Module 1: Introduction to JAPE

Topics covered in this module

- What is JAPE?

- Parts of the rule: LHS and RHS

- How to write simple patterns

- How to create new annotations and features

- Different operators

- Different matching styles

- Macros

What is JAPE and what is it good for?

What is JAPE?

- a Jolly And Pleasant Experience :-)
- Specially developed pattern matching language for GATE
- Each JAPE rule consists of
  - LHS which contains patterns to match
  - RHS which details the annotations to be created
- JAPE rules combine to create a phase
- Rule priority based on pattern length, rule status and rule ordering
- Phases combine to create a grammar

Limitations of gazetteers

- Gazetteer lists are designed for annotating simple, regular features
- Some flexibility is provided, but this is not enough for most tasks
  - recognising e-mail addresses using just a gazetteer would be impossible
  - but combined with other linguistic pre-processing results, we have lots of annotations and features
- POS tags, capitalisation, punctuation, lookup features, etc can all be combined to form patterns suggesting more complex information
- This is where JAPE comes in.

JAPE example

- A typical JAPE rule might match all university names in the UK, e.g. "University of Sheffield"

- The gazetteer might contain the word "Sheffield" in the list of cities

- The rule looks for specific words such as "University of" followed by the name of a city.

- This wouldn't be enough to match all university names, but it's a start.

- Later, we'll see how we can extend this kind of rule to cover other variations.

Simple JAPE Rule

```
Rule: University1

(

 {Token.string == "University"}

 {Token.string == "of"}

 {Lookup.minorType == city}

):orgName

-->

:orgName.Organisation =

  {kind = "university", rule = "University1"}
```

Parts of the rule

Rule: University1 ← Rule Name

```
(
 {Token.string == "University"}
 {Token.string == "of"}
 {Lookup.minorType == city}
):orgName
```
← LHS

```
-->
```

```
:orgName.Organisation =
   {kind = "university",
    rule = "University1"}
```
← RHS

8

LHS of the rule

Rule: University1

```
(
  {Token.string == "University"}
  {Token.string == "of"}
  {Lookup.minorType == city}
):orgName
-->
```

- LHS is everything before the arrow

- It describes the pattern to be matched, in terms of annotations and (optionally) their features

- Each annotation is enclosed in a curly brace

Matching a text string

- Everything to be matched must be specified in terms of annotations

- To match a string of text, use the "Token" annotation and the "string" feature

`{Token.string == "University"}`

- Note that case is important in the value of the string

- You can combine sequences of annotations in a pattern

`{Token.string == "University"}`
`{Token.string == "of"}`
`{Lookup.minorType == city}`

Labels on the LHS

- For every combination of patterns that you want to create an annotation for, you need a label

- The pattern combination that you want to label is enclosed in round brackets, followed by a colon and the label

- The label name can be any legal name you want: it's only used within the rule itself

```
(
 {Token.string == "University"}
 {Token.string == "of"}
 {Lookup.minorType == city}
) :orgName
```

Operators on the LHS

Traditional Kleene and other operators can be used

| OR

* zero or more occurrences

? zero or one occurrence

+ one or more occurrences

```
({Lookup.minorType == city}|
 {Lookup.minorType == country})
```

Delimiting operator range

- Use round brackets to delimit the range of the operators

```
({Lookup.minorType == city}|
 {Lookup.minorType == country}
)+
```

