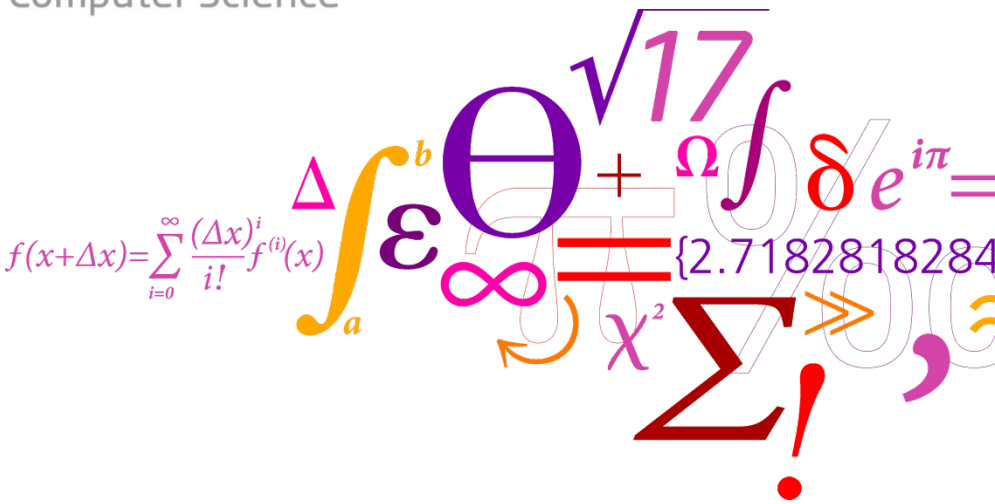
Slides PiSU-L07

Videregående programmering (02324)
Projekt i software-udvikling (02362)
Forår 2021

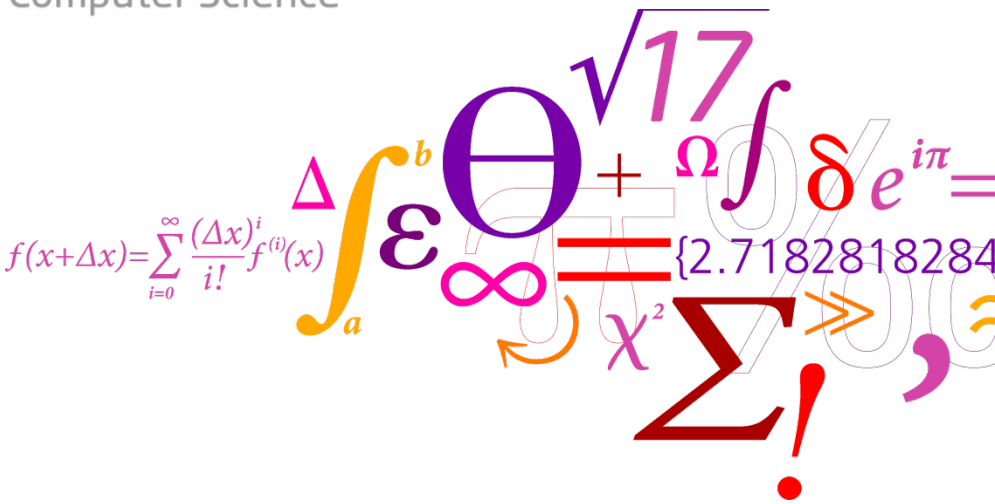
Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science



VII. Læse og skrive filer, **Gson**

DTU Compute
Department of Applied Mathematics and Computer Science



1. Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Til at gemme tilstanden af et spil, kan vi bruge databasen (via DAL, DAO → uge 6)
- Men der er nogle aspekter af RoboRally-spillet, som er statiske (dvs. som ikke ændrer sig under selve spillet), men som alligevel skal/kunne gemmes et sted: fx
  - Hvilke felter (spaces) samt alle deres væge og aktioner er der på pladen?
  - …
- Det giver (måske) ikke så meget mening at gemme dem i databasen

Motivation

- Disse informationer kan gemmes i filer, som bliver udleveret med selve software (fx. som ressource-filer → se senere)

- Java har en masse klasser med metoder, til at tilgå, og finde filer og foldere og til at kreere, læse, skrive og slette dem (se → slide 6ff)

- Til at gemme komplekse informationer **var** man ofte nødt til at skrive meget kode til at **serialisere** og **deserialisere** informationerne!

Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- I dag eksisterer der JSON (JavaScript Object Notation) og mange biblioteker (APIer), til at læse og skrive JSON filer (fx Google's **Gson**)
- Når man bruger dem,
  - er man **ikke** nødt til at programmere så meget til at serialisere og deserialisere informationer til og fra filer
  - er man **ikke** nødt til at behandle filer på det lave tekniske niveau
  - kan man indlæse informationer fra filer direkte som **objekter/modeller**

2. API: Filer og Folder

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Nogle vigtige klasser til at håndtere filer i Java er

- **`java.io.File`**
- **`java.io.FileReader`**
- **`java.io.FileWriter`**

Se https://docs.oracle.com/javase/8/docs/api/index.html?java/io/File.html

java.io.File

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Konstruktorer:

- `public File(String pathname)`
- `public File(File parent, String child)`
- …

Metoder:

- `public boolean exists()`
- `public boolean isFile()`
- `public boolean isDirectory()`
- `public boolean createNewFile()`
- `public File[] listFiles()`
- …

En fil kan ogs
være en folde

API: Filer og Folder

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Bemærk at **`java.io.File`** kan ikke umiddelbar bruges til at læse eller skrive filer!
Det kan man gøre med

- **`java.io.FileReader`**
- **`java.io.FileWriter`**
- ...

og deres metoder

Se https://docs.oracle.com/javase/8/docs/api/index.html?java/io/FileReader.html og https://docs.oracle.com/javase/8/docs/api/index.html?java/io/FileWriter.html men det taler vi ikke nærmere om, da vi bruger Gson, på lidt højere niveau

For at lade brugeren vælge en eller flere filer, eksisterer der også fil-dialoger. Derfor er man heller ikke nødt til at navigere rund i fil-systemet selv.

3. JSON og **Gson**

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Vi starter med at præsentere JSON med hjælp af et eksempel (spillepladen til RoboRally eller ”templates” dertil)
- Bagefter diskuterer vi, hvordan man kan læse og skrive sådanne filer med Gson

JSON: Eksempelmodel

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public class BoardTemplate {

    public int width;
    public int height;

    public List<SpaceTemplate> spaces =
        new ArrayList<>();

}
```

JSON: Eksempelmodel

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public class SpaceTemplate {

    public int x;
    public int y;

    public List<Heading> walls = new ArrayList<>();
    public List<FieldAction> actions = new ArrayList<>();

}
```

JSON: Eksempelmodel

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public abstract class FieldAction {

    public abstract boolean doAction(
        GameController gameController, Space space);

}
```

Måske bedre bare at bruge et interface til GameController

`FieldAction` beskriver aktioner på pladen, men bliver samtidigt brugt som kommandoer (→ design pattern) som kan eksekvere aktionen (**`doAction`**).

```
public class ConveyorBelt extends FieldAction {

    private Heading heading;

    ...

