HOT

Handcraft Objects Tool

4. Juni 2020

UNIVERSITETET I BERGEN

Oppbygning

- Hva er Handcraft Objects Tool?
- Hvordan verktøyet funker
- Videre arbeid

Hva er Handcraft Objects Tool?

Hva er HOT?

- Hvor kom ideen fra?

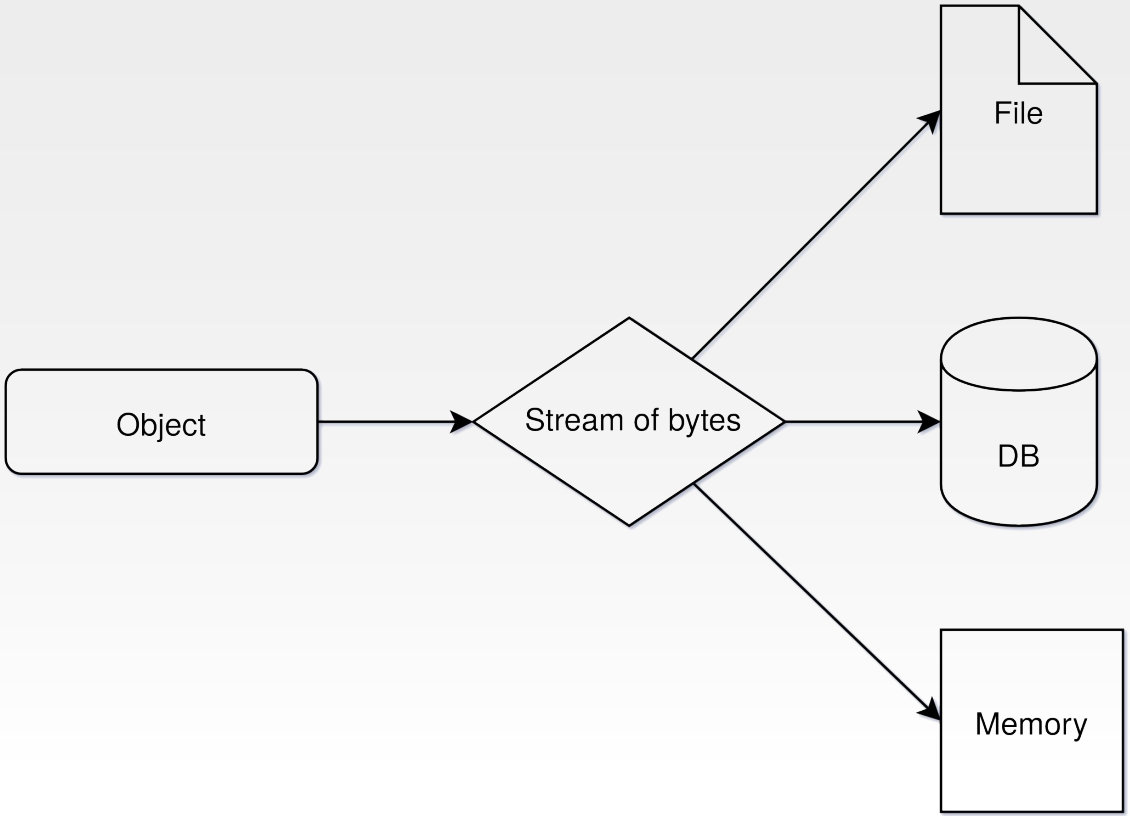
```
name: "Minimal CountdownObjective"
description: "An description of the quest is required"
objectives:
  - class: CountdownObjective
    finish_action: SUCCEED
    count_down_time_ms: 3000

```

```
#Showcase how to use the response_layout attribute
conversations:
  - name: "inject at last page"
    messages:
      - "test page! what do you say?"
      - "TEST page 2"
      - "last test page does Nice response!"
    responses:
      #inject text on the last page (index 2)
      - name: "LAST_PAGE test"
        response_layout: LAST_PAGE
        text: "This is the response! LAST_PAGE"

      #Inject text on page 2 (index 1)
      - name: "PAGE_NR test"
        response_layout: PAGE_NR
        layout_page_number: 0
        text: "This is the response! PAGE_NR"

      #create a new page (index 3)
      - name: "NEW_PAGE test"
        response_layout: NEW_PAGE
        text: "This is the response! NEW_PAGE"

      #create new page between last org page and the new page
      - name: "INSERT_BEFORE_PAGE test"
        response_layout: INSERT_BEFORE_PAGE
        layout_page_number: 1
        text: "This is the response! INSERT_BEFORE_PAGE"
```

Hva er HOT?

