D8 CF 98 A7 C3 1C E6 D8 93 94 64  ..................d
0098C663E1 44 95 6B 3A 9F 0D 27 77 67 11 E4 2C 89 44 7A B6 D2 18  .D.k:..'wg..,.Dz...
0098C67603 52 22 60 F2 CC 1B 1B FB DE 95 3B 3D CC FE 94 24 6A 7D  .R"`.......;=...$j}
0098C68936 3F FE 97 CB 38 1F 68 90 B2 3F D3 23 6C C7 EE C9 38 80  6?...8.h..?.#l...8.
0098C69C78 74 62 92 49 24 C0 9C 23 66 A1 90 4D 11 DD 23 5D 20 56  xtb.I$..#f..M..#] V
0098C6AF28 CC F4 0C 22 18 18 8C 23 18 71 90 50 14 C8 4B E6 B9 5B  (..."...#.q.P..K..[
0098C6C20C 8A 3B 02 3C 74 EE B3 85 24 B7 7E 2B 51 93 41 30 E5 DA  ..;.<t...$.~+Q.A0..
0098C6D569 A7 13 40 88 86 56 B1 CA 45 E2 0D 50 93 A6 B9 89 20 E8  i..@..V..E..P.... .
0098C6E8AC 4A C4 90 30 D5 4B 8D 9F 9D 3C 5D E5 B8 DD 12 36 CA C2  .J..0.K...<]....6..
```

File with Characters

In COMP 202, we'll just read and write data to/from text files.
We will need to know the format of our files we want to read and write.
We'll talk about this in a bit.

Reading/Writing Files in Java

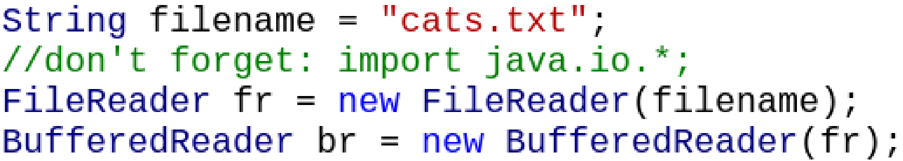
Java has some objects for reading and writing files

- `FileReader`
- `BufferedReader`
- `FileReader`
- `BufferedWriter`

But Java forces us to handle possible errors when doing so

Java File Objects

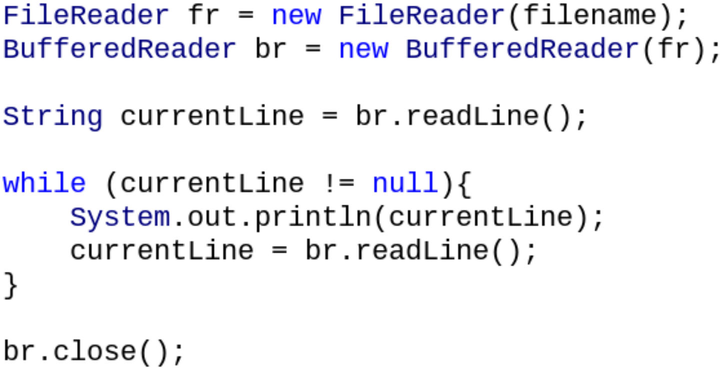
Java requires the use of two different classes to read from files.
Here's the creation of the objects:



```
String filename = "cats.txt";
//don't forget: import java.io.*;
FileReader fr = new FileReader(filename);
BufferedReader br = new BufferedReader(fr);
```

Java File Objects

How to read lines from the file:



```
FileReader fr = new FileReader(filename);
BufferedReader br = new BufferedReader(fr);

String currentLine = br.readLine();

while (currentLine != null){
    System.out.println(currentLine);
    currentLine = br.readLine();
}