One or more cities or countries in any order and combination

is not the same as

```
({Lookup.minorType == city}|
 ({Lookup.minorType == country})+
)
```
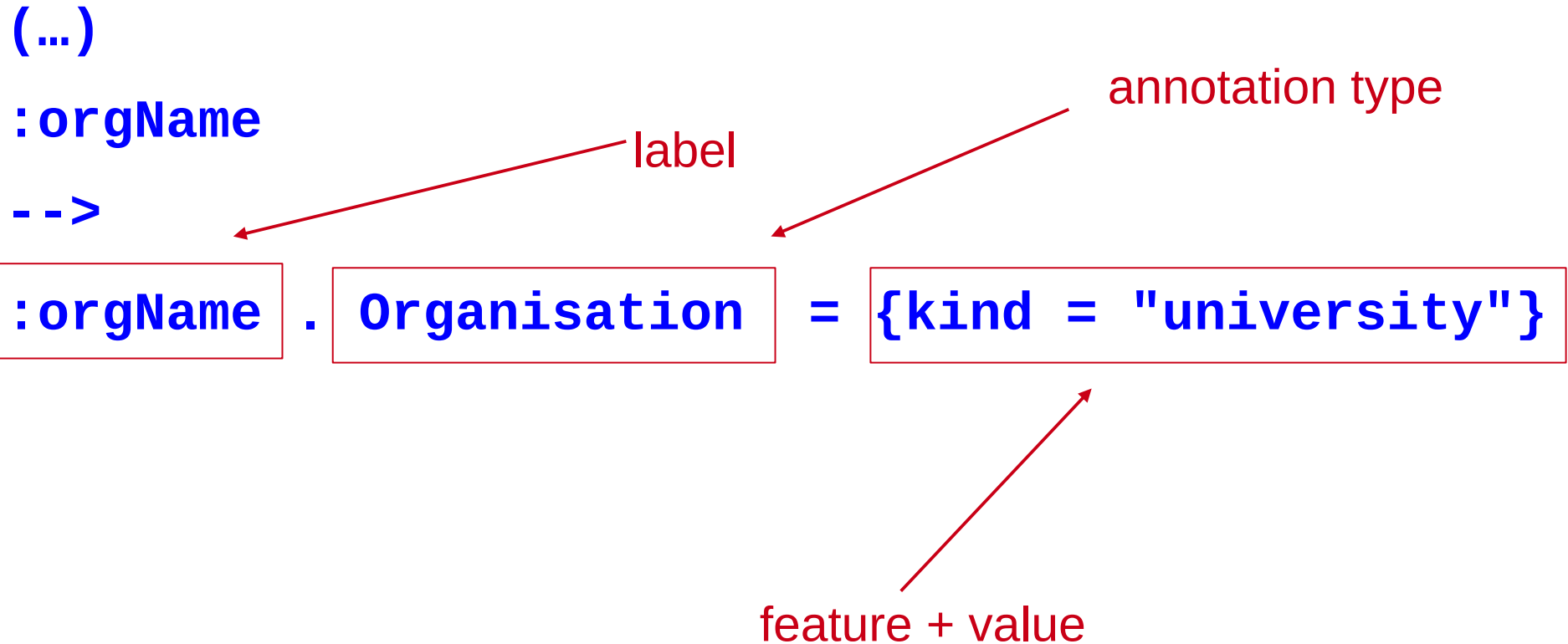
One city OR one or more countries

JAPE RHS

```
Rule: University1
(
  {Token.string == "University"}
  {Token.string == "of"}
  {Lookup.minorType == city}
):orgName
-->
:orgName.Organisation =
  {kind = "university", rule = "University1"}
```

Breaking down the RHS

(…)

:orgName

-->

label

annotation type

| **:orgName** | . **Organisation** | = **{kind = "university"}** |

feature + value

Labels

- The label on the RHS must match a label on the LHS

```
(

 {Token.string == "University"}

 {Token.string == "of"}

 {Lookup.minorType == city}

) :orgName

-->

:orgName .Organization = {kind = organization}
```

- This is so we know which part of the pattern to attach the new annotation to

Go label crazy...

- You can have as many patterns and actions as you want

- Patterns can be consecutive, nested, or both!

- Patterns cannot overlap

```
(
 ({Token.string == "University"}):uniKey
 {Token.string == "of"}
 ({Lookup.minorType == city}):cityName
) :orgName
-->
```

17

Multiple patterns and labels

- We can have several actions on the RHS corresponding to different labels.

