Languages and Grammars

For Programming Assignment

Languages and grammars

- (formal) **language**: A set of words or symbols.
- **grammar**: Syntax and structure of a language
  - describes language *syntax* (rules) but not *semantics* (meaning)
  - can be used to generate strings from a language, or to determine whether a given string belongs to a given language
- Natural language : a language that has developed naturally through use

Backus-Naur (BNF)

- **Backus-Naur Form (BNF)**: A syntax for describing language grammars in terms of transformation *rules*, of the form:

  <**symbol**> **: :** = <**expression**> | <**expression**> ... | <**expression**>

  - **terminal**: A fundamental symbol of the language.
  - **non-terminal**: A high-level symbol describing language syntax, which can be transformed into other non-terminal or terminal symbol(s) based on the rules of the grammar.

  <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

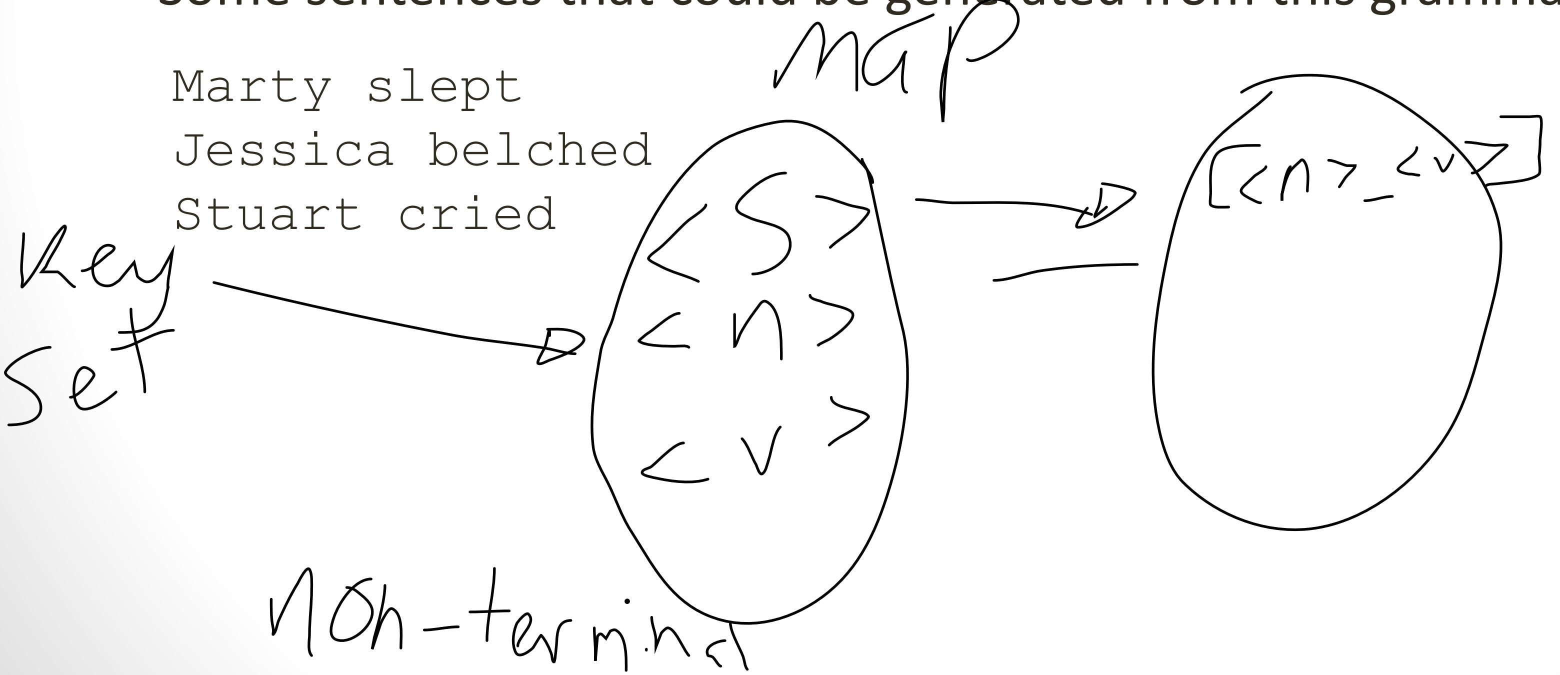
  - developed by two Turing-award-winning computer scientists in 1960 to describe their new ALGOL programming language