br.close();
```

We'll use the `readLine()` method on the `BufferedReader` to get the next line from the file.

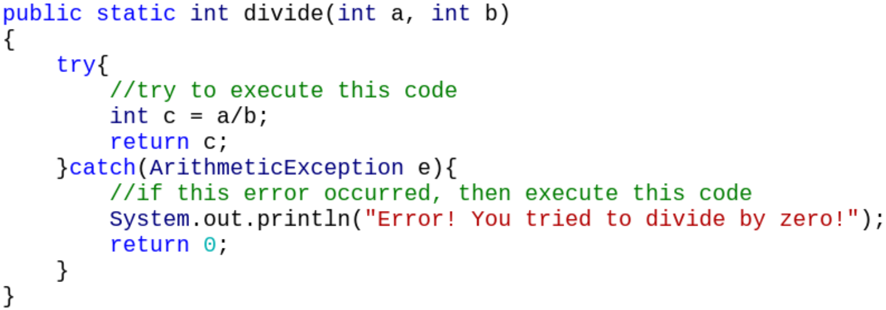
Section 4

IO Errors

Handling Errors

- Recall that we talked about handling errors before
- *Try* the operation and *catch* the error

Here we will have a **try/catch** block for division
If there's an error, then the catch block occurs



```java
public static int divide(int a, int b)
{
    try{
        //try to execute this code
        int c = a/b;
        return c;
    }catch(ArithmeticException e){
        //if this error occurred, then execute this code
        System.out.println("Error! You tried to divide by zero!");
        return 0;
    }
}
```

Handling the Exceptions

Java forces us to handle some `Exceptions` when we read from or write to a file.

- FileNotFoundException - if the file was missing when reading
- IOException - any other problem

File Reading

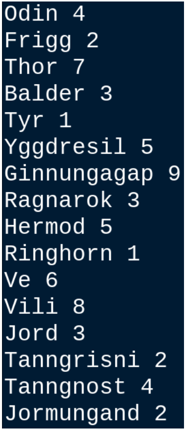
```
String filename = "cats.txt";

try{ //need import java.io.*; at top of file
    FileReader fr = new FileReader(filename);
    BufferedReader br = new BufferedReader(fr);

    String currentLine = br.readLine();

    while (currentLine != null){
        System.out.println(currentLine);
        currentLine = br.readLine();
    }

    br.close();
}catch (FileNotFoundException e){
    System.out.println("File not found: " + filename);
}catch (IOException e){
    System.out.println("Problem with file: " + filename);
}
```

Section 5

Instance Creation

The File to Read

Here's the file we're going to read
Has cat names and cat ages on each line



```
Odin 4
Frigg 2
Thor 7
Balder 3
Tyr 1
Yggdresil 5
Ginnungagap 9
Ragnarok 3
Hermod 5
Ringhorn 1
Ve 6
Vili 8
Jord 3
Tanngrisni 2
Tanngnost 4
Jormungand 2
```

Splitting a String

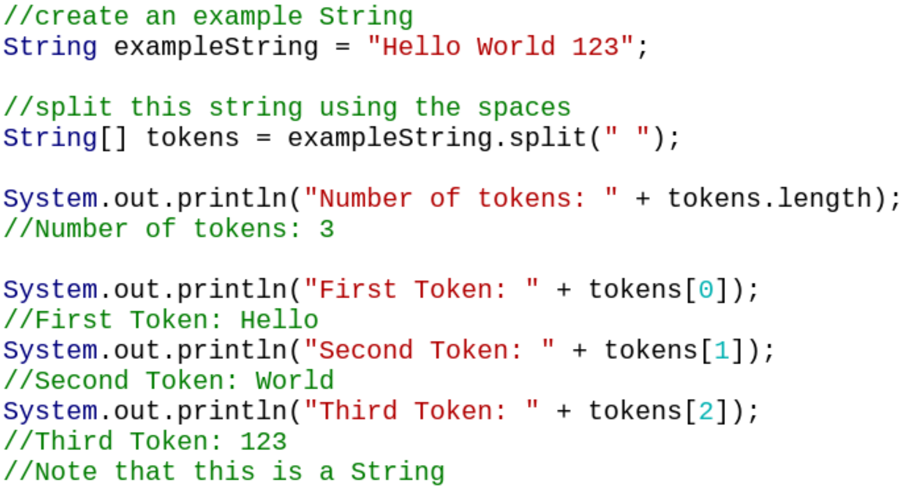
Let's look at the `split` method.

Takes a `String` and produces a `String[]`
``hello world 123'' → split on spaces → {``hello'', ``world'', ``123''}

This method takes a `String` and a *delimiter*, and returns a `String[]` where the `String` is split on that *delimiter*.

We can then parse the parts as we need

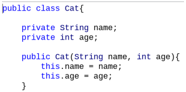
Split Example



```java
//create an example String
String exampleString = "Hello World 123";

