

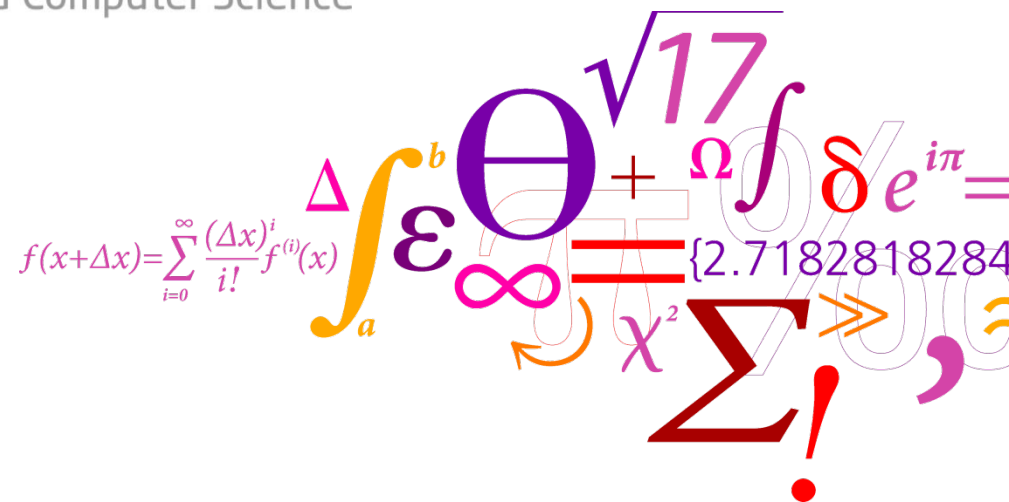
Projekt i software-udvikling (02362)

Forår 2022

Ekkart Kindler

DTU Compute

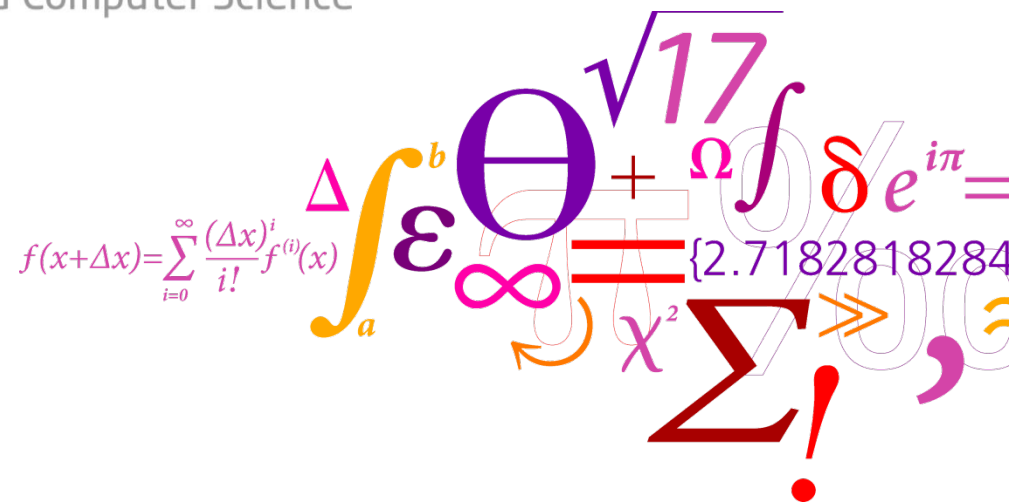
Department of Applied Mathematics and Computer Science



VIII. Inputvalidering regulære udtryk

DTU Compute

Department of Applied Mathematics and Computer Science



A collage of colorful mathematical symbols and expressions. It includes a Taylor series expansion $f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$, an integral $\int_a^b \epsilon \Theta$, a square root $\sqrt{17}$, a plus sign $+$, a Greek letter Ω , a delta function δ , an exponential function $e^{i\pi}$, an equals sign $=$, a set of numbers $\{2.7182818284\}$, a chi-squared symbol χ^2 , a summation symbol Σ , a greater-than symbol $>$, and an exclamation mark $!$.

