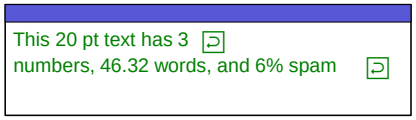
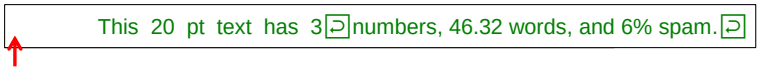
Text Input: reading multiple values

- When the user types into the text pane:



  - nothing happens until they type a newline (“enter”)
  - Then all the characters on the line are put into a buffer that the program can access



  - The program can access the buffer using the “UI.next…” methods:
    - UI.next() → “This” reads next “token” as a string
    - UI.nextInt() → 20 reads next “token” as an integer
    - UI.nextDouble() → ERROR! reads next token as a double
    - UI.nextLine() → “pt text has 3” reads up to next ↵ as a string
  - all the methods move the cursor forward, past what was read.

Text Input: reading multiple values

- If there is no input yet, the UI.next…() methods will just wait.

  ⇒ Always print a prompt to the user before you try to read!

- It is not safe to call UI.nextInt() or UI.nextDouble() unless you can be certain the next token is an integer / double!

- How can you tell?
  - UI.hasNextInt() → boolean true if next token is an integer
  - UI.hasNextDouble() → boolean true if next token is a double
  - UI.hasNext() → boolean true if there is a next token (always true for text pane)

next vs. nextLine()

- next(), nextInt(), nextDouble()
  - picks up any spaces, discards them,
  - picks up characters to make next “token” (until it reaches a space),
  - returns the token
    - next() returns it as a String
      nextInt() returns it as an int,
      nextDouble() returns it as a double.

- nextLine()
  - Picks up all the characters (including spaces) until it reaches end-of-line character,
  - throws away end-of-line, and
  - returns all the characters (including spaces) as a String.

Input with "next" methods

Method	What it does	Returns
next()	Read and return next token of user’s input	String
hasNext()	Returns true if there is another token in the user input. Waits for the user to type something if necessary.	boolean
nextInt() nextDouble()	Read the next token of the user's input. Return it as a integer if it is a number. Throws an exception if it is not a number.	int double
hasNextInt() hasNextDouble()	Returns true if next token in the input is an int / double. Waits for user to type something if necessary.	boolean
nextBoolean()	Read the next token of the user's input. Return true if it is "yes", "y", or "true", return false if it is "no", "n", or "false" (case insensitive). Throws an exception if it is anything else.	boolean
nextLine()	Read the remaining characters of the user's input up to (but not including) the next end-of-line and return them as a string. Reads and throws away the end-of-line character. If there are no characters on the line, then it returns an empty string ("").	String

Reading words from user

```
public void  countWordsBeforeThe( )  {
    int count = 0;
    UI.print("Enter some words: ");
    // loop, stopping when you get to 'the'
        //  read next token
        //  increment count
    UI.printf("You had %d words before 'the'. %n", count);
}
```


Reading words from user: BAD

```
public void  countWordsBeforeThe( )  {
    int count = 0;
    UI.print("Enter some words: ");
    while ( ! word.equalsIgnoreCase("the")   ) {
        String word = UI.next();
        count  = count + 1;
    }
    UI.printf("You had %d words before 'the'. %n", count);
}
```


Reading words from user: BAD

```
public void  countWordsBeforeThe( )  {
    int count = 0;
    UI.print("Enter some words: ");
    String word = UI.next();
    while ( ! word.equalsIgnoreCase("the")   ) {
        count  = count + 1;
    }
    UI.printf("You had %d words before 'the'. %n", count);
}
```


Reading words from user: Fixed

```
public void  countWordsBeforeThe( )  {
    int count = 0;
    UI.print("Enter some words: ");
    String word = UI.next();
    while ( ! word.equalsIgnoreCase("the")   ) {
        count  = count + 1;
        word = UI.next();
    }
    UI.printf("You had %d words before 'the'. %n", count);
}
```

read first word before loop

read next word at end of loop ("increment")

Alternate design: using break.

```
public void  countWordsBeforeThe( )  {
    int count = 0;
    UI.print("Enter some words: ");
    while ( true  ) {
        String word = UI.next();
        if ( word.equalsIgnoreCase("the") ){
            break;
        }
        count  = count + 1;
    }
    UI.printf("You had %d words before 'the'. %n", count);
}
```


