RegularExpressions

- Java supports a Perl-like regular expression syntax
  - Regular expressions are compiled expressions (patterns) that allow to express syntax rules
  - They enable you to find and replace text an based on an exact syntax (language) (e.g, an email address, creditcard number, zip code etc)

Expression	example matches
`mon|wed|fri`	**mon** tue **wed** thu **fri** sat sun abc 123.2
`colo.?r`	**colour** **color**
`[0-9]*Kg`	100gr **200Kg** 300 100MB 12KM **150Kg**
`[0-9]{2}-[0-9]{3}`	**11-891** 111-90 992-172 a1-222

- Main classes
  - Main class is `Pattern` - compiled regular expression

Method		Description
Factory	**`compile (expr)`**	Creates the `Pattern` for the specific expressions
	`compile ( expr, flags)`	Creates the `Pattern` using different flags
use	**`split ( CharSequence )`**	Return an array of string particles, split on matches
	`matcher( CharSequence )`	Returns a `Matches` (used to find or replace text)

- The `Pattern` is instantiated passing the expression in one of the `compile` methods

```
Pattern p = Pattern.compile("3[47][0-9]{13}");
... = p.matcher(paymentStr);
```

- An unchecked `PatternSyntaxException` is thrown when pattern is invalid.

```
try{
  // Invalid expression
  Pattern pattern =Pattern.compile("10{x}");
}catch(PatternSyntaxException pse){
  System.out.println("Invalid regular expression");
}
```

- `Matcher` class used for matching (find based on different scenarios) and replacing text

- Matching

Method		Description
Matching	`matches`	Attempts to match the entire string
	`lookingAt`	Attempts to match the start of the string
	`find`	Find the next subsequence that matches the pattern
	`find(int startAt )`	Find starting at a specified position
match info	`start`	The start index of the previous match
	`end`	The end index of the previous match
	`group`	Returns the subsequence of previous match

- Replacing

Method	Description
`replaceAll(String)`	Replaces every match with the supplied string
`replaceFirst (String)`	Replace the first match with the supplied string
`appendReplacement`	Append/replace the previous match (append position)
`appendTail`	Append the last part of the input (from append position)

- Pattern flags

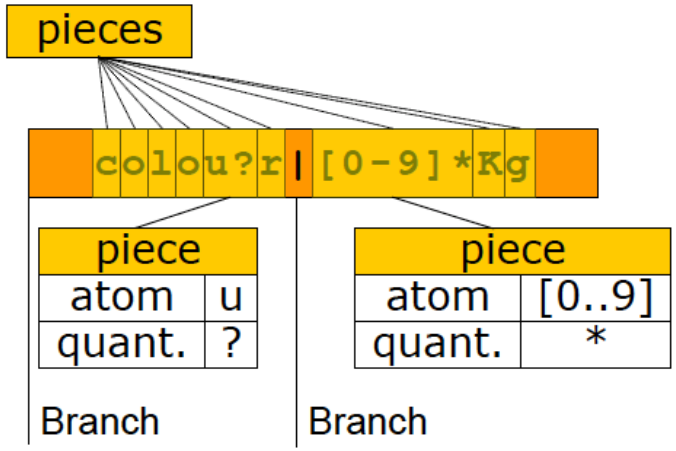
Flag	Description
CASE_INSENSITIVE	For case-insensitive matching
MULTILINE	Position markers are aware of the newline character
DOTALL	The "any character" includes the line terminator
UNICODE_CASE	Uses unicode instead of ASCII
CANON_EQ	Enables canonical equivalence
UNIX_LINES	Enable only Unix lines mode
LITERAL	The pattern is literal (special characters are literal)
COMMENTS	Permits whitespace and comments (are both ignored)

- The `String` class contains methods which take regular expressions
  - Without the direct use of `Pattern` and `Matcher` object

```
public boolean matches(String regex)
public String replaceFirst(String regex,String replacement)
public String replaceAll(String regex,String replacement)
public String[] split(String regex,int limit)
public String[] split(String regex)
```

  - These methods might throw a `PatternSyntaxException`

- each branch is made up of multiple *pieces* which is an *atom* and an optional *quantifier*