    @Override
    public boolean doAction(
        GameController gameController,
        Space space) {

        // TODO

    }

}
```

Da GameController kommer med som parameter, kan man bruge metoder fra GameControlleren her (så at kode ikke er dobbelt).

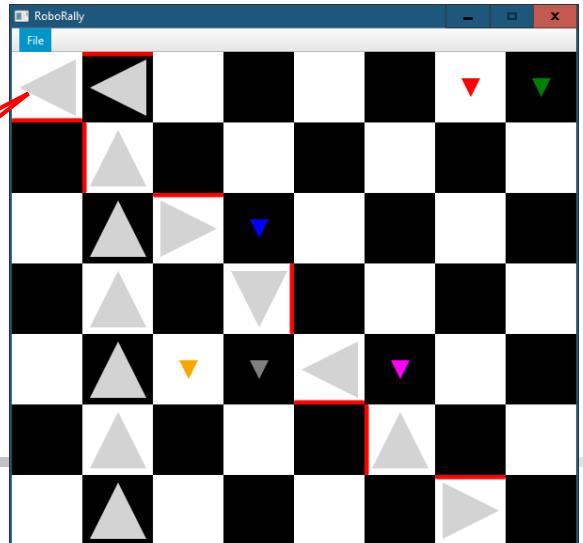
JSON: Eksempel fil

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
{ "width": 8,
  "height": 8,

  "spaces": [
    { "walls": [
        "SOUTH" ],
      "actions": [ {
          "CLASSNAME": "dk.dtu. … .ConveyorBelt",
          "INSTANCE": { "heading": "WEST" }
        } ],
      "x": 0,
      "y": 0
    }, ...
```

Det er det her felt



Gson: Læse JSON

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Vi antager at filen "**`boards/defaultboard.json`**" ligger i projektets resource-folder (en folder "**`resources`**" lige under "**`main`**" folderen i IntelliJ → se demo)
- Vi læser filen, så at dens indhold til sidst ligger i et objekt af type **`BoardTemplate`**

(Gson): kode

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
GsonBuilder simpleBuilder = new GsonBuilder().
  registerTypeAdapter(FieldAction.class,
     new Adapter<FieldAction>());


Gson gson = simpleBuilder.create();


ClassLoader classLoader =
  LoadBoard.class.getClassLoader();
InputStream inputStream =
  classLoader.getResourceAsStream(
    "boards/defaultboard.json");
```

Denne adapter få I udleveret sammen med noget kode til Gson (se slides 22/23). Den gøre det muligt at læse lister med flere typer aktioner.

Dette objekt kan so læse inputtet og give os et objekt som er repræsenteret i filen.

Det giver en input-stream til denne res urce.

Det har ikke noget med Java Streams fra Java 8 at gøre!

Gson: kode

Bemærk at vi ikke fanger exceptions her; og koden er ikke meget "defensiv" (koden I får udleveret gør det lidt bedre)

```
...
InputStreamReader streamReader =
    new InputStreamReader(inputStream);
JsonReader reader = gson.newJsonReader(streamReader);


BoardTemplate boardTemplate =
  gson.fromJson(reader, BoardTemplate.class);


reader.close();
```

Giver en Gson/JSON-reader med Gsons konfiguration ovenpå vores inputStream

Indeholder så **BoardTemplate**-objektet med alle kort-objekter fra resourcen

Læser inputtet og returnere et objekt af type **BoardTemplate**, hvis input ellers er korrekt.

Lukker readeren ordentligt

Gson: Skrive JSON

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Vi antager at vi har et **`BoardTemplate`**-objekt **`board`** som indeholder vores board-template, og at vi har en resource-folder i vores projekt
- Vi skriver en fil ”**`board.json`**”, som repræsenterer vores **`BoardTemplate`**-objekt samt dettes indhold
- **Bemærk**: For at gøre det enkelt, skriver vi filen også i vores "boards" folder i resourcen. Men det ville man normalt IKKE gøre! Man kunne bruge fil-dialoger for at spørge brugeren hvor en fil skal gemmes.

Gson: kode

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
ClassLoader classLoader =
  TestGson.class.getClassLoader();
String filename = classLoader.
  getResource("boards").getPath() +
  "/board.json");

FileWriter fileWriter = new FileWriter(filename);

GsonBuilder builder = new GsonBuilder().
    registerTypeAdapter(FieldAction.class,
        new Adapter<FieldAction>())
    setPrettyPrinting();

Gson gson = builder.create();
```

Det giver stien til filen (filen er ikke nødt til at eksistere endnu).