Menu

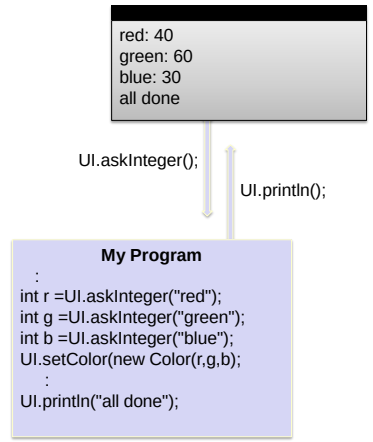
- Files

Admin
- assignments
- you are important to people!

Files

- The UI text pane window is transient:
  - Typing large amounts of input into the text pane is a pain!
  - It would be nice to be able to save the output of the program easily.
- Large amounts of text belong in files
- How can your program read from a file and write to a file?
- Writing to files is like writing to the UI text pane!
  - Use print, println, printf methods
  - But, need extra objects: File and PrintStream objects
- Reading from files is a bit different
  - Doesn't use "ask…" methods
  - Need to use "next…" methods
  - And need extra objects: File and Scanner objects

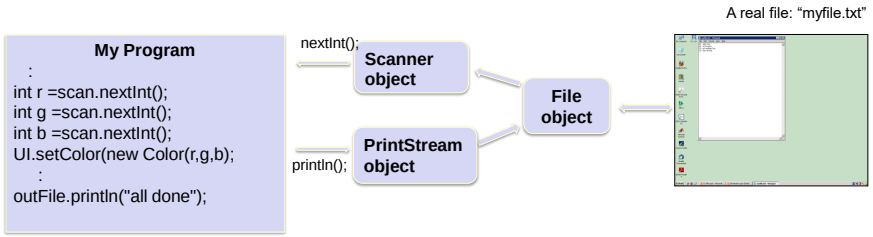
Text with the text pane



Text with Files

Needs several objects:

- Need File object to talk to the actual file on the disk.
- Need Scanner/PrintStream object to talk to the File object
- Program talks to the Scanner or PrintStream object.



Using a Scanner

- Scanner: a class in Java that allows a program to read input from a file (or any other source of characters such as a String, a socket, …)
- File: a class in Java that connects to an actual file on disk and get characters in and out of the file
- Program needs to make a File object an get the next token, or the next line

```
String fileName = "My File.txt";
File inFile = new File(fileName);
Scanner scan = new Scanner(inFile);
   ⋮
int r = scan.nextInt();
```

My File.txt

```
25 53
201 240 2 150
100 250 0
```


Scanner

- A Scanner breaks up the source into a sequence of chunks that the program can get, one at a time.
  - lines, (separated by the end-of-line characters)
  - tokens (separated by spaces, tabs, or end-of-line's)
- Program can read the next token (or the next line)

```
Scanner scan = new Scanner ( new File("My File.txt") );
while ( scan.hasNext() ){
    double radius = scan.nextDouble();
    UI.drawOval(X-radius, Y-radius, radius*2, radius*2);
}
```

My File.txt

```
25 53
201 240 2 150
100 250 0
```


Scanner "next" methods

Method	What it does	Returns
next()	Read and return next token	String
nextInt() nextDouble()	Read the next token. Return it as a number, if it is a number. Throws an exception if it is not a number.	int double
nextBoolean()	Read the next token. Return true if it is "true"; return false if it is "false". Throws an exception if it is anything else.	boolean
hasNext()	Returns true if there is another token	boolean
hasNextInt() hasNextDouble() hasNextBoolean()	Returns true if there is another token AND the next token is an int / double / Boolean	boolean
nextLine()	Read characters up to the next end-of-line and return them as a string. Reads and throws away the end-of-line character. If the first character is an end-of-line, then it returns an empty string ("").	String
close()	close the file	

Scanner methods.

- Scanner has a cursor that keeps track of where it is up to in the file.