- An atom is a regular expression particle (which has an occurrence)

Atom type			Examples			
character		any character except metacharacters	all except: `.`, `\`, `?`, `*`, `+`, `{`, `}` `(`, `)`, `[` and `]`			
character class	escape class	Predefined characters sequences	`\n`	newline	`.`	wildcard
			`\d`	digits	`\p{M}`	marks
	expression class	Character groups (negative, positive or intersection)	`[abc]`	a,b or c	`[^ab]`	no a or b
			`[a-z]`	a to z	`\p{Sc}`	currency
(	*regex*)	A group. Contains another regular expression	`([0..9]| na)*`		sequence with digits and 'na' strings	

- The *metacharacters* are reserved characters that need to be escaped if used literally.

Metacharacter	
`.`	wild card escape character
`\`	escape character
`[]`	character class expression
`()`	grouping character
`? * + {}`	quantifier characters

Escaping metacharacter examples			
`1\.5`	1.5	`1\+0\.5`	1+0.5
`3\\4`	3\4	`3\\4`	3\4

- A (greedy) quantifier specifies the occurrence of the atom (quantity is `1` when omitted)

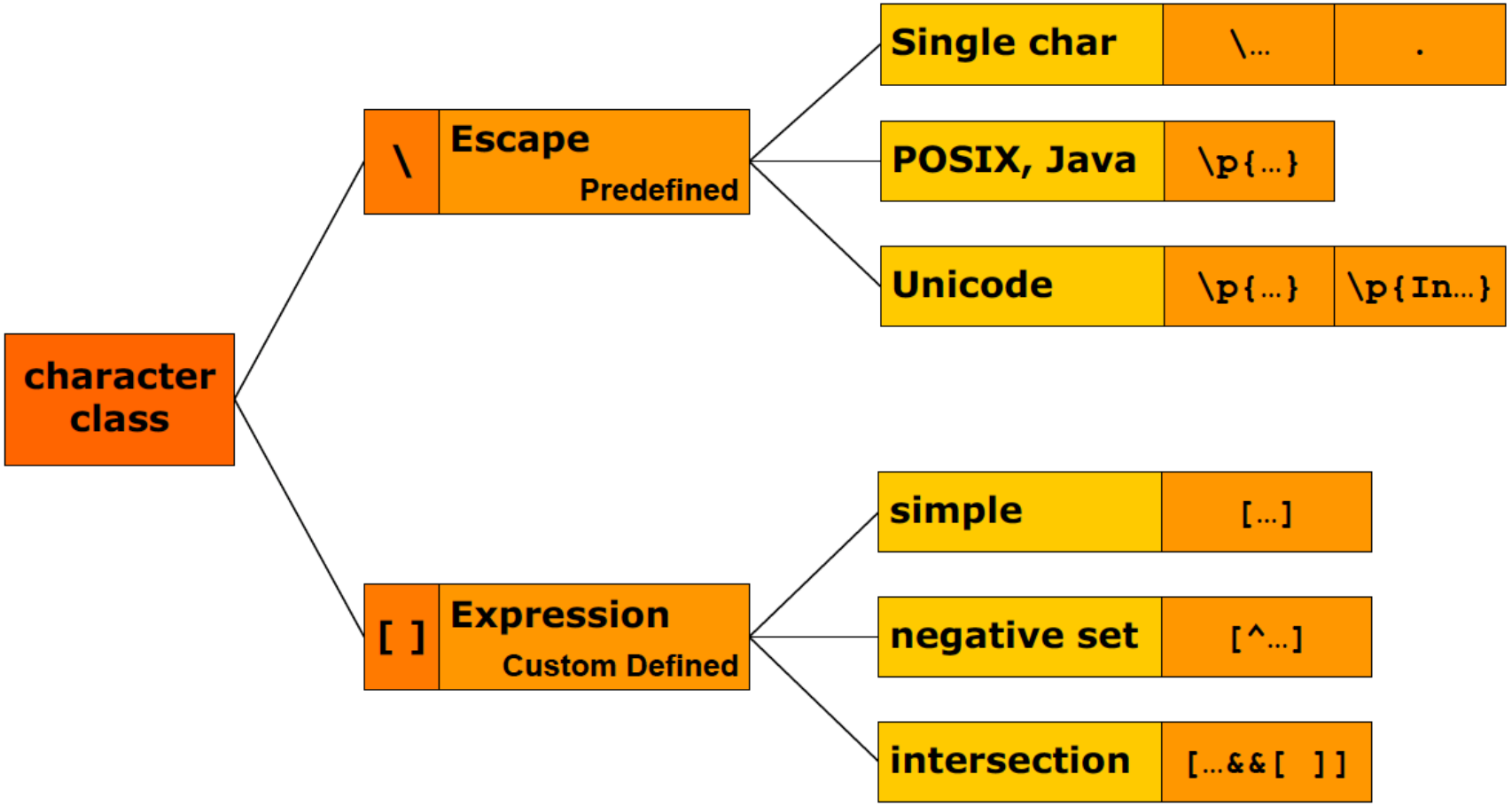
Quantifier		#	Example	matches
multiplicity	*	0..n	`How.*\?`	"How?", "How are you?", ~~"How much?!"~~
	+	1..n	`http://.+/`	"http://www.w3c.org/", ~~"http://abc"~~
	?	0 or 1	`[+-]?[1-9]`	"+1", "-9", "4", "-4", "+4"
range	{x,y}	range	`[0-9]{1,3}`	"1", "28", "133", ~~"9810"~~, ~~" 1",~~
	{x,}	min	`[0-9]{2,}`	"10", "299", "392039029", "~~8~~"
	{x}	exact	`[0-9]{2}-[0-9]{2}`	"12-32", "00-01", "~~1 12~~", "~~100 10~~"

