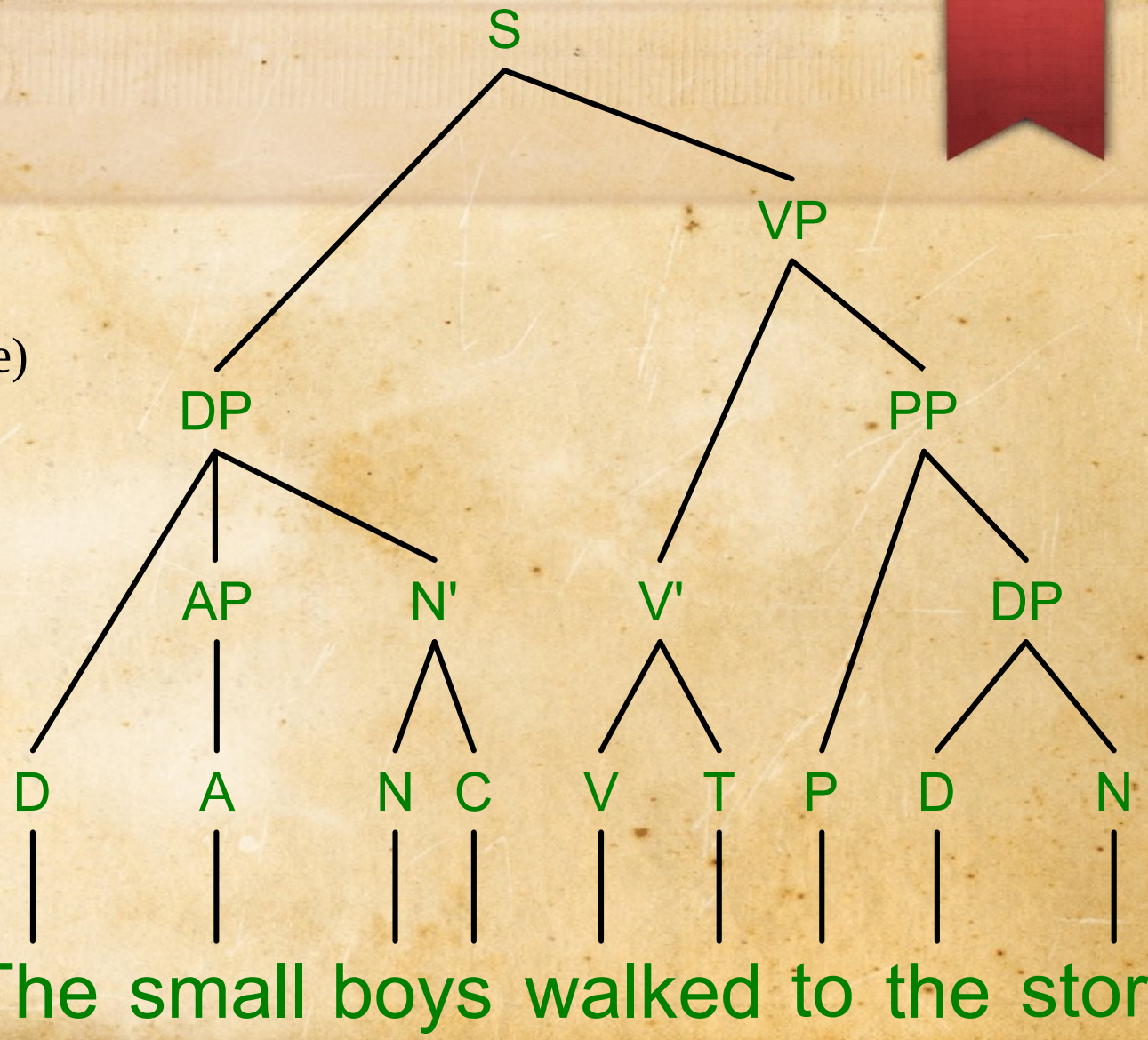
Regular Expressions

Lecture Contents

- Parsing
- Parsing in Java
- Regular Expressions

Parsing

- Parse (*verb*)
  - analyze (a sentence) into its parts and describe the syntactic roles of the parts



Parsing

- Parse (*verb*)
  - analyze (a sentence) into its parts and describe the syntactic roles of the parts
  - ***Computing***: analyze (some text) into logical syntactic components, typically in order to test conformability to a logical grammar