ACCY308	Lecture	Tue	1030	1120	GBLT2
ACCY308	Lecture	Fri	1440	1530	GBLT3
ACCY308	Lecture	Tue	1640	1730	GBLT3
ACCY330	Lecture	Fri	1340	1430	RHLT2
ACCY330	Lecture	Wed	1240	1330	RHLT2
ACCY401	Comp-Lab	Mon	0930	1220	RWW402
ACCY401	Lecture	Mon	0930	1220	RWW220
ACCY402	Lecture	Wed	1240	1530	RWW311
ACCY412	Lecture	Wed	0830	1120	RWW311
ACCY421	Lecture	Thu	1340	1630	RWW311
ALIN201	Lecture	Mon	1200	1250	KK204
ALIN201	Lecture	Wed	1200	1250	KK204
ALIN201	Tutorial Wed	1610	1800	AM102	
ALIN301	Lecture	Tue	0900	0950	KK105
ALIN301	Lecture	Thu	0900	0950	KK105
ALIN301	Tutorial Thu	1610	1700	MY103	
ANTH101	Lecture	Mon	1310	1400	KKLT303
ANTH101	Lecture	Tue	1310	1400	KKLT303

Reading lines using Scanner:

```
/** Read lines from a file and print them to UI text pane. */
public void readFile(){
    File myfile = new File("input.txt");
    Scanner scan = new Scanner(myfile);
    UI.println("---------- input.txt ----------");
    while (scan.hasNext()){
        String line= scan.nextLine();
        UI.println(line);
    }
    UI.println("---------- end of input.txt ----------");
}
```

Missing bits to handle exceptions !!

- Almost right, but compiler complains!!!
- Dealing with files may "raise exceptions"

Files: handling exceptions

If a piece of code might raise an exception:

- Have to enclose it in a

```
try {
    …
}
catch (IOException e) {   … }
```

what to do

what to do if it goes wrong

```
public void readFile(){
    File myfile = new File("input.txt");
    try {
        Scanner scan = new Scanner(myfile);
        while (scan.hasNext()){
            String line = scan.nextLine();
            UI.println(line);
        }
        UI.println("---------- end of input.txt ----------");
    }
    catch (IOException e) { UI.println("File failure: " + e); }
```


Reading from files: example

```
/** Finds oldest person in file of ages and names. */
public void printOldest(String filename){
    try {
        Scanner scan = new Scanner(new File(filename));
        String oldest = "";
        int maxAge = 0;
        while (scan.hasNext()){
            int age = scan.nextInt();
            String name = scan.nextLine();
            if (age > maxAge) {
                maxAge = age;
                oldest = name;
            }
        }
        UI.printf("Oldest is %s (%d)%n", oldest, maxAge);
    } catch (IOException e) { UI.println("File failure: " + e); }
}
```

```
66 Marie Curie
48 James Clerk Maxwell
84 Isaac Newton
62 Aristotle
```

Read a token, then read rest of line

Reading data from a file

```
public void drawShapes(String filename){
    try {
        Scanner scan = new Scanner( new File(fileName) );
        while ( scan.hasNext() ){
            double left = scan.nextDouble();
            double top = scan.nextDouble();
            String shape = scan.next();
            int r = scan.nextInt();
            int g = scan.nextInt();
            int b = scan.nextInt();
            UI.setColor( new Color (r, g, b) );
            if (shape.equals("Oval") ){  UI.fillOval(left, top, WIDTH, HEIGHT);  }
            else {                       UI.fillRect(left, top, WIDTH, HEIGHT); }
        }
    }
    catch (IOException e) { UI.println("File failure: " + e); }
}
```

50.0 20.0 Oval 25 53 201
75.0 100.2 Rect 240 2 150
304.0 28.7 Oval 100 250 0

Stop at end of file

Reading all the values on the line

Do something with all the values

A common simple pattern

- File with one entity per line, described by multiple values:

```
while (sc.hasNext() ){
    String type = sc.next();
    double  cost = sc.nextDouble();
    int wheels = sc.nextInt();
    String colour = sc.next();
    String make = sc.next()
    if (wheels > 4) {
        ….
    }
    else {
        …
    }
}
```

bicycle 1025 2 green Giant
truck 120000 18 black Isuzu
car 26495 4 red Toyota

Read all the values into variables

process the values in the variables

Reading files line by line

If items have a varying number of values:
May need to read a line at a time, then process:

```
/**Adds up sales of item on each line of a file */
public void addCounts(){
  try {
    Scanner scan = new Scanner(new File("data.txt"));
    while (scan.hasNext()){
        String line = scan.nextLine();
        Scanner lineSc = new Scanner(line);
        int code = lineSc.nextInt();
        String item = lineSc.next();
        int lineTot = 0;
        while (lineSc.hasNextInt()) {
            lineTot = lineTot + lineSc.nextInt();
        }
        UI.printf("%s (%d): %d\n", item, code, lineTot);
    }
  } catch (IOException e) { UI.println("File failure: " + e); }
}
```