- Boundary

anchor		example matches
`\bex`	word	The **ex**travagant boy ordered T**ex**-M**ex**
`ing\z`	end	**ing**rid, who was listen**ing**, ordered another thi**ng**
`\Aing`	start	**ing**rid, who was listen**ing**, ordered another th**ing**
`^ing`	linestart*	**ing**rid, who was listen- **ing**, ordered another th**ing**
`ing&`	lineend*	**ing**rid, who was listen**ing**, ordered another th**ing**

*Work only in MULTILINE mode (otherwise same as \z and \A)

- *Character classes* define a class of characters either predefined (through escaping) or custom defined by an expression



- Escape characters classes reference predefined character classes

- *Single-character escape* used for character classes with a single character (Metacharacters and other additional characters)

\n	Linefeed (\u0A)	**\f**	form feed (\u0C)
\r	carriage return (\u0D)	**\a**	alert / bell (\u09)
\t	tab (\u09)	**\e**	Escape (\u1B)
\-	hyphen	**\cX**	Control char X
\^	circumflex	**|**	pipe

- *Multi-character escape* used for character classes with multiple characters based on Perl groups, which identify commonly used sets of characters

Escape sequence		Explanation
\s	spaces	All Whitespace characters [#x20\t\n\r]
\d	decimal digits	9, 1, ٢ ,٩ (all digits in Unicode)
\w	word characters	All except Unicode punctuation, separator and 'other' characters
\S \D \W	Complement	Negative set (e.g., \S is all but spaces)

```
\d*Kg (?#Digit followed by Kg )
\d*\sKg (?#Digit followed a space, followed by Kg )
```

- The *wildcard* is a special Multi-character escape matching any character (except `\n` and `\r`)

```
(?#Sentence ending with a question mark )
.*\?
How.+\?
```

- POSIX character classes (US-ASCII)

POSIX (\p{…})			
Lower	Lower case	**Punct**	Punctuation
Upper	Upper case	**Graph**	Visible character
ASCII	All ASCII	**Print**	Printable
Alpha	Alphabetic	**Blank**	space or tab
Digit	Decimal	**Cntrl**	Control
Alnum	Alphanumeric	**XDigit**	Hexidecimal
		Space	whitespace

- *Category escape* used for character classes with multiple characters based on Unicode *general categories* and *blocks*