Det giver en ”writer” til selve filen.

Adapteren gøre det muligt at skrive lister med flere typer aktioner.

Det giver en builder-konfiguration (med ”pretty printing”)

Gson: kode


```
...

JsonWriter writer =
   gson.newJsonWriter(fileWriter);

gson.toJson(board, board.getClass(), writer);

writer.close();
```

Bemærk at vi ikke fanger exceptions her; og koden er ikke meget "defensiv".

Giver en Gson/JSON-writer med Gsons konfiguration ovenpå vores fileWriter

Writeren som objektet skal serialiseres til

Objektet som skal skrives

Klassen af objektet som skal skrives

Lukker writeren ordentligt

Gson integration

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Når man har et maven-projekt i forvejen, er det nemmest at integrere gson til projektet med en Gson-dependency i selve **`pom.xml`**-filen:

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.8.2</version>
  <scope>compile</scope>
</dependency>
```

Men, da jeg vidste det I forvejen, var det med I pom-filen I projektet fra starten.

JSON og Gson links

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- JSON: https://www.json.org/
- Gson: https://github.com/google/gson/blob/master/UserGuide.md

- Gson example adapter for a class with subtypes (will be discussed next time): https://github.com/mperdikeas/json-polymorphism/blob/master/src/IAnimalAdapter.java

Adapter: kode

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public class Adapter<E> implements JsonSerializer<E>,
JsonDeserializer<E>{

    private static final String CLASSNAME = "CLASSNAME";
    private static final String INSTANCE  = "INSTANCE";

    public JsonElement serialize(E src, Type typeOfSrc,
            JsonSerializationContext context) {

        JsonObject retValue = new JsonObject();
        String className = src.getClass().getName();
        retValue.addProperty(CLASSNAME, className);

        JsonElement elem = context.serialize(src);
        retValue.add(INSTANCE, elem);
        return retValue;
    }
```

For en god ordens skyld!
Men det bliver ikke diskuteret her!

Adapter: kode

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public E deserialize(JsonElement json, Type typeOfT,
        JsonDeserializationContext context)
           throws JsonParseException  {
    JsonObject jsonObject = json.getAsJsonObject();
    JsonPrimitive prim = (JsonPrimitive) jsonObject.
        get(CLASSNAME);
    String className = prim.getAsString();
    Class<?> klass = null;
    try {
        klass = Class.forName(className);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        throw new JsonParseException(e.getMessage());
    }
    return context.deserialize(
        jsonObject.get(INSTANCE), klass);
} }
```

Opgave V4a (til 12./6. 4.)

Bare som huskeseddel
(det er den fra sidste uge)

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 GUIen vise den aktuelle situation rigtig (især når den var interaktiv modus). Men hvis MVC og AppControler er sat op hensigtsmæssigt, burde det næsten være ”gratis”

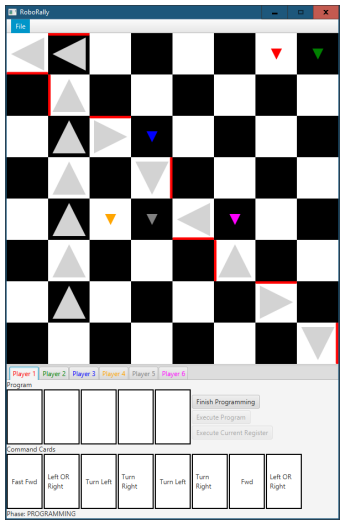
Vi diskuterer mere næste uge (og der udvides også opgaven så at en spilleplade kan lades V4b)

Opgave V4b (til 12./6. 4.)

Første undervisning efter påske.

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, som vi har diskuteret i dag (og i uge 6), fik I udleveret sidste uge: roborally-1.4.0. Opgave V4b er så ”bare” at tilpasse løsningen til jeres software.