- Separate the actions with a comma

```
(
 ({Token.string == "University"})
 {Token.string == "of"}
 ({Lookup.minorType == city}) : cityName
) :orgName
-->
:cityName. Location  = {kind = city},
:orgName.Organization = {kind = university}
```

Patterns and actions

- A pattern does not have to have a corresponding action

- If there's no action, you don't need to label it

- Patterns specified will normally be consumed (more on this later)

- Here, we want to add a special annotation for university towns

```
(
  {Token.string == "University"}
  {Token.string == "of"}
)
  ({Lookup.minorType == city}):  cityName
-->
:cityName. Location  = {kind = university_town}
```

Annotations and Features

- The annotation type and features created can be anything you want (as long as they are legal names)

- They don't need to currently exist anywhere

- Features and values are optional, and you can have as many as you like

- All the following are valid:

```
:orgName.Organization = {}

:orgName.Organization = {kind=university}

:orgName.Organization =
  {kind=university, rule=University1}

:fishLabel.InterestingFishAnnotation = {scales=yes}
```

Exercise: annotation types and features

- Remove any existing applications and processing resources that you have loaded in GATE

- Load ANNIE and remove the JAPE grammar and orthomatcher

- Load the grammar *university1.jape*, add it to your application, and run on the text *university1.txt*

- View the results

- Now open the grammar *university1.jape* in your favourite text editor and change the name of the annotation type created

- Save the file, then reinitialise the grammar in GATE and run the application again. View your new annotation.

- Try changing the name of the features, removing features, and adding new ones, and adding multiple labels

More complex RHS

- So far we've just shown RHS syntax involving JAPE

- You can also use any Java on the RHS instead, or as well

- This is useful for doing more complex things, such as

  - Iterating through a list of annotations of unknown number

  - Checking a word has a certain suffix before creating an annotation

  - Getting information about one annotation from inside another annotation

- More complex Java on the RHS will be taught later in this module

JAPE Headers

- Each JAPE file must contain a set of headers at the top

```
Phase: University

Input: Token Lookup

Options: control = appelt
```

- These headers apply to all rules within that grammar phase
- They contain Phase name, set of Input annotations and other Options

JAPE Phases

- A typical JAPE grammar will contain lots of different rules, divided into phases
- The set of phases is run sequentially over the document
- You might have some pre-processing, then some main annotation phases, then some cleanup phases
- Each phase needs a name, e.g `Phase: University`
- The phase name makes up part of the Java class name for the compiled RHS actions, so it must contain alphanumeric characters and underscores only, and cannot start with a number

JAPE Phases (2)

- Rules in the same phase *compete for input*
- Rules in separate phases run independently
- One phase can use annotations created by previous phases
- Instead of loading each JAPE grammar as a separate transducer in GATE, you can combine them in a *multiphase transducer*
- A multiphase transducer chains a set of JAPE grammars sequentially

Multiphase transducer

- The multiphase transducer lists the other grammars to be loaded: all you need to load is this file

- In ANNIE this is called main.jape - by default we usually label multiphase transducers with "main" in the filename

MultiPhase: TestTheGrammars ← name of the multiphase

Phases: ← list the phases in order of processing

first

name

date

final

Input Annotations

- The Input Annotations list contains a list of all the annotation types you want to use for matching on the LHS of rules in that grammar phase, e.g.
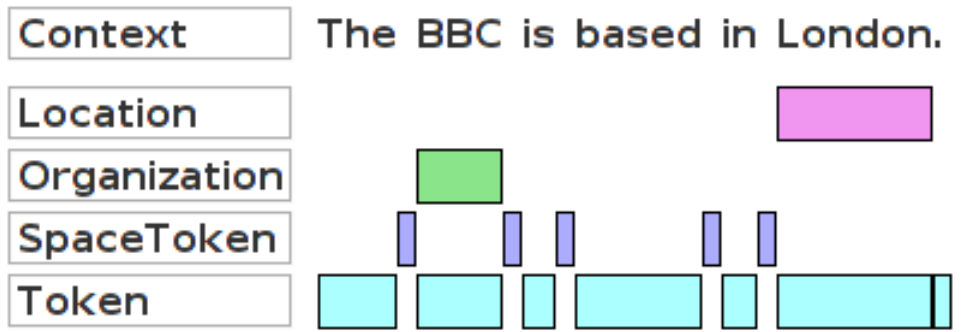
  `Input: Token Lookup`