I nogle situationer vil man sikre sig, at brugeren indtaster informationer, som overholder en hvis struktur:

- Adresser
- email-adresser
- Telefonnumre
- Datoer
- Tal og variabelnavn i et programmer
- ...

Eksempler:

- ekki@dtu.dk
- anton.bach@gmail.com
- anton24.bach.copenhagen@gmail.com
- xyz.234.bla@blubs.domain.etc.xy

Modeksempler:

- ekki@dtu.dk@dtu.dk
- anton bach@domain.bla
- @dtu.dk

- Java Identifier:
Består af mindst en bogstav og efterfølgende lige så mange tal eller bogstaver som man vil:
eksempler: i, test1, Test4711, TEST
modeksempel: i k, 4711Test
- En Java heltal:
eksempler: -5, 000, -0764, +05, 5, 4711
modeksempel: 3-5, 7.89, 10ab, +, -

Mange af sådanne strukturer kan man defineres med såkaldte **regulære udtryk**

Regulære udtryk er et meget gammelt koncept i datalogi – meget ældre end Java. Men her bruger vi især notationen fra Java (*, +, ?, | er i generel brug også i andre kontekster)

→ Kompilerkonstruktion (→ datalogiske modeller)

Regulære udtryk består af nogle begreber for at beskrive hvilke tekster er legal input til noget.

Begreberne er næsten altid de samme, men der er mange forskellige notationer (som ligner hinanden) for at formulere dem. Her bruger vi en notation, som man kan bruge med nogle "matchere" i Java.

- **[a-z]**
matcher en string bestående af helt præcist et lille bogstav mellem **a** og **z** (og ikke noget andet)
- **[A-Z]**
matcher en string bestående af helt præcist et stor bogstav mellem **A** og **Z** (og ikke noget andet)
- **[1-6]**
matcher en string bestående af helt præcist et tal mellem **1** og **6** (og ikke noget andet)
- **[ade]**
matcher de enkelte bogstaver "**a**", "**d**" eller "**e**" (og ikke noget andet) (og ikke noget andet)

Det siger jeg ikke igen på de næste slides!

- `\w` alle bogstaver (store og små) mellem a-z og enkelte tal (engelsk "word") og "_" (underscore)
- `\d`
svarer til [0-9] dvs. en enkelte tal
- `[^A-Z]`
matcher alle enkelte tegn som ikke er en stor bogstav mellem A og Z
- `.`
matcher hvert enkelte tegn
- `\s`
matcher alle enkelte tomrums-tegn ("white **s**pace"):
space, tab, line feed, ...

Symbolet "^" hedder "caret" og efter "[" betyder det en negation.

- `\"`
matcher det enkelte tegn `\"`
- `\.`
matcher det enkelte tegn `.`

Det er nødvendigt, da `\"` og `\.` bruges med en særlig betydning indenfor regulære udtryk (se slide 9). `\"` er "escape"-tegnet indenfor regulære udtryk (i den notation vi bruger).

Vi antager at $R1$ og $R2$ er regulære udtryk

- $R1 \mid R2$
matcher en string som matcher $R1$ eller $R2$ (englsk. "or")
- $R1^*$
matcher en string som kan deles op i lige så mange dele som man vil (inkl. 0) som alle matcher $R1$ (engl. iteration)
- $R1^+$
matcher en string som kan deles op i lige så mange dele som man vil (dog mindst en) som alle matcher $R1$

Vi antager at $R1$ er regulært udtryk

- $R1?$

matcher en String som enten er tom eller som
matcher $R1$ (engl. optional)

- For at gruppere udtryk kan man bruge parenteser

$([A-Z] [a-z]^*) (\backslash s (([A-Z] | [a-z]) [a-z]^*))^* \backslash .$

- der er nogle flere
- fx. for at sige at teksten, som matcher er i starten af den string man matcher eller at i slutningen af den string: $\wedge$ (dog ikke efter $[]$) og $\$$