An example BNF grammar

```
<s>::=<n> <v>
<n>::=Marty | Victoria | Stuart | Jessica
<v>::=cried | slept | belched
```

- Some sentences that could be generated from this grammar:

```
Marty slept
Jessica belched
Stuart cried
```



BNF grammar version 2

```
<s>::=<np> <v>
<np>::=<pn> | <dp> <n>
<pn>::=Marty | Victoria | Stuart | Jessica
<dp>::=a | the
<n>::=ball | hamster | carrot | computer
<v>::=cried | slept | belched
```

- Some sentences that could be generated from this grammar:

```
the carrot cried
Jessica belched
a computer slept
```

how to pick random: check length - rad numb 0-length - 1

```
Random r = new Random ();
r.nextInt(4)
```

BNF grammar version 3

```
<s>::=<np> <v>
<np>::=<pn> | <dp> <adj> <n>
<pn>::=Marty | Victoria | Stuart | Jessica
<dp>::=a | the
<adj>::=silly | invisible | loud | romantic
<n>::=ball | hamster | carrot | computer
<v>::=cried | slept | belched
```

- Some sentences that could be generated from this grammar:

```
the invisible carrot cried
Jessica belched
a computer slept
a romantic ball belched
```

Grammars and recursion

```
<s>::=<np> <v>
<np>::=<pn> | <dp> <adjp> <n>
<pn>::=Marty | Victoria | Stuart | Jessica
<dp>::=a | the
<adjp>::=<adj> <adjp> | <adj>
<adj>::=silly | invisible | loud | romantic
<n>::=ball | hamster | carrot | computer
<v>::=cried | slept | belched
```

- Grammar rules can be defined *recursively*, so that the expansion of a symbol can contain that same symbol.
  - There must also be expressions that expand the symbol into something non-recursive, so that the recursion eventually ends.

