Scanning Tokens

Tokens

- When a `Scanner` reads input, it separates it into "tokens".
  - … at least when using methods like `nextInt()`.
  - `nextInt()` takes the next token from the input, tries to convert it an `int`, and returns it.
  - If the conversion can't be done (by `Integer.parseInt()`?), it throws an exception.

Delimiting Tokens

- You may have noticed that `Scanner` will use any whitespace to separate values.
  - i.e. it doesn't matter if numbers in a file are separated by a space or a newline.
- By default, `Scanner` uses any whitespace to separate ("delimit") tokens.
  - i.e. space, tab, newline, etc.
- But this isn't always what you want.

Changing Delimiters

- What if you want to read chunks of a file (or user input) separated by something else?
  - e.g. "`1,2,3,4,5,6,7`", as seven tokens separated by commas
- It is possible to configure the way a `Scanner` delimits tokens.
- The `useDelimiter` method returns a **new** `Scanner` object with different delimiter string.

`useDelimiter()`

- e.g.

```
Scanner filein = new
                  Scanner(…).useDelimiter(",");
```

- Creates a `Scanner` that looks for the string `","` between tokens that are read.
  - Every time it sees a comma, that will be the end of the token.
  - The character after the comma is the start of the next token.

Example

- This reads a file with comma-separated tokens.
  - Just reads the tokens as strings and prints them.

```
Scanner filein = new Scanner(
                   new File("file.txt")
                   ).useDelimiter(",");

while ( filein.hasNext() ) {
  System.out.printf("(%s)", filein.next() );
}
```

Inflexible Delimiters

- With `useDelimiter(",")`, the `Scanner` looks strictly for a single comma character.
  - Surrounding spaces or other characters are part of the tokens, not the delimiters.
- e.g. In this file, the third token contains a space; the fourth contains a newline.

```
1,2, 3,
4,5
```

- tokens: “`1`”, “`2`”, “` 3`”, “`\n4`”, “`5`”

Inflexible Delimiters

- In that case, using `nextInt()` on the `Scanner` would fail.
  - “` 3`” is not a valid integer: integers don’t have spaces in them.
- The solution: include the space or newline in the delimiter, so it doesn’t end up in the token.
  - We can’t just use `useDelimiter(", ")` since not every separator includes the space.

More Flexible Delimiters

- What we need to do is express “a comma and then maybe some whitespace” as the delimiter.
  - we want to count any of these:

```
"," ", " ",\n" ", \n" ",\t"
```

- We have no way to do that at the moment.
  - We can only specify a single literal string.
  - … but the default behaviour (any combination of whitespace) is like what we want, so there’s hope.
- Hmmm…