//split this string using the spaces
String[] tokens = exampleString.split(" ");

System.out.println("Number of tokens: " + tokens.length);
//Number of tokens: 3

System.out.println("First Token: " + tokens[0]);
//First Token: Hello
System.out.println("Second Token: " + tokens[1]);
//Second Token: World
System.out.println("Third Token: " + tokens[2]);
//Third Token: 123
//Note that this is a String
```

The Cat Class

Let's create the `Cat` class to store their names and ages



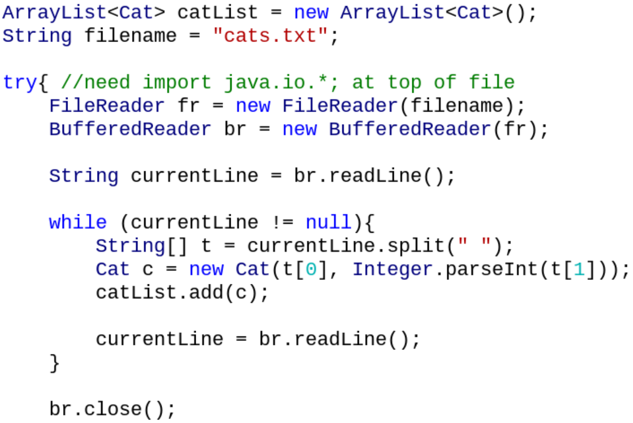
```
public class Cat{

    private String name;
    private int age;

    public Cat(String name, int age){
        this.name = name;
        this.age = age;
    }
```

Creating Instances

For each line in the file:

- First step: Split each `String` into a name and an age
- Second step: Create a new `Cat` out of the name and ages
- Third Step: Place the `Cat` in an `ArrayList`

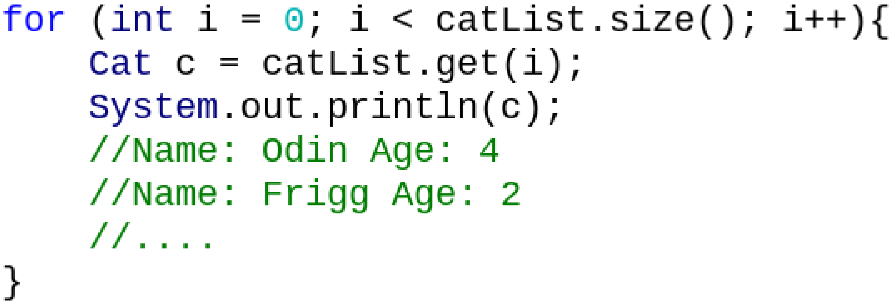
Creating Cats



```java
ArrayList<Cat> catList = new ArrayList<Cat>();
String filename = "cats.txt";

try{ //need import java.io.*; at top of file
    FileReader fr = new FileReader(filename);
    BufferedReader br = new BufferedReader(fr);

    String currentLine = br.readLine();

    while (currentLine != null){
        String[] t = currentLine.split(" ");
        Cat c = new Cat(t[0], Integer.parseInt(t[1]));
        catList.add(c);

        currentLine = br.readLine();
    }

    br.close();
```

Printing Cats

As before, we will loop through the array and print out the cats
This calls the toString method



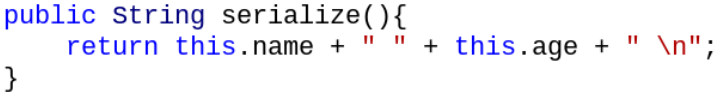
```
for (int i = 0; i < catList.size(); i++){
    Cat c = catList.get(i);
    System.out.println(c);
    //Name: Odin Age: 4
    //Name: Frigg Age: 2
    //....
}
```

Section 6

Writing to a File

Writing to a File

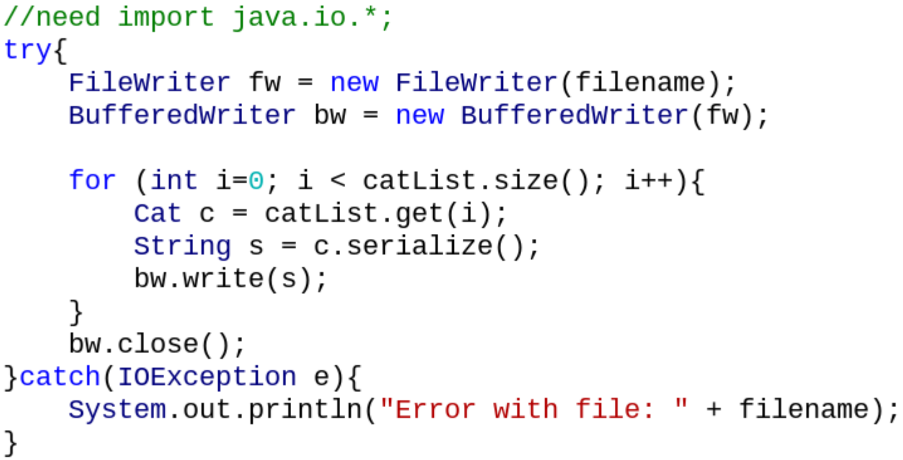
- Let's write some code to write the `ArrayList` of cats back to a file
- We'll write a method for `Cats` which defines how they should be written to a file
- Let's call this the `serialize` method
- Should give the same format as when we loaded the `Cats`