Parsing in Java

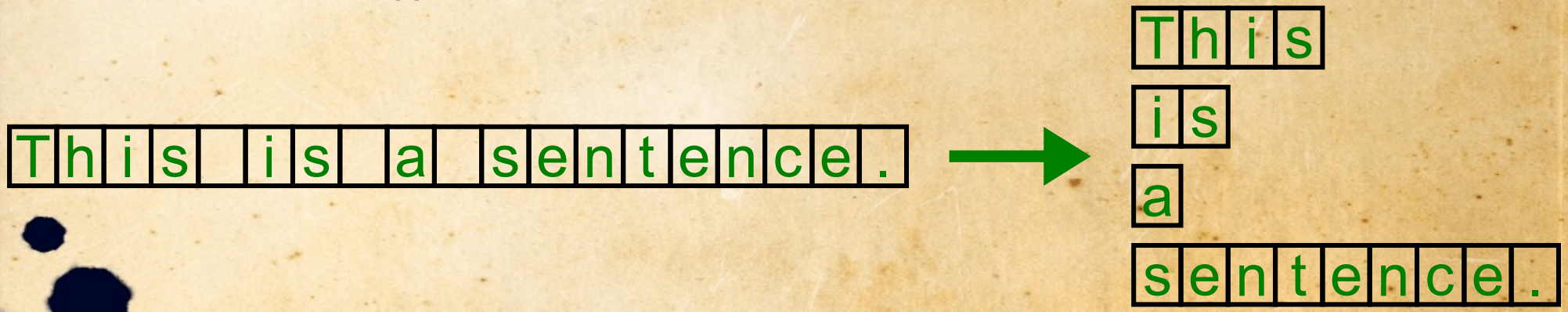
- `class StringTokenizer`
- Method `String.split()` and `String.matches()`
- `class Pattern` and `class Matcher`
- `class Scanner`
- Write your own custom parser

Parsing in Java

- `class StringTokenizer`
  - a ***legacy class*** (don’t bother learning unless necessary)
- Write your own custom parser
  - if speed is *really* required

Parsing in Java

- Method `String.split()` and `String.matches()`
  - for simple `String` splitting and pattern matching
  - each takes a ***regular expression*** `String` as a parameter
  - `split()` returns type `String[]`
  - `matches()` returns `boolean`



Parsing in Java

- `class Pattern` and `class Matcher`
  - matches a ***regular expression*** `String`
  - `class Pattern` sets up the `matcher`
  - method `matcher.find()` looks for the next sequence
  - method `matcher.group()` returns the matched sequence
  - does not process the entire `String` at once
    - May be better than `String.split()` and `String.match()` for long `String` variables

Parsing in Java

- Class `Scanner`
  - most flexible
  - can take input from types: `String`, `File`, `InputStream`, `Readable`
  - pre-defined patterns, or match a ***regular expression***

What are *Regular Expressions*?

- ***Regular Expressions*** allow searching for specific patterns of text.
  - Not programming language specific
  - Also used by application software
  - Unix-like OS command line: `grep`
    - **g**et **r**egular **e**xpression and **p**rint
  - A bit tedious to learn, but *very* powerful

Examples of Use

- Match a *string*:
  - `"hi"`
    - "**hi**"
    - "c**hi**cken"
    - "t**hi**s is **hi**s **hi**story"

Examples of Use

- Match a *character* from a defined set:
  - `"[abc]"`
    - `"hi"`
    - `"chicken"`
    - `"this was his history"`

Examples of Use

- Match a *character* from a defined set with a range of values:
  - `"[abc0-9]"`
    - `"hi"`
    - `"chicken"`
    - `"add 6 and 9"`

Examples of Use

- Match a *character* from a defined set with a range of values:
  - `"[a-zA-Z]"`
    - **"hi"**
    - **"chicken"**
    - **"add** 6 **and** 9"
    - **"Hello World**!!"

Examples of Use

- Match a *character* from a defined set with a range of values:
  - `"[a-zA-Z0-9]"`
    - **`"hi"`**
    - **`"chicken"`**
    - **`"add 6 and 9"`**
    - **`"Hello World!!"`**

Examples of Use

- Match a *character* from a defined set with a range of values:
  - `"Q[0-9]"`
    - `"Read **Q1** again"`
    - `"Read q1 again"`
    - `"Answer **Q3** through **Q7**"`
    - `"Answer Q12 but not **Q3**"`

Examples of Use

- Match a *string* of *characters* from a defined set:

syntax	meaning
?	match the previous character 0 or 1 times
*****	match the previous character 0 or more times
+	match the previous character 1 or more times
{*n*}	match the previous character exactly *n* times
{*n*,}	match the previous character at least *n* times
{,*m*}	match the previous character at most *m* times
{*n*,*m*}	match the previous character between *n* and *m*

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'+'` in a *regex* means “one or more of the previous character”.
  - `"[a-zA-Z]+"`
    - **"hi"**
    - **"chicken"**
    - **"Car**13"
    - **"Hello**!!"