Grammar, final version

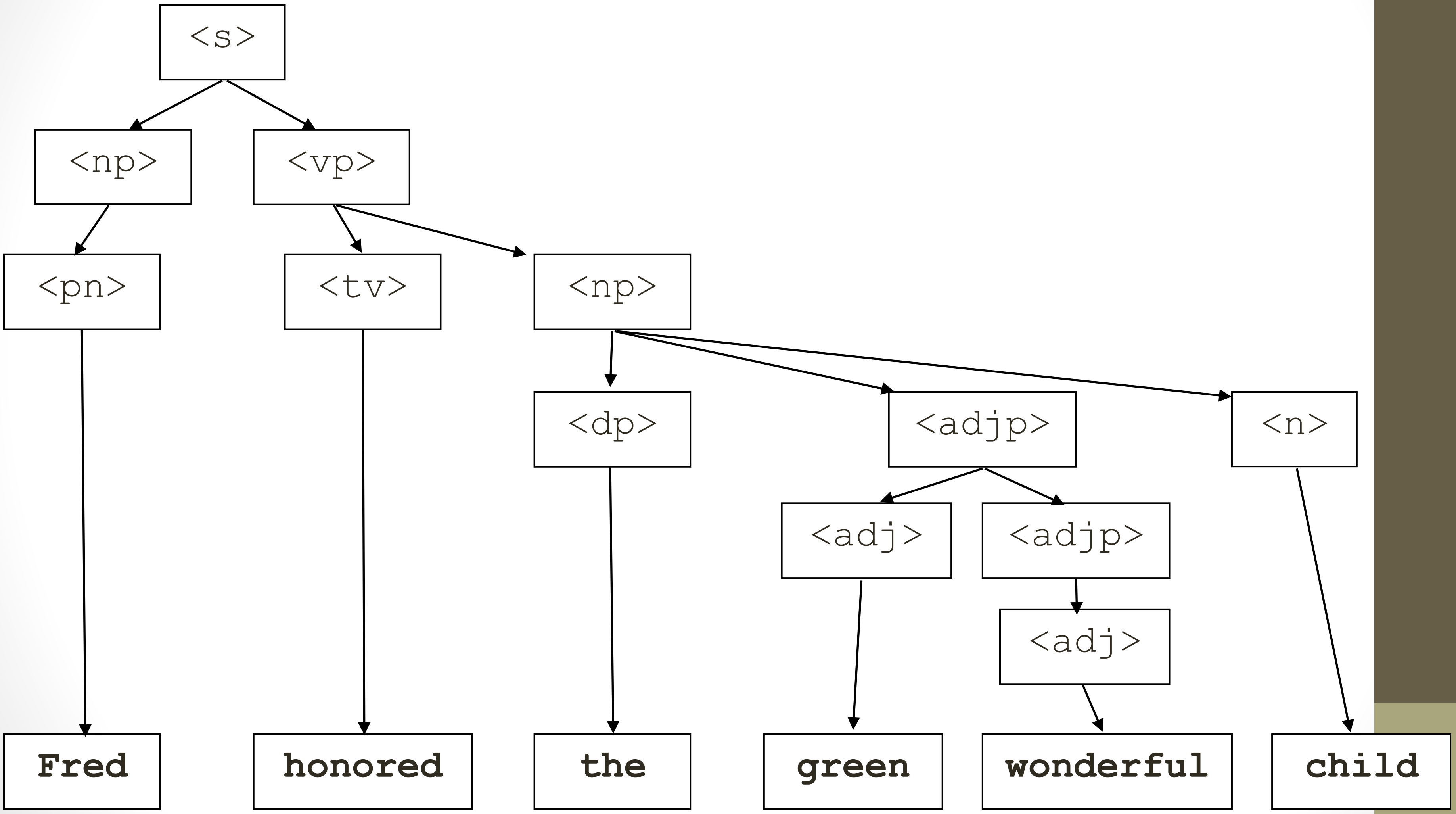
```
<s>::=<np> <vp>
<np>::=<dp> <adjp> <n>|<pn>
<dp>::=the|a
<adjp>::=<adj>|<adj> <adjp>
<adj>::=big|fat|green|wonderful|faulty|sublimina
  l
<n>::=dog|cat|man|university|father|mother|child
<pn>::=John|Jane|Sally|Spot|Fred|Elmo
<vp>::=<tv> <np>|<iv>
<tv>::=hit|honored|kissed|helped
<iv>::=died|collapsed|laughed|wept
```

- Could this grammar generate the following sentences?

```
Fred honored the green wonderful child
big Jane wept the fat man fat
```

- Generate a random sentence using this grammar.

Sentence generation



Another example

```
ROBOTNAME ::= 2D | 3D
3D   ::= NUM      2D       LET
2D   ::= 2D   2D   |   LET NUM   |   LET LET
LET  ::= A | B | R | T | P | O | D
NUM  ::= 1 | 2 | 3 | 5 | 4 | 7 | 9 | 0 | 8
```

Is R2D2 a valid ROBOTNAME?