```java
public String serialize(){
    return this.name + " " + this.age + " \n";
}
```

Writing to a File

- Let's save all the Cats in our ArrayList.
- This involves the objects FileWriter and BufferedWriter.



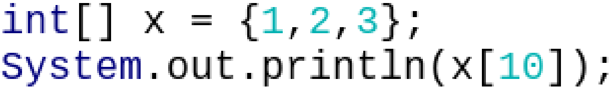
```
//need import java.io.*;
try{
    FileWriter fw = new FileWriter(filename);
    BufferedWriter bw = new BufferedWriter(fw);

    for (int i=0; i < catList.size(); i++){
        Cat c = catList.get(i);
        String s = c.serialize();
        bw.write(s);
    }
    bw.close();
}catch(IOException e){
    System.out.println("Error with file: " + filename);
}
```

Section 7

Exception Handling

Exceptions Handling

- Let's review some concepts in exception handling
- Exceptions happen when there is an error

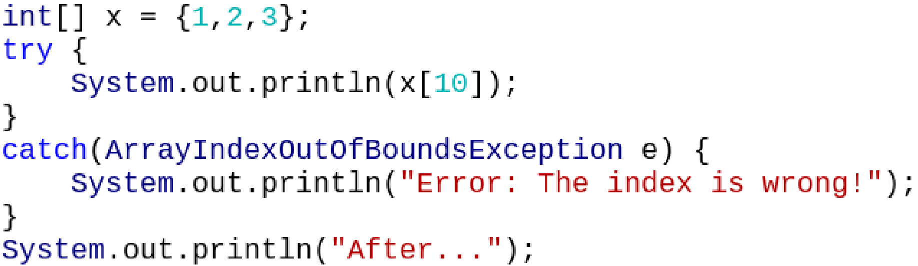


```
int[] x = {1,2,3};
System.out.println(x[10]);
```

The second line tries to access an invalid index. An `ArrayIndexOutOfBoundsException` will be thrown.

Exceptions Review

- Let's review how to *catch* these `Exceptions`
- We use a try/catch block to decide what code to run
- An exception that is caught will not cause the program to crash.

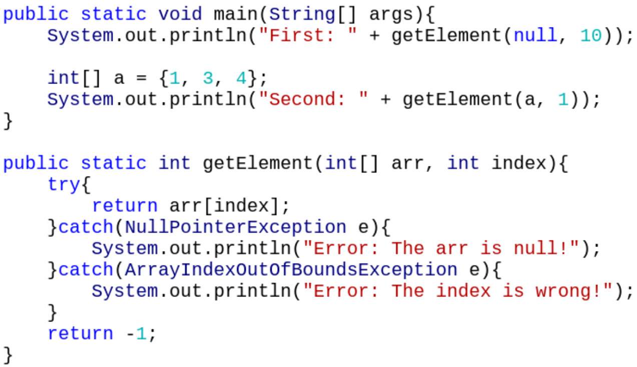


```
int[] x = {1,2,3};
try {
    System.out.println(x[10]);
}
catch(ArrayIndexOutOfBoundsException e) {
    System.out.println("Error: The index is wrong!");
}
System.out.println("After...");
```