Character class	Syntax	Complement	example	
Unicode Category	**\p{…}**	**\P{…}**	\p{Lu}	Uppercase letters
Unicode Block	**\p{In…}**	**\P{In…}**	\p{InGreek}	Greek characters

- Unicode *categories* and *blocks*
  - Unicode categories

Letters (L)		
Lu	Uppercase	Δ
Ll	Lowercase	a
Lt	Title case	Lj
Lm	Modifiers	ˆ
Lo	Other	ƻ

Numbers (N)		
Nd	Decimal	1
Nl	Number	xi
No	Other	²

Separators (Z)		
Zs	Space	x20
Zl	Line	x2028
Zp	Pargrph.	x2029

Symbols (S)		
Sm	Math	+
Sc	Currency	€
Sk	Modifier	¨
So	Other	©

Punctuation (P)		
Pc	Connector	‿
Pd	Dash	-
Ps	Open	(
Pe	Close	)
Pi	Initial quote	‘
Pf	Final quote	”
Po	Other	!

Not shown

Marks (M): Mn (nonspacing), Mc (spacing combining), Me (enclosing)
Other (C) :Cc (control), Cf (format), Co (private), Cn (not assigned)

```
(?# Question starting with a uppercase letter)
\p{Lu}.*\?
```

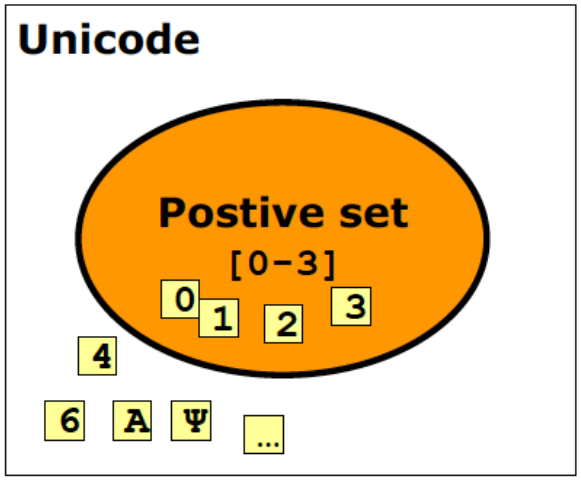
- Unicode blocks

Example Unicode Blocks			
BasicLatin	Syriac	Tibetan	Dingbats
Latin-1Supplement	Thaana	Runic	BraillePatterns
LatinExtended-A	Devanagari	Khmer	YiSyllables
LatinExtended-B	Bengali	Mongolian	YiRadicals
Greek	Gurmukhi	LatinExtendedAd…	HangulSyllables
Cyrillic	Gujarati	GreekExtended	MathematicalOp…
Armenian	Tamil	GeneralPunctuation	Arrows
Hebrew	Telugu	CurrencySymbols	NumberForms
Arabic	Thai	GeometricShapes	…

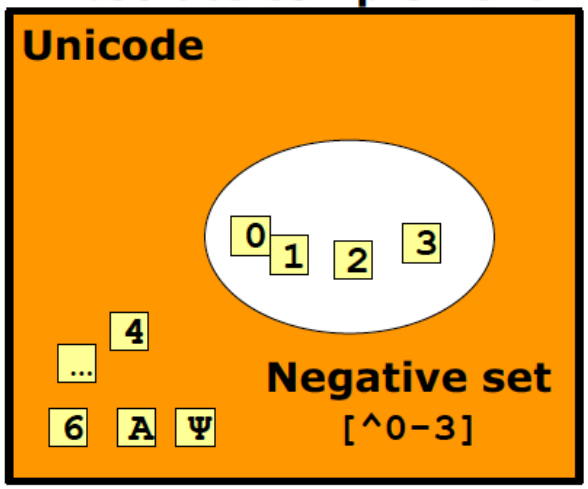
```
(?# Sequence of basic latin characters)
\p{inBasicLatin}*
```