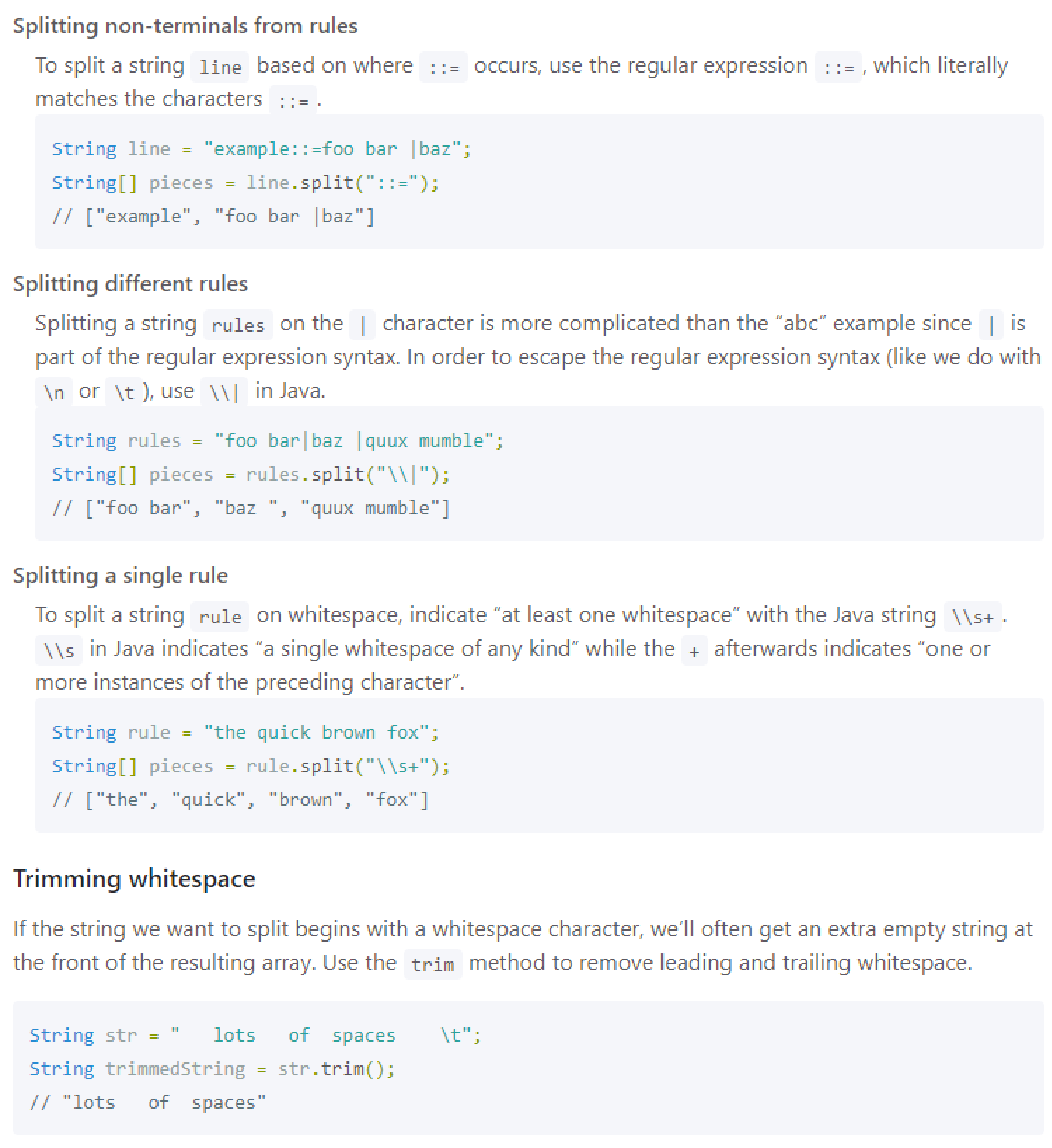
What about C3PO?

What about BB8?

Make your own name:

Another example

```
webaddress ::= name . domain | front . name . domain
name ::= comcast | newegg | google | amazon
front  ::= www
domain ::= com | net | biz | domain . country
country ::= au | de | aus | mx
```



Splitting non-terminals from rules

To split a string `line` based on where `::=` occurs, use the regular expression `::=`, which literally matches the characters `::=`.

```
String line = "example::=foo bar |baz";
String[] pieces = line.split("::=");
// ["example", "foo bar |baz"]
```

Splitting different rules

Splitting a string `rules` on the `|` character is more complicated than the “abc” example since `|` is part of the regular expression syntax. In order to escape the regular expression syntax (like we do with `\n` or `\t`), use `\\|` in Java.

```
String rules = "foo bar|baz |quux mumble";
String[] pieces = rules.split("\\|");
// ["foo bar", "baz ", "quux mumble"]
```

Splitting a single rule

To split a string `rule` on whitespace, indicate “at least one whitespace” with the Java string `\\s+`. `\\s` in Java indicates “a single whitespace of any kind” while the `+` afterwards indicates “one or more instances of the preceding character”.

```
String rule = "the quick brown fox";
String[] pieces = rule.split("\\s+");
// ["the", "quick", "brown", "fox"]
```

Trimming whitespace

If the string we want to split begins with a whitespace character, we’ll often get an extra empty string at the front of the resulting array. Use the `trim` method to remove leading and trailing whitespace.

```
String str = "   lots   of  spaces    \t";
String trimmedString = str.trim();
// "lots   of  spaces"
```