```
Error:  The index is wrong!
After...
```

Exceptions Review

- Let's look at handling multiple errors

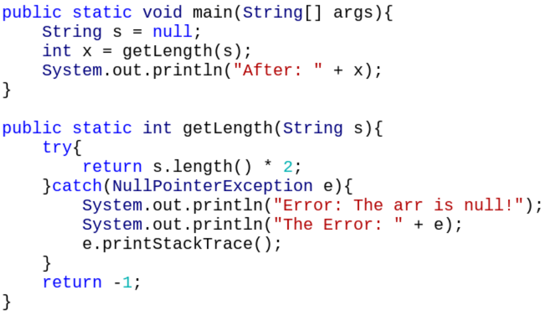


```
public static void main(String[] args){
    System.out.println("First: " + getElement(null, 10));

    int[] a = {1, 3, 4};
    System.out.println("Second: " + getElement(a, 1));
}

public static int getElement(int[] arr, int index){
    try{
        return arr[index];
    }catch(NullPointerException e){
        System.out.println("Error: The arr is null!");
    }catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Error: The index is wrong!");
    }
    return -1;
}
```

```
Error:  The arr is null!
First:  -1
Second:  3
```

Exceptions Review

- Let's review how to print information about the `Exception`
- Note that the program still continues



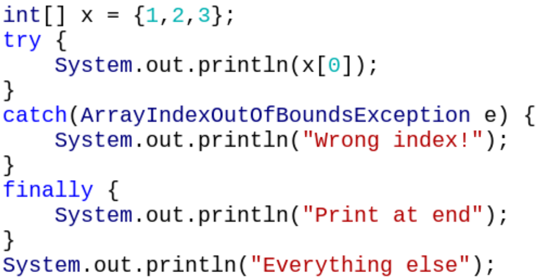
```java
public static void main(String[] args){
    String s = null;
    int x = getLength(s);
    System.out.println("After: " + x);
}

public static int getLength(String s){
    try{
        return s.length() * 2;
    }catch(NullPointerException e){
        System.out.println("Error: The arr is null!");
        System.out.println("The Error: " + e);
        e.printStackTrace();
    }
    return -1;
}
```

```
Error:  The arr is null!
The Error:  java.lang.NullPointerException
java.lang.NullPointerException
at ExceptionHandling.getLength(ExceptionHandling.java:13)
After:  -1
```

The Finally Block

- Let's see the *finally* block, just to finish off our discussion of `Exceptions`
- The finally block is attached to a try block, and it always executes.
- Even if one of the following happens:
  - an unexpected exception occurs in the try block
  - an exception occurs in the catch block
  - There's a return/continue/break statement in the try/catch block.
- You can have a finally even with just a try block (and no catch).

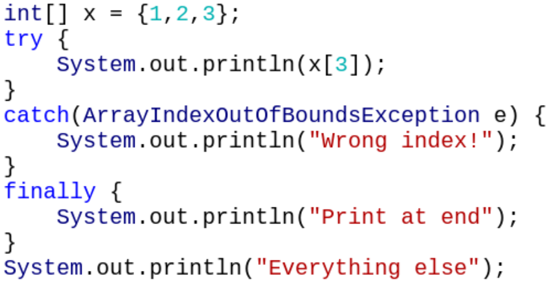
Exceptions Example



```
int[] x = {1,2,3};
try {
    System.out.println(x[0]);
}
catch(ArrayIndexOutOfBoundsException e) {
    System.out.println("Wrong index!");
}
finally {
    System.out.println("Print at end");
}
System.out.println("Everything else");
```

```
1
Print at end
Everything else
```

Exceptions Example 2



```java
int[] x = {1,2,3};
try {
    System.out.println(x[3]);
}
catch(ArrayIndexOutOfBoundsException e) {
    System.out.println("Wrong index!");
}
finally {
    System.out.println("Print at end");
}
System.out.println("Everything else");
```

