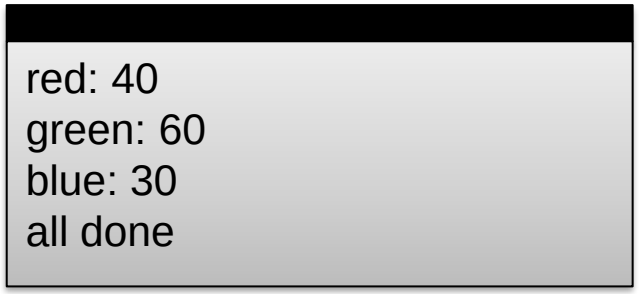
Files

- 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 a PrintStream object instead of UI
- Reading from files is a bit different
  - Doesn't use "ask…" methods
  - Lots of different ways of reading from files
  - We will just use a very simple one that reads a list of lines from a text file
  - We will use Scanner objects to break up the lines into separate values.

Text with the text pane



UI.askInt();

UI.println();

My Program

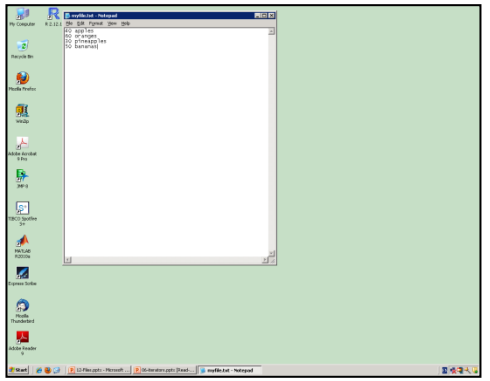
```
   :
int r =UI.askInt("red");
int g =UI.askInt("green");
int b =UI.askInt("blue");
UI.setColor(new Color(r,g,b));
      :
UI.println("all done");
```

Text with Files

Reading data from a file:

- Read the file into a list of lines (Strings).
- Either
  - Use the lines directly, or
  - Use a Scanner object to get the values out of the lines

A real file: “mydata.txt”



Text with Files

Reading data from a file:

- Read the file into a list of lines (Strings).
- Use a Scanner object to get the values out of the lines

Writing data to a file:

- Use a PrintStream object.

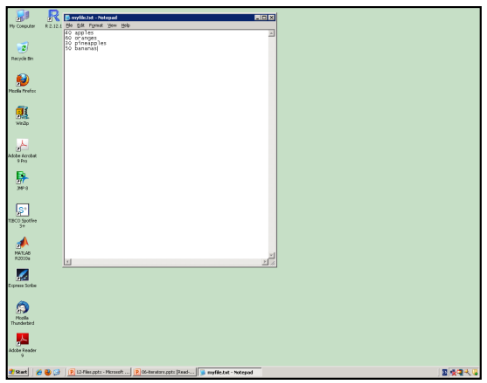
My Program

```
   :
int r =scan.nextInt();
int g =scan.nextInt();
int b =scan.nextInt();
UI.setColor(new Color(r,g,b));
      :
outFile.println("all done");
```

print…

PrintStream object

A real file: “output.txt”



Reading lines from a file:

```
/** Read lines from a file and print them to UI text pane. */
public void displayFile(String fileName){
   try {
        List<String> allLines = Files.readAllLines(Path.of(fileName));
        for (String line : allLines){
            UI.println(line);
        }
   } catch (IOException e) { UI.println("File failure: " + e); }
}
```

Missing bits to handle exceptions !!

what to do

what to do if it goes wrong

- Files.readAllLines(Path.of(….)) reads every line of the file into a List of Strings.
- Almost right, but compiler complains!!!
- Dealing with files may “raise exceptions”
- Need a **try** { …… } **catch** (…){ … }

Writing to a file:

```
/** Read lines from a user and print them to a file. */
public void makeFile(String filename){
    ArrayList<String> lines = UI.askStrings("Type in file contents:");
    try {
        PrintStream outfile = new PrintStream(filename);
        for (String line : lines) {
            outfile.println(line);
        }
        outfile.close();
    } catch (IOException e) { UI.println("File failure: " + e); }
}
```

- PrintStreams work just like printing to UI
- Close the file when finished.
- Need a try { … } catch (…){….} around printing to files also.