- If an annotation type is used in a rule but not mentioned in the list, a warning will be generated when the grammar is compiled in GATE

- If an annotation is listed in Input but not used in the rules, it can block the matching (e.g *Split*)
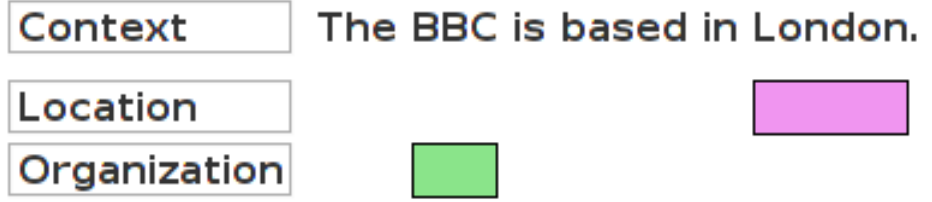
- If no input is included, then all annotations are used

Input Annotations

{Organization} {Location}

No Input

Input: Organization Location

Exercise: input annotations

- Try altering the Input annotations in *university1.jape*

- Remove the Lookup annotation from the list. What happens when you run the grammar?

- Why?

- Add "SpaceToken" to the list. What happens when you run the grammar?

- What happens if you then add SpaceToken annotations into the rule?

- Check the Messages tab each time to see if GATE generates any warnings.

Matching styles

Options: control = appelt

- "*Rules in the same phase compete for input*"

- What happens when 2 rules can match the same input?

- What happens when the same rule can match different lengths of input (e.g. +,* operators)?

- The matching style controls

  - Which rule gets applied

  - How much document content is '*consumed*'

 - Which location to attempt matching next

Matching styles

5 different control styles possible:

- **appelt** (longest match, plus explicit priorities)

- **first** (shortest match fires)

- **once** (shortest match fires, and all matching stops)

- **brill** (fire every match that applies) (this is the default)

- **all** (all possible matches, starting from each offset in turn)