```
Wrong index!
Print at end
Everything else
```

Section 8

Exception Types

Checked vs Unchecked Exceptions

There are two kinds of exceptions in Java:

- Checked
  - `Exception`
  - `IOException`
- Unchecked
  - `NullPointerException`
  - `ArrayIndexOutOfBoundsException`
  - `IllegalArgumentException`

Unchecked Exceptions

- These exceptions are not checked at compile-time
- Most exceptions are unchecked, and they can cause your code to crash at run-time
- You are not forced by the compiler to handle these exceptions
- It is up to the programmer to decide to catch the exceptions

Example: `int c = 12/0;` is a valid statement, and produces an unchecked `Exception`

Checked Exceptions

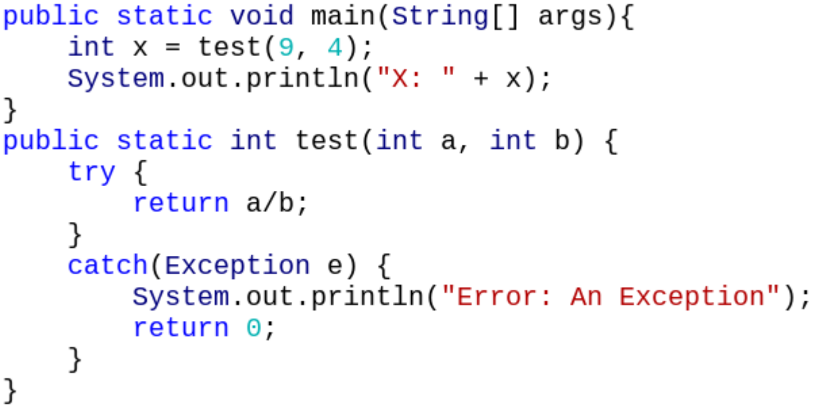
- These exceptions are checked at compile-time!
- The programmer is forced to handle these exceptions

There are two ways to handle these *checked exceptions*:

1. Use try/catch block to surround the code that might throw a checked exception
2. Specify that your method might throw a particular Exception
    - Force anyone using the method to handle the Exception

Option 1

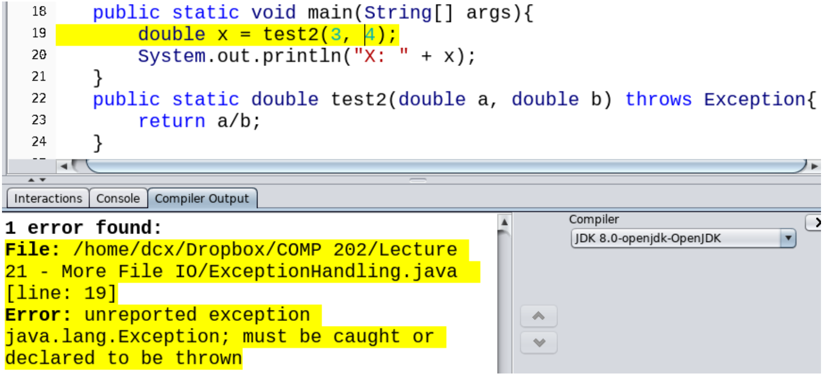
Surround the code that might throw an exception with a try/catch block.



```java
public static void main(String[] args){
    int x = test(9, 4);
    System.out.println("X: " + x);
}
public static int test(int a, int b) {
    try {
        return a/b;
    }
    catch(Exception e) {
        System.out.println("Error: An Exception");
        return 0;
    }
}
```

Option 2

Specify in the method header that there's an exception using the throws keyword followed by the type of the exception.
The method call then needs to be caught.

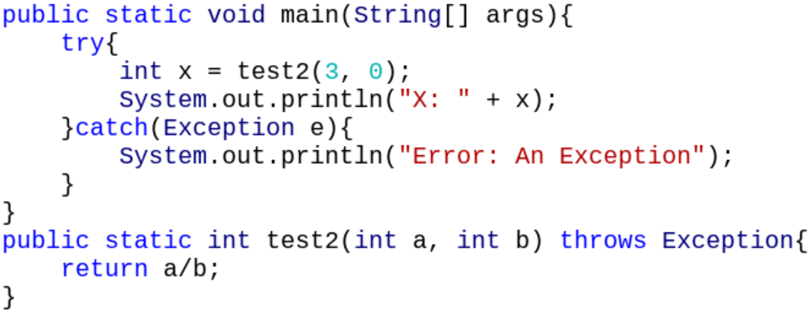


```
18      public static void main(String[] args){
19          double x = test2(3, 4);
20          System.out.println("X: " + x);
21      }
22      public static double test2(double a, double b) throws Exception{
23          return a/b;
24      }
```