Writing to a file, using UIFileChooser

```
/** Read lines from a user and print them to a file. */
public void makeFile(){
    ArrayList<String> lines = UI.askStrings("Type in file contents:");
    String filename = UIFileChooser.save("Filename to save to");
    try {
        PrintStream outfile = new PrintStream(filename);
        for (String line : lines) {
            outfile.println(line);
        }
        outfile.close();
    } catch (IOException e) { UI.println("File failure: " + e); }
}
```

- UIFileChooser.save("….prompt….") lets the user choose a (possibly new) file.

Copying a file

```
/** Read lines from one file and print them to another file. */
public void copyFile(){
    String fromFile = UIFileChooser.open("File to copy");
    String toFile = UIFileChooser.save("Filename to save to");
    try {
        List<String> lines = Files.readAllLines(Path.of(fromFile));
        PrintStream outfile = new PrintStream(toFile);
        for (String line : lines) {
            outfile.println(line);
        }
        outfile.close();
    } catch (IOException e) { UI.println("File failure: " + e); }
}
```

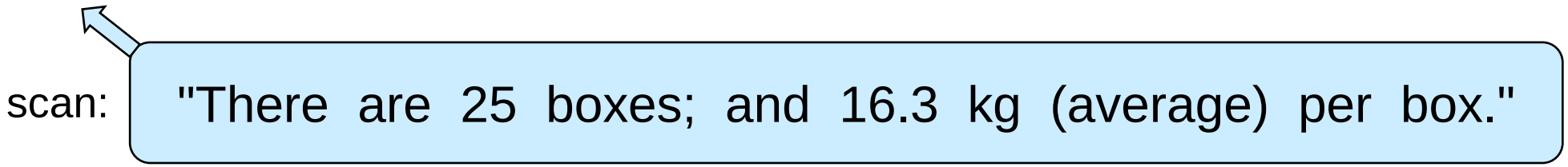
- UIFileChooser.open("….prompt….") lets the user choose an existing file.

Doing more with data in a file:

```
/** Find all lines in a file containing a search word. */
public void findWordInFile(){
    String fileName = UIFileChooser.open("Choose file to search");
    String word = UI.askString("Word to search for");
    try {
        List<String> allLines = Files.readAllLines(Path.of(fileName));
        int lineNumber = 1;
        for (String line : allLines){
            if (line.contains(word)){
                UI.printf("Found %s on line %d: %s\n",  word,  lineNumber,  line);
            }
            lineNumber++;
        }
    } catch (IOException e) { UI.println("File failure: " + e); }
}
```

Scanners

- Scanner: a class in Java that allows a program to read values out of a String (or any other source of characters…)
- Gives the values one at a time



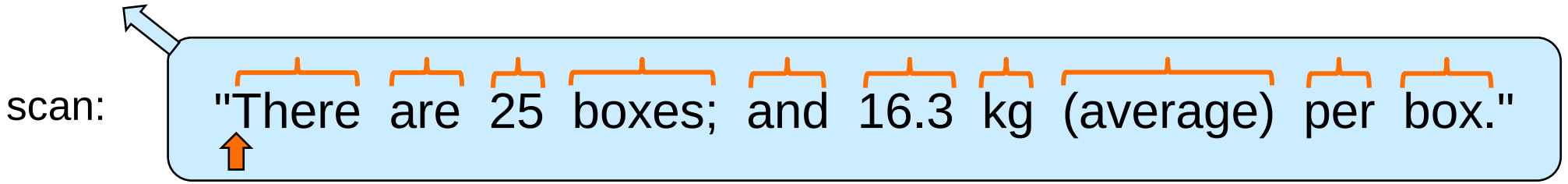
To get a Scanner:

- Create a new Scanner object, passing it the source:

```
Scanner scan = new Scanner("There are 25 boxes; and 16.3 kg (average) per box.");
Scanner sc = new Scanner(UI.askString("Enter some text");
String line = ….
Scanner lineSc = new Scanner(line);
```

Scanners

- A Scanner breaks up the source string into a sequence of tokens, separated by spaces or tabs.