Matching styles

Tim   Berners   Lee

{Name}+
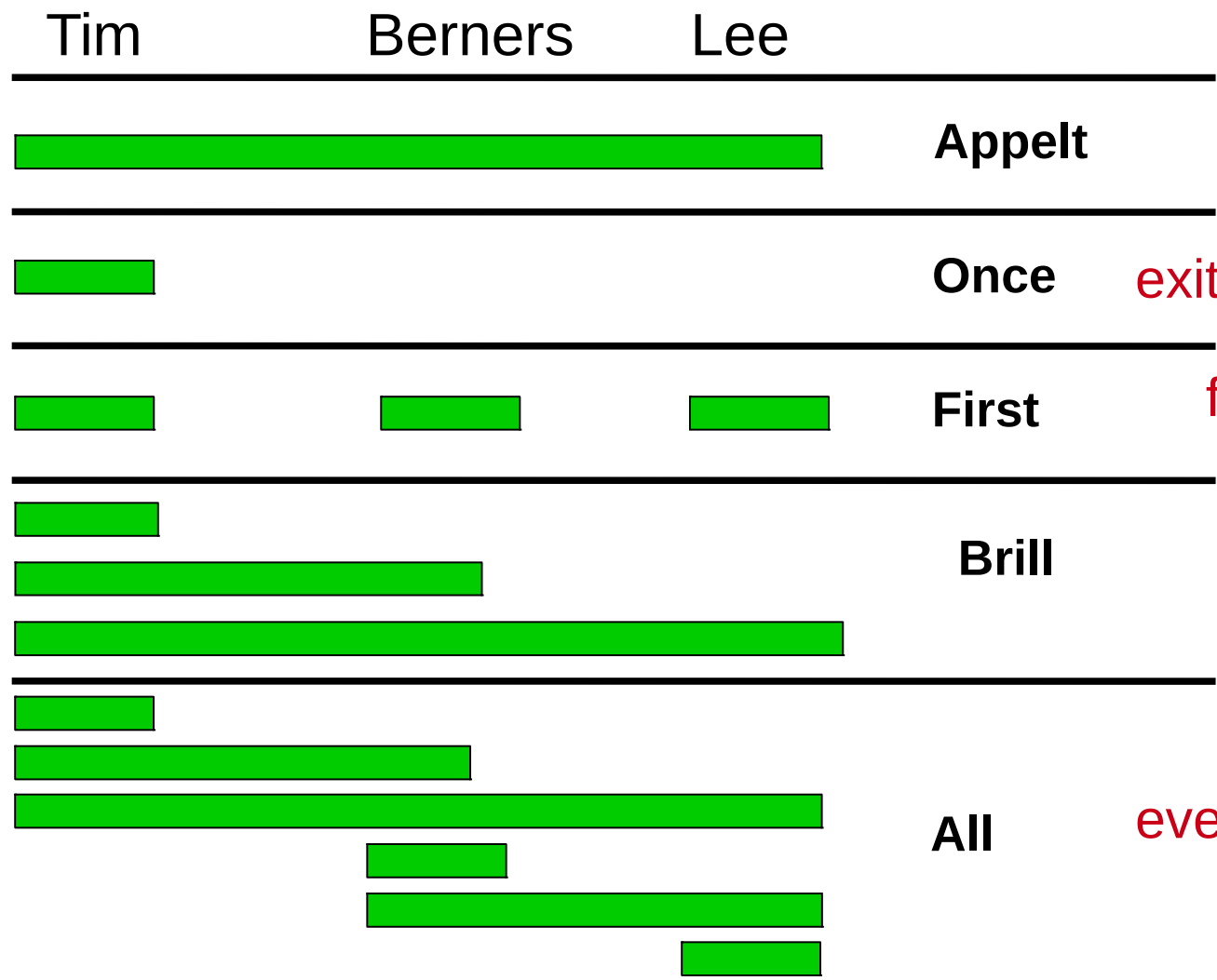
Appelt   longest match

Once   exit after first match

First   first match

Brill   every combination
from start of match

All   every combination

Appelt style

- In the appelt style, which rule to apply is selected in the following order:

  - longest match

  - explicit priority

  - rule defined first

- Each rule has an optional priority parameter, whose value is an integer

- Higher numbers have greater priority

- If no explicit priority parameter, default value is -1

- Once a match has fired, matching continues from the next offset following the end of the match

```
Rule:    Location1
Priority: 25
```

Difference between first and once

- With both styles, the first match is fired

- This means they're inappropriate for rules ending in the operators + ? or *

- The difference between the two styles is what happens after a match has been found

- With the once style, the whole grammar phase is exited and no more matches are attempted

- With the first style, matching continues from the offset following the end of the existing match

Difference between brill and all

- Both *Brill* and *all* match every possible combination from a given starting position

- When a match has been found, *brill* starts looking for the next match from the offset at the **end** of the longest match

- *All* starts looking for the next match by advancing one offset from the **beginning** of the previous match

LHS Macros

- Macros provide an easy way to reuse long or complex patterns

- The macro is specified once at the beginning of the grammar, and can then be reused by simply referring to its name, in all future rules

- Macros hold for ALL subsequent grammar files

- If a new macro is given later with the same name, it will override the previous one for that grammar

- Macro names are by convention written in capitals, and can only contain alphanumeric characters and underscores

- A macro looks like the LHS of a rule but without a label

Using a macro in a rule

```
Macro: NUMBER_FULL
({Token.kind == number}
 (({Token.string == ","}| {Token.string == "."})
  {Token.kind == number}
 )*
)
Rule: MoneyCurrencyUnit
  (
   (NUMBER_FULL)?
   ({Lookup.majorType == currency_unit})
  )
:number -->
  :number.Money = {kind = "number", rule =
  "MoneyCurrencyUnit"}
```

Multi-constraint statements

- You can have more than one constraint on a pattern

- Just separate the constraints with a comma

- Make sure that all constraints are enclosed within a single curly brace

```
{Lookup.majorType == loc_key,
 Lookup.minorType == post}
```

Is not the same as

```
{Lookup.majorType == loc_key}
{Lookup.minorType == post}
```

Negative constraints on annotations (!)