```
1 error found:
File: /home/dcx/Dropbox/COMP 202/Lecture
21 - More File IO/ExceptionHandling.java
[line: 19]
Error: unreported exception
java.lang.Exception; must be caught or
declared to be thrown
```

Compiler: JDK 8.0-openjdk-OpenJDK

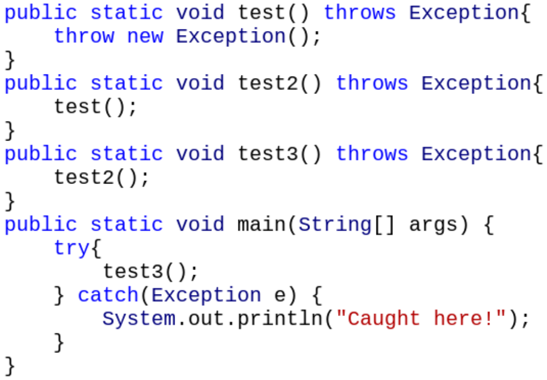
Option 2



```java
public static void main(String[] args){
    try{
        int x = test2(3, 0);
        System.out.println("X: " + x);
    }catch(Exception e){
        System.out.println("Error: An Exception");
    }
}
public static int test2(int a, int b) throws Exception{
    return a/b;
}
```

Chaining

- We can keep throwing the `Exception` to the calling method

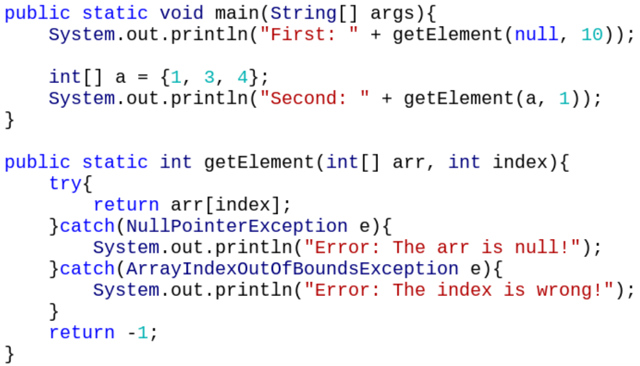


```java
public static void test() throws Exception{
    throw new Exception();
}
public static void test2() throws Exception{
    test();
}
public static void test3() throws Exception{
    test2();
}
public static void main(String[] args) {
    try{
        test3();
    } catch(Exception e) {
        System.out.println("Caught here!");
    }
}
```

- In this course, you're not allowed to throw Exceptions from the main method

Exceptions Review

- Let's look at handling multiple errors



```java
public static void main(String[] args){
    System.out.println("First: " + getElement(null, 10));

    int[] a = {1, 3, 4};
    System.out.println("Second: " + getElement(a, 1));
}

public static int getElement(int[] arr, int index){
    try{
        return arr[index];
    }catch(NullPointerException e){
        System.out.println("Error: The arr is null!");
    }catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Error: The index is wrong!");
    }
    return -1;
}
```

```
Error:  The arr is null!
First:  -1
Second:  3
```

Why Pass the Exception?

- Why would we throw an `Exception` to the calling method?
- We force the programmer to decide what to do when there's an error
  - This is more related to design of your program

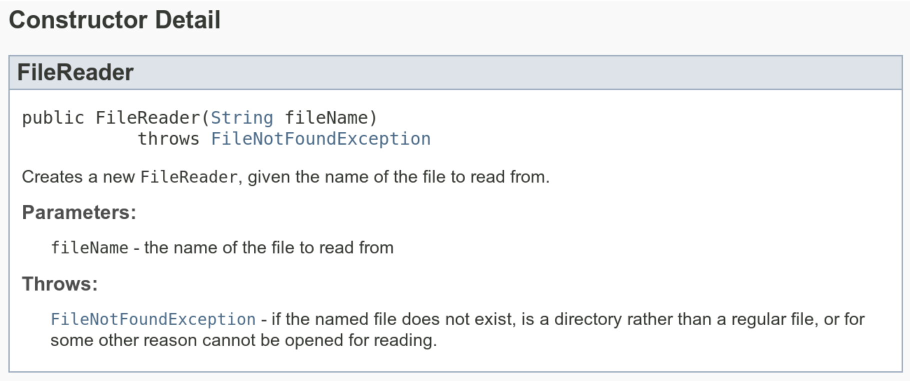
Section 9

File IO Exceptions

IOException and FileNotFoundException

- File IO operations are so error prone that the code containing them can throw an `IOException`

From the `FileReader` object:



Constructor Detail

FileReader

```
public FileReader(String fileName)
           throws FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read from.