- Token: a word, a number, or … any sequence of non-space characters.

- A Scanner provides the tokens, one at a time, using the .next…() methods:

  scan.next() ⇒ next token as a string

  scan.nextInt() ⇒ next token as an int (error if next token is not an integer)

  scan.nextDouble() ⇒ next token as a double (error if next token is not a number)

- Each call to .next…() moves the "cursor" to the end of the token.

Reading Tokens from a Scanner

- If you know how many tokens in the Scanner, you can just pull them out:

```
Scanner scan = new Scanner ("4447 quince 11.45");
String PLU = scan.next();
String product = scan.next();
String price = scan.next();

Scanner scan = new Scanner ("This string has (exactly) 10 tokens:  a-b-c-d & 9.0  #10");
for (int i = 0; i <10; i++){
    String tok = scan.next();
    UI.printf("Token %d : %s\n", i, tok);
}
```

- Tokens are Strings (whether they look like words, numbers, other…)
- Can only take them out in order

Reading from a Scanner

- If you know the number of tokens and their types, can extract them.
  - Eg, if the string has an integer, a word, and a double:

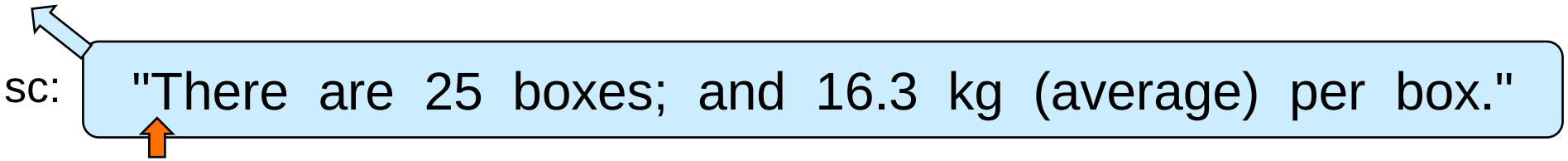
```
Scanner scan = new Scanner ("4447 quince 11.45");
int PLU = scan.nextInt();
String product = scan.next();
double price = scan.nextDouble();

Scanner scan = new Scanner ("4430 pineapple 6.82");
double PLU = scan.nextDouble();
double product = scan.nextDouble();
int price = scan.nextInt();
```

- Safe to read a number as a String, or an integer as a double.
- Not safe to read a non-number as a number, or a double as an int

Reading from a Scanner

- If the number of tokens in a scanner is unknown,
  How can you tell when to stop?

```
Scanner sc = new Scanner (UI.askString("Enter a line of text"));
```



- Scanner lets you ask if there is another token using the .hasNext() method:

  sc.hasNext() ⇒ true or false: is there another token in the scanner?

- Can use a while loop with a Scanner:

```
while (sc.hasNext()){
    String word = sc.next();
    ….
}
```

Reading from a Scanner

- If the types of the tokens in a Scanner can vary,
  How can you tell what type they are?

- Scanner lets you "peek" at the next token using the .hasNext…() methods:

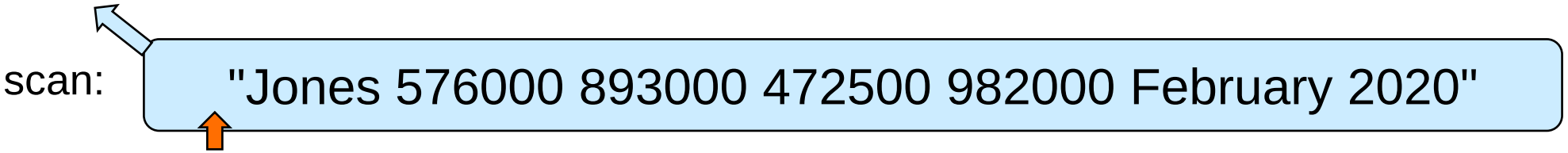
  scan.hasNextInt() ⇒ true or false: is there another token AND is it an integer?