Se reference I får på det sidste slide

- Java Identifier:

Består af mindst en bogstav og efterfølgende lige så mange tal eller bogstaver som man vil:

eksempler: i, test1, Test4711, TEST

modeksempel: i k, 4711Test

Kort diskussion (i grupper)

- En Java heltal:

Eksempler: -5, 000, -0764, +05, 5, 4711

Modeksempler: 3-5, 7.89, 10ab, +, -

- Email adresse uden særlige tegn:
Eksempler: ekki@dtu.dk
Modeksempler: ekki, ekki@dtu@dk, ekki.dtu.dk
→ Diskussion på tavlen

Vi tager bare den simple version. En regexp som fanger alle detaljer af en korrekt email-adresse er forbavsende kompleks!

Prøv selv: Vi diskuterer det under live-delen af dagens undervisning. Men kun en enkel version.

Javas **String**-klasse har en række indbyggede operationer, for at tjekke om en **String** matcher en regulær udtryk:

```
String SIMPLE_DOMAIN_NAME_PATTERN =  
    "\\w+(\\.\\w+)*";  
String example = "compute.dtu.dk";
```

Da "\" er et "escape"-tegn i Javas String-konstanter, er vi nødt til at have to "\\" her "\\" → "\". Da "\" også er et "escape"-tegn i regexp-sproget og "\" = "\", er der faktisk fire "\\\\" = "\"", når vi vil referere til bare "\".

```
if (example.matches(SIMPLE_DOMAIN_NAME_PATTERN) {  
    // do something  
}
```

Tjekker om **example** matcher **SIMPLE_DOMAIN_NAME_PATTERN** mønstre

Java har også metoder som erstatter den første delstring af en **String** som matcher en regexp med en anden **String**:

- `String s2 =
 s.replaceFirst(PATTERN, "replacement");`

eller en metode som erstatter alle delstring af en **String** som matcher en regexp med en anden **String**:

- `String s2 =
 s.replaceAll(PATTERN, "replacement");`

- Match og replace metoderne direkte på **Strings** er ikke særlige effektive
- Derfor eksisterer der i Java to klasser **`java.util.regex.Matcher`** og **`java.util.regex.Pattern`** hvor regulære udtryk kan kompileres (oversættes til noget som kan eksekveres effektivt) før de bliver anvendt
- Mere information om det findes på:
<http://www.vogella.com/tutorials/JavaRegularExpressions/article.html>
<https://docs.oracle.com/javase/tutorial/essential/regex/intro.html>

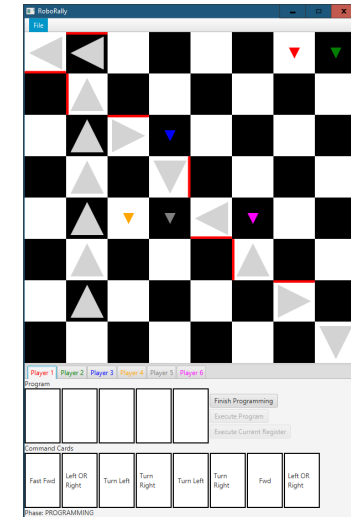
Jeres software skal kunne **gemme og genlade spil i/fra en database** (visnings opgave):

- spillets tilstand,
- spillerens/robottens tilstand inkl.
- spillerens kort (på hånden) og især programmet

Når spillet genlades, skal også GULen vise den aktuelle situation rigtig (især når den var i interaktiv modus). Men hvis MVC og ApplicationController er sat op hensigtsmæssigt, burde det næsten være "gratis".

Med denne opgave skal I vise (V-opgave), at jeres software kan lade en spilleplade fra en (JSON)-fil:

- Størrelsen af pladen,
- væg på felterne,
- aktion(er) på felterne



Og sørg for at felternes aktioner bliver udført, når alle robotter har udført kommandoet på det aktuelle register.

Koden til databasen, og Gson, fik I udleveret i uge 6: roborally-1.4.1.