973 biscuits 27 33 15 4 9
731 cake 3 5 2
189 fruit 54 2 83 96
446 beans 1 3 2 5 3 4 7 2 5 1

Wrapping a Scanner around a String, Lets you "read" values from the String

Files that specify how big they are.

- Sometimes a data file may specify how many values it contains
- Can then use a "counted" loop to read the values:

```
try {
    Scanner scan = new Scanner( new File ( orderFileName ) );
    while ( scan.hasNext( ) ){
        String model = scan.nextLine();
        int count = scan.nextInt();
        int totalOrders = 0;
        int i = 0;
        while (i < count){
            totalOrders = totalOrders + scan.nextInt();
            i++;
        }
        UI.println( model + " had a total of " + totalOrders + " orders.");
    }
    scan.close();
} catch (IOException e) { UI.println("File error: " + e); }
```

Honda EV Orders.txt

Fit EV
5
35
270
15
380
89
Clarity
6
35
28
18
9
17
29

Files that specify how big they are.

- Image files: ppm format

"Magic number" – code for ppm files

read into variables:
int cols
int rows

width (number of columns of pixels) and height (the number of rows of pixels)

maximum colour value

red-green-blue of each pixel, in turn

```
P3
12 5
255
200 182 163 215 198 177 130 116 93 37 28 9 31 22 7
81 67 38 83 71 42 6 5 6 0 0 0 57 68 60 97 112 104
97 92 76 202 186 165 97 82 60 32 25 5 38 30 13 103 90
63 158 140 97 58 49 25 43 42 17 107 104 74 127 140
113 95 102 79 66 58 41 71 57 37 41 30 7 82 71 41 111
95 64 174 157 120 115 101 63 49 43 12 67 65 30 126
124 74 133 136 97 88 87 62 98 93 54 78 63 37 108 93
62 121 104 69 135 120 88 190 172 139 36 30 15 1 0 0
16 17 9 64 77 58 50 57 39 7 2 0 105 106 64 121 103 71
117 100 67 159 144 113 212 197 171 161 146 114 0 0 0
0 0 0 37 48 32 72 88 68 24 26 19 12 12 9 74 72 49
```

nested while loops to read colour of each pixel set colour of UI draw pixel.

Writing to a File

- Open a File object
- Wrap it in a new PrintStream object.
- Call print, println, or printf on it.
- Close the file

Just like printing to UI

PrintStream Object

File Object

```
try {
    PrintStream out = new PrintStream(new File("powers-table.txt"));
    int n=1;
    out.println("Number\tSquare\tCube");
    while ( n <= 1000 ) {
        out.printf("%4d \t%7d \t%10\n",  n,  n*n,  n*n*n);
        n = n+1;
    }
    out.close()
}
catch (IOException e) { UI.println("File error: " + e); }
```


Checking if files exist

- Can check that file exists before trying to read:

```
public void lineNumber(String fname){    /** Make a copy of a file with line numbers */
    File infile = new File(fname);
    if ( ! infile.exists()) { UI.println("The file " + fname + " doesn't exist");   return; }
    File outfile = new File("numbered-" +fname);
    try {
        Scanner sc = new Scanner ( infile );
        PrintStream out = new PrintStream(outfile);
        int lineNum = 0;
        while (sc.hasNext()) {
            out.println(lineNum + ": " + sc.nextLine() );
            lineNum++;
        }
        out.close();
        sc.close();
    } catch (IOException e) { UI.printf("File failure %s\n", e);}
}
```


Passing an open scanner

- First method: *Just opens and closes the file*

```
public void countTokensInFile(String fname){
    try {
        Scanner scan = new Scanner (new File(fname));
        int numTokens = this.countTokens(scan);
        UI.printf("%s has %d tokens\n", fname, numTokens);
        sc.close();
    } catch (Exception e) {UI.printf("File failure %s\n", e);}
}
```

scan:

Scanner-2543

File-872

973 biscuits 27
731 cake 3
189 fruit 54
446 beans 1

- Second Method: *Just reads from the scanner and counts*

```
public int countTokens (Scanner sc){
    int count = 0;
    while (sc.hasNext()) {
        sc.next();                // throws result away !
        count = count+1;
    }
    return count;
}
```

sc:

UIFileChooser

- So far, we've specified which file to open and read or write with a String.
  eg: File myfile = **new** File("input.txt");
- How can we allow the user to *choose* a file?
  - UIFileChooser class (part of ecs100 library, like UI)

Method	What it does	Returns
open()	Opens dialog box; User can select an **existing** file to open. Returns name of file or null if user cancelled.	String
open(String title)	Same as open(), but with specified title;	String
save()	Opens dialog box; User can select file (possibly new) to save to. Returns name of file, or null if the user cancelled.	String
save(String title)	Same as save(), but with specified title.	String

Using UIFileChooser methods: open

```
/** allow user to choose and open an existing file*/
String filename = UIFileChooser.open();
File myfile = new File(filename);
Scanner scan = new Scanner(myfile);
```

OR

```
Scanner scan = new Scanner(new File(UIFileChooser.open()));
```

```
/** allow user to choose and open an existing file,
   specifies a title for dialog box*/
File myfile = new File(UIFileChooser.open("Choose a file to copy"));
Scanner scan = new Scanner(myfile);
```

- Two "open" methods in one class?
  Overloading : two methods in the same class can have the same name as long as they have different parameters.

Using UIFileChooser methods: save

```
/** allow user to choose and save to a (new/existing) file*/
String filename = UIFileChooser.save();
File myfile = new File(filename);
PrintStream ps = new PrintStream(myfile);
```

OR

```
PrintStream ps = new PrintStream(new File(UIFileChooser.save()));
```

```
/** allow user to choose and save to a (new/existing) file,
   Specifies a title for dialog box */
File myfile = new File(UIFileChooser.save("File to save data in"));
PrintStream ps = new PrintStream(myfile);
```


Coercion

- Mismatching types:

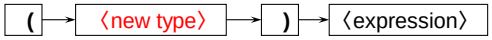
```
double num = scan.nextInt( );
int number = scan.nextDouble( );        ← Can't do this
double squareroot  =  Math.sqrt(25);    ← but sqrt wants double?
String name = "number-" + num;
```

- Java will "coerce" a value to the needed type if it can: eg
  - If a method needs a double and is given an int:
  - If an int is assigned to a double variable
  - If "adding" any value to a String
- But only if it does not lose any information:
  - WON'T coerce a double to an int
  - WON'T coerce a String to a number, or vice versa
    - except when "adding" a number to a String
  - WON'T coerce any object to a mismatching type
    - except when printing or "adding" to a String

Casting

- Where it makes sense to convert a value into another type, but some information may be lost...
- You can *sometimes* "cast" the value to the other type:

```
int number = (int) Math.sqrt(49.5);
float red =  (float) Math.random();
```



- casting a double to an int will lose the fractional part and may mess up the value if the number is too big!
- Not everything can be cast to everything else!
  - ~~Scanner scan = (Scanner) (new File("data.txt"));~~

More about static

```
/** Play a guessing game with the user*/
public class GuessingGame{
    public static final int maxValue = 40;
    public void GuessingGame(){
        UI.addButton("Play", this::playGame);
    }

    /** plays rounds of game*/
    public void playGame ( ){
        … …
    }

    /** main method */
    public static void main(String[ ] args){
        new GuessingGame();
    }
}
```

static means

"Belongs to class as a whole, Not to individual objects"

main method
- called when the program is run directly from Java
- used when running a jar files

Static methods:

- Static methods are methods that don't need an object:
  - Methods in the Math class are static methods:
    Math.min(…)
    Math.max(…)
    Math.random()
    Math.sqrt(…)
  - Methods in the UI class are static methods:
    UI.drawRect(…)
    UI.println(…)
    UI.askInt(…)

None of these methods need an object to be created first.

Methods are called on the class itself, not on an object of that class.