- *Character class expressions* define custom character classes

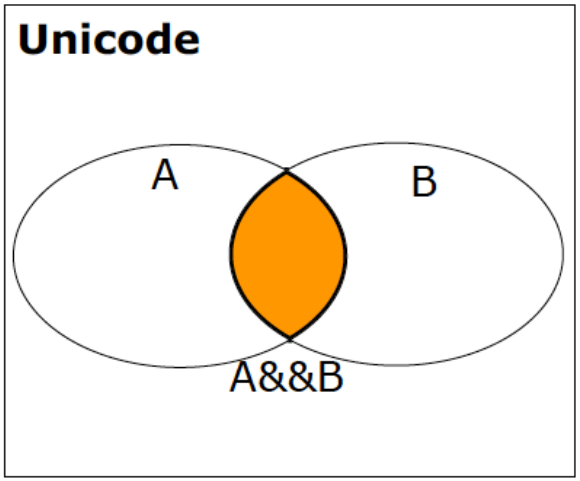
positive character groups
Union of atoms



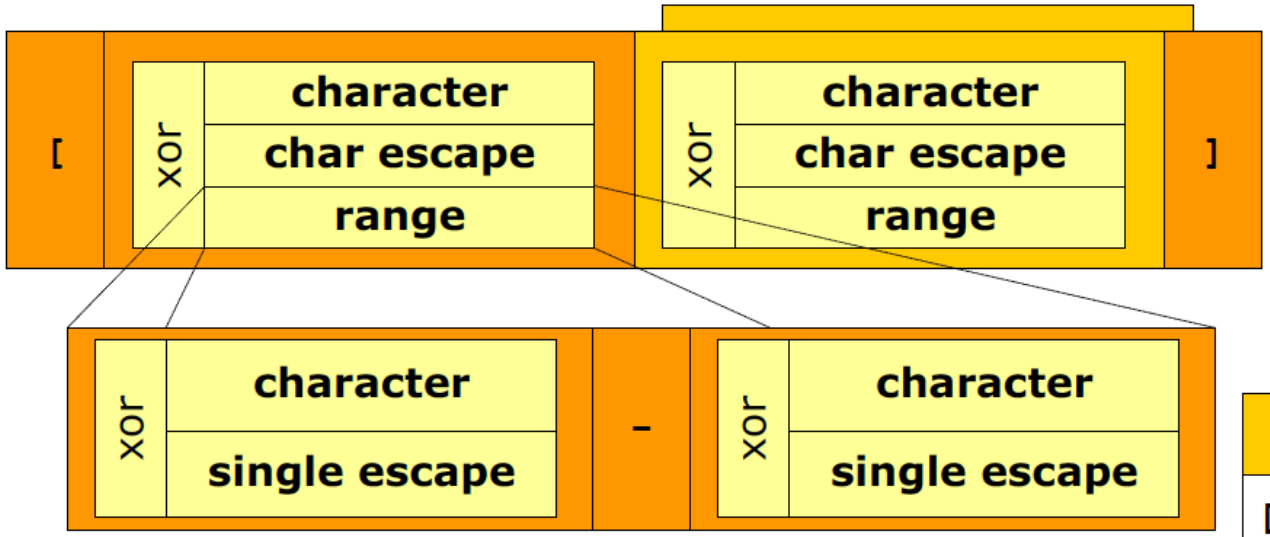
negative character group
Absolute complement



character class intersection
Intersection



- The *simple character group* is the basic expression that defines the set of characters as a union of atoms.



is a range of Unicode code points

Examples	
`[a]`	`[0-46-9]`
`[!?]`	`[0-4\p{Ll}]`
`[a\p{N}\s]`	`[!-\.]`
`[0-9]`	`[a-z]`

- The *negative character group* defines an **absolute complement**

Examples
`[^a]`
`[^!?]`
`[^a\p{N}\s]`
`[^0-9]`

- The *intersection character group* defines an **intersection**

Examples	Results in Set
`[\p{inBasicLatin}&&[\p{Lu}]]`	All Latin uppercase letters
`[\p{InArabic}&&[\p{N}]]`	All Arabic numbers

- Last two can be used to make a subtraction of two sets (**relative complement**)

[	positive/nagative character group	&&	[	^	character group	]	]

Examples
`[0-9&&[^3]]`
`[\p{Lu}&&[^M-Z]]`

Questions