  scan.hasNextDouble() ⇒ true or false: is there another token AND is it a number?

```
Scanner sc = new Scanner (UI.askString("Enter some tokens"));
int total = 0;
while (sc.hasNext()){
    if (sc.hasNextInt()){   // if the next token is an integer, read it and add to total
        int num = sc.nextInt();
        total = total + num;
    }
    else {                  // if next token is not an integer, read it and throw it away
        sc.next();
    }
}
```

Reading from a Scanner

- More unknown values:




```
Scanner scan = new Scanner (line);
String salesperson = scan.next();
double total = 0;
while (scan.hasNextDouble()){
        total = total + scan.nextDouble();
}
String month = scan.next();
int year = scan.nextInt();
```

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

Files and Scanners

If a file has lines, each with several values in it:

- Wrap each line from the file in a Scanner, and
- Read the values from the Scanner.

```
List<String> allLines = Files.readAllLines(Path.of("image.pxm"));
for (String line : lines){
    Scanner scan = new Scanner (line);
    UI.setColor(new Color (scan.nextInt(), scan.nextInt(), scan.nextInt()));
    UI.fillRect(x, y, 2,2);
    x = x+2;
    if (x > RIGHT) {
        x = LEFT;
        y = y+2;
    }
}
```

image.pxm

```
25 53 201
240 2 150
100 250 0
240 220 220
....
```

Files and Scanners

If a file has lines, each with varying number of values in it:

- Wrap each line from the file in a Scanner, and
- Read the values from the Scanner.

```
List<String> allLines = Files.readAllLines(Path.of("numbers.txt"));
double max = Double.NEGATIVE_INFINITY;
for (String line : lines){
    Scanner scan = new Scanner (line);
    while (scan.hasNextDouble()){
        double num = scan.nextDouble();
        if (num > max) {
            max = num;
        }
    }
}
```

numbers.txt

```
25 53 201 240
2 150
100 250 0 240
220 220 15
....
```

Files and Scanners

- If each line has fixed number of values of different types:

```
try {
    List<String> allLines = Files.readAllLines(Path.of("flights.txt"));
    for (String line : lines){
        Scanner scan = new Scanner (line);
        int PLU = scan.nextInt();
        String product = scan.next();
        double price = scan.nextDouble();
        ….. // do something with the values
    }
} catch (IOException e) { UI.println("File failure: " + e); }
```

fruit.txt

```
4447 quince  11.45
4430 pineapple 6.82
4041 red-plum 5.99
4416 D'Anjou-pear  5.44
4011 Banana     2.99
```

A common simple pattern

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

```
List<String> lines = Files.readAllLines(Path.of(filename));

for (String line : lines){
    Scanner sc = new Scanner(line);
    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

Processing values from a line

```
try {
    List<String> lines = Files.readAllLines(Path.of(filename));
    for ( String line : lines  ){
        Scanner sc = new Scanner(line);
        double left = sc.nextDouble();
        double top = sc.nextDouble();
        double wd = sc.nextDouble();
        double ht = sc.nextDouble();
        String shape = sc.next();
        int r = sc.nextInt();
        int g = sc.nextInt();
        int b = sc.nextInt();

        UI.setColor( new Color (r, g, b) );
        if (shape.equals("Oval") ){  UI.fillOval(left, top, wd, ht);  }
        else {                       UI.fillRect(left, top, wd, ht); }
    }
} catch (IOException e) { UI.println("File failure: " + e); }
```

Diagram.txt

```
50.0 20.0 10.3 7.8 Oval 25 53 201
75.0 100.2 16.9 12.0 Rect 240 2 150
304.0 28.7 25.0 51.5 Oval 100 250 0
...
```

extracting all the values on the line

Do something with all the values