- You can use the ! operator to indicate negation

- Negative constraints are generally used in combination with positive ones to constrain the locations at which the positive constraint can match.

```
Rule: PossibleName
(
 {Token.orth == upperInitial, !Lookup}
):name
-->
 :name.PossibleName = {}
```

- Matches any uppercase-initial Token, where there is no Lookup annotation starting at the same location

Negative constraints on features (!=)

- The previous example showed a negative constraint on an annotation **{!Lookup}**

- You can also constrain the features of an annotation

- **{Lookup.majorType != stop}** would match any Lookup except those with majorType "stop" (stopwords)

- Be careful about the difference between this and

{!Lookup.majorType == stop}

- This matches ANY annotation except a Lookup whose majorType is "stop", rather than any Lookup where the majorType is not "stop"

Comparison operators

- So far, we have compared features with the equality operators == and !=

- We can also use the comparison operators >, >=. < and <=

- **`{Token.length > 3}`** matches a Token annotation whose length is an integer greater than 3

Kleene operator for ranges

- You can specify ranges when you don't know the exact number of occurrences of something

- ({Token})[2,5] will find between 2 and 5 consecutive Tokens

- In most cases you do NOT want to use unbounded Kleene operators (*, +) because they are not very efficient

Regular expression operators

- You can also use =~ and ==~ to match regular expressions

- **`{Token.string ==~ "[Dd]ogs"}`** matches a Token whose string feature value **is (exactly)** either "dogs" or "Dogs"

- **`{Token.string =~ "[Dd]ogs"}`** is the same but matches a Token whose string feature **contains** either "dogs" or "Dogs" within it

- Similarly, you can use !=~ and !~

- In the first example, it would match a Token whose string feature is NOT either "dogs" or "Dogs"

- In the second example, it would match a Token whose string feature does NOT contain either "dogs" or "Dogs" within it

Contextual operators

- The contextual operators "contains" and "within" match annotations within the context of other annotations

- {Organization contains Person} matches if an Organization annotation completely contains a Person annotation.

- {Person within Organization} matches if a Person annotation lies completely within an Organization annotation

- The difference between the two is that the first annotation specified is the one matched

- In the first example, Organization is matched

- In the second example, Person is matched

Combining operators

- You can combine operators of different types, e.g.

- **{Person within {Lookup.majorType == organization}}**

- **{!Person within {Lookup.majorType == organization}}**

- **{Person within {Lookup.majorType != organization}}**

- **{Person contains {!Lookup}, Person within {Organization}}**

- But be sure you know what you're doing, as it can get quite complicated!

- Note that {Person contains Person} might give some unexpected results!

45

Copying Feature Values to the RHS

- JAPE provides simple support for copying feature values from the LHS to the RHS

```
(
{Lookup.majorType == location}
):loc
-->
:loc.Location = { type = :loc.Lookup.minorType}
```

- This copies the value of the Lookup minorType feature from the LHS to the new Location annotation

- Note that if more than one Lookup annotation is covered by the label, then one of them is chosen at random to copy the feature value from

- It's best not to use this facility unless you know there is only one matching annotation

46

Exercise: copying Lookup features

- Open university1.jape in your text editor and create a new annotation called "UniversityTown" that matches just the city name.

- Also copy the value of the majorType of the city to a new feature of this annotation, called "kind"

Hint: the RHS for the previous example looked like this:

```
:loc.Location = { type = :loc.Lookup.minorType}
```

- Run this on the university1.txt document and check the results

- You should see "Sheffield" annotated like this:

Type	Set	Start	End	Id	Features
UniversityTown		36	45	49	{kind=location}

Summary

- This module has looked at some basic operations within JAPE.

- The best way to learn is to keep practising. Try things out and see what happens.

- It's usually best to build up JAPE rules in simple steps.

- Trying to do too much in a single rule will get you confused.

- Pay close attention to syntax and to things like making sure case is respected and that you have no typos in your rules.

- Remember you can use in your JAPE rules any annotations that you have previously used in your pipeline.

- You can also use any Java you want in your rules.