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'+'` in a *regex* means “one or more of the previous character”.
  - `"[a-zA-Z]+[0-9]"`
    - `"hi"`
    - `"chicken"`
    - `"ch1cken"`
    - `"Race13!!"`

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character sequence `"{n}"` in a *regex* means “match the previous character *n* times”:
  - `"[a-zA-Z]{5}"`
    - `"hi!"`
    - `"four!"`
    - `"Hello!!"`

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character sequence **"{*n*}"** in a *regex* means “match the previous character *n* times”:
  - `"[a-zA-Z]{5}"`
    - `"hi!"`
    - `"four!"`
    - **"Hello!!"**

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'?'` in a *regex* means “zero or one of the previous character”.
  - `"[+-]?[0-9]+"`
    - `"35"`
    - `"-45a"`
    - `"ch1cken"`
    - `"Race+13!!"`

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'?'` in a *regex* means “zero or one of the previous character”.
  - `"[+–]?[0-9]+"`
    - "**35**"
    - "**–45**a"
    - "ch**1**cken"
    - "Race+**13**!!"

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'.'` in a *regex* means “any one character”.
  - `"b.d"`
    - `"bad"`
    - `"bed"`
    - `"abide"`
    - `"ab!de"`
    - `"bead"`
    - `"The lamb didn’t run."`

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'.'` in a *regex* means “any one character”.
  - `"b.d"`
    - "**bad**"
    - "**bed**"
    - "a**bid**e"
    - "a**b!d**e"
    - "bead"
    - "The lam**b** **d**idn’t run."

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'+'` in a *regex* means “one or more of the previous character”.
  - `"Hell+o"`
    - `"Helo!"`
    - `"Hello!"`
    - `"Hellllllllo!"`
    - `"hello”`

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'+'` in a *regex* means “one or more of the previous character”.
  - `"Hell+o"`
    - `"Helo!"`
    - `"**Hello**!"`
    - `"**Hellllllo**!"`
    - `"hello”`

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'.'` in a *regex* means “any one character”.
  - `"[a-zA-Z]{1}.+[0-9]"`
    - `"A9"`
    - `"A10"`
    - `"A%0"`
    - `"Apple"`
    - `"iPhone14"`
    - `"iPhone14max"`
    - `"Apple 3e"`
    - `"123o–4"`

Examples of Use

- Match a *string* of *characters* from a defined set:
  - The character `'.'` in a *regex* means “any one character”.
  - `"[a-zA-Z]{1}.+[0-9]"`
    - `"A9"`
    - **`"A10"`**
    - **`"A%0"`**
    - `"Apple"`
    - **`"iPhone14"`**
    - **`"iPhone14`**`max"`
    - **`"Apple 3`**`e"`
    - `"123`**`o-4`**`"`

The `Scanner` Class and Regular Expressions

- When using the Scanner class:
  - Method `hasNext(String` ***pattern***`)`
    - returns `true` only if the *regular expression* given in pattern matches the **entire** next token.
    - **`"[A-Za-z]"`** returns true for a single letter
    - **`"[0-9]"`** returns true for a single digit
    - **`"[A-Za-z]+"`** returns true for a word with only letters

The Scanner Class and Regular Expressions

```
public static String getUserWord(String prompt) {
   while(true) {
      System.out.print(prompt);
      if(in.hasNext("[A-Za-z]+")) {
         String s = in.next();
         in.nextLine(); // remove rest of line from buffer
         return s;
      } else {
         in.nextLine(); // remove invalid input line from buffer
      }
   }
}
```

The Scanner Class and Regular Expressions

```
public static String getUserWord(String prompt) {
   return getUserPattern(prompt, "[A-Za-z]+");
}

public static String getUserPattern(String prompt, String pattern) {
   while(true) {
      System.out.print(prompt);
      if(in.hasNext(pattern)) {
         String s = in.next();
         in.nextLine(); // remove rest of line from buffer
         return s;
      } else {
         in.nextLine(); // remove invalid input line from buffer
      }
   }
}
```

The Scanner Class and Regular Expressions

```
public static char getUserLetter(String prompt) {
   return getUserPattern(prompt, "[A-Za-z]").charAt(0);
}

public static char getUserDigit(String prompt) {
   return getUserPattern(prompt, "[0-9]").charAt(0);
}

public static String getUserWord(String prompt) {
   return getUserPattern(prompt, "[A-Za-z]+");
}

public static String getUserPattern(String prompt, String pattern) {
      while(true) {
         System.out.print(prompt);
         if(in.hasNext(pattern)) {
            String s = in.next();
            in.nextLine(); // remove rest of line from buffer
            return s;
```

Regular Expressions