Parameters:

`fileName` - the name of the file to read from

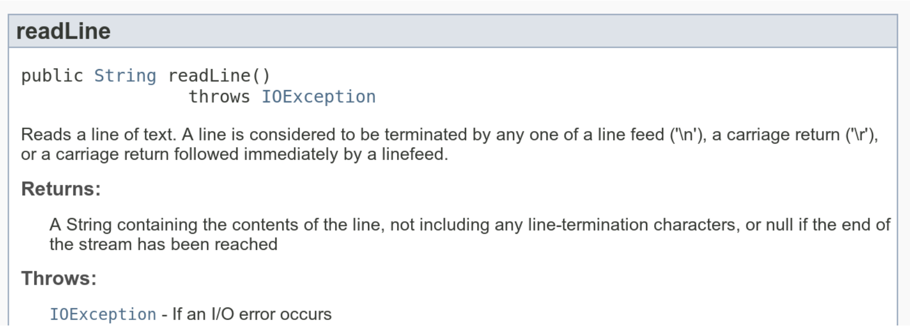
Throws:

`FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

IOException and FileNotFoundException

- The `readline` method of the `BufferedReader` object can throw an `IOException`

From the `BufferedReader` object:



readLine

```
public String readLine()
                throws IOException
```

Reads a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed.

Returns:

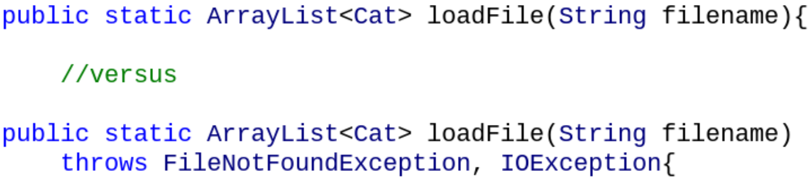
A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

Throws:

`IOException` - If an I/O error occurs

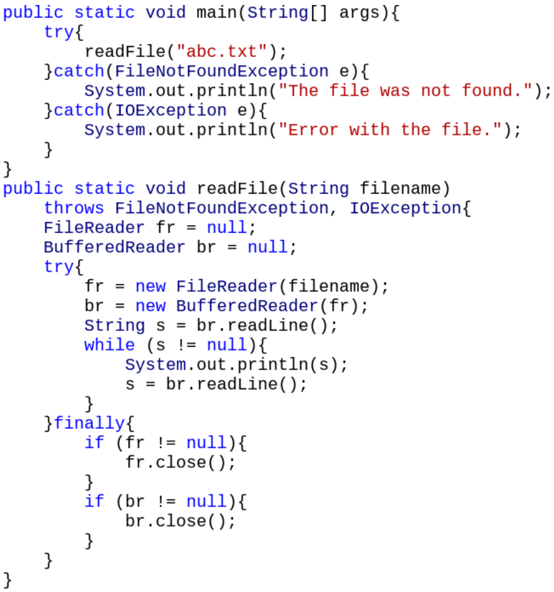
Handling These Exceptions

- Both these exceptions are checked exceptions. Therefore, you must handle them or you will get a compile-time error
- Put all File IO operations inside a try-block and have a catch block for the exceptions
- Or, pass on the exceptions by using throws in the header of the method



```
public static ArrayList<Cat> loadFile(String filename){

    //versus

public static ArrayList<Cat> loadFile(String filename)
    throws FileNotFoundException, IOException{
```

File Reading With Throws



```java
public static void main(String[] args){
    try{
        readFile("abc.txt");
    }catch(FileNotFoundException e){
        System.out.println("The file was not found.");
    }catch(IOException e){
        System.out.println("Error with the file.");
    }
}
public static void readFile(String filename)
    throws FileNotFoundException, IOException{
    FileReader fr = null;
    BufferedReader br = null;
    try{
        fr = new FileReader(filename);
        br = new BufferedReader(fr);
        String s = br.readLine();
        while (s != null){
            System.out.println(s);
            s = br.readLine();
        }
    }finally{
        if (fr != null){
            fr.close();
        }
        if (br != null){
            br.close();
        }
    }
}
```