- Hvor kom ideen fra?
- Målet med prosjektet
  - Abstrahere bort detaljer
  - Dokumentere

Serialisering

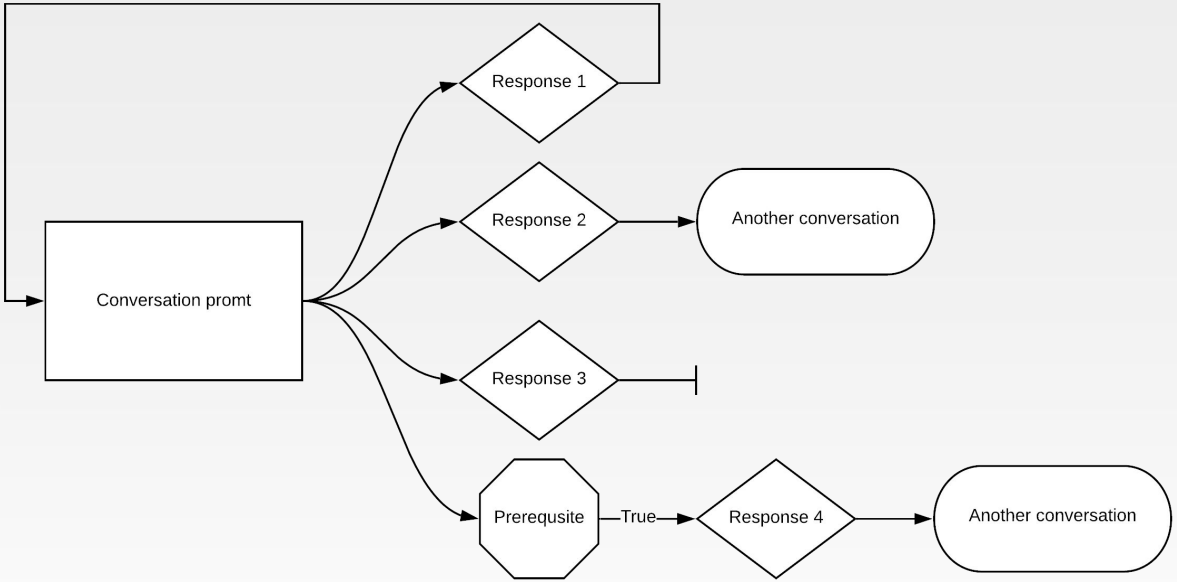


Dataformat i serialisering

```
{
  "type": "java.io.File"
  "path": "/home/elg/Documents/java/HandCraftObjectsTool/jars"
}
```

```
00000000   AC ED 00 05  73 72 00 0C  6A 61 76 61  2E 69 6F 2E  46 69 6C 65  04 2D A4 45  ....sr..java.io.File.-.E
00000018   0E 0D E4 FF  03 00 01 4C  00 04 70 61  74 68 74 00  12 4C 6A 61  76 61 2F 6C  .......L..patht..Ljava/l
00000030   61 6E 67 2F  53 74 72 69  6E 67 3B 78  70 74 00 32  2F 68 6F 6D  65 2F 65 6C  ang/String;xpt.2/home/el
00000048   67 2F 44 6F  63 75 6D 65  6E 74 73 2F  6A 61 76 61  2F 48 61 6E  64 63 72 61  g/Documents/java/Handcra
00000060   66 74 4F 62  6A 65 63 74  73 54 6F 6F  6C 2F 6A 61  72 73 77 02  00 2F 78     ftObjectsTool/jarsw../x
```

Samtale eksempel



Conversation Properties		
Property Name	Type	Required
prompt	String	Yes
responses	Collection of Responses	No

Response Properties		
Property Name	Type	Required
text	String	Yes
prerequisite	Prerequisite	No
conversation	Conversation	No

Hvordan serialisere et objekt

- Egenskaper
- Type informasjon

Response Properties	
Property Name	Type
text	String
prerequisite	Prerequisite
conversation	Conversation

```
class Response(
    val text: String,
    val prerequisite: Prerequisite,
    val conversation: Conversation
) {}
```

Interface!

Hvordan serialisere et objekt

- Egenskaper
- Type informasjon

```
class Response(
    val text: String,
    val prerequisite: Prerequisite,
    val conversation: Conversation
) {}
```

```
{
  "prerequisite": {
    "class": "no.uib.inf219.example.data.prerequisite.AlwaysTruePrerequisite"
  },
  "text": "Some text",
  "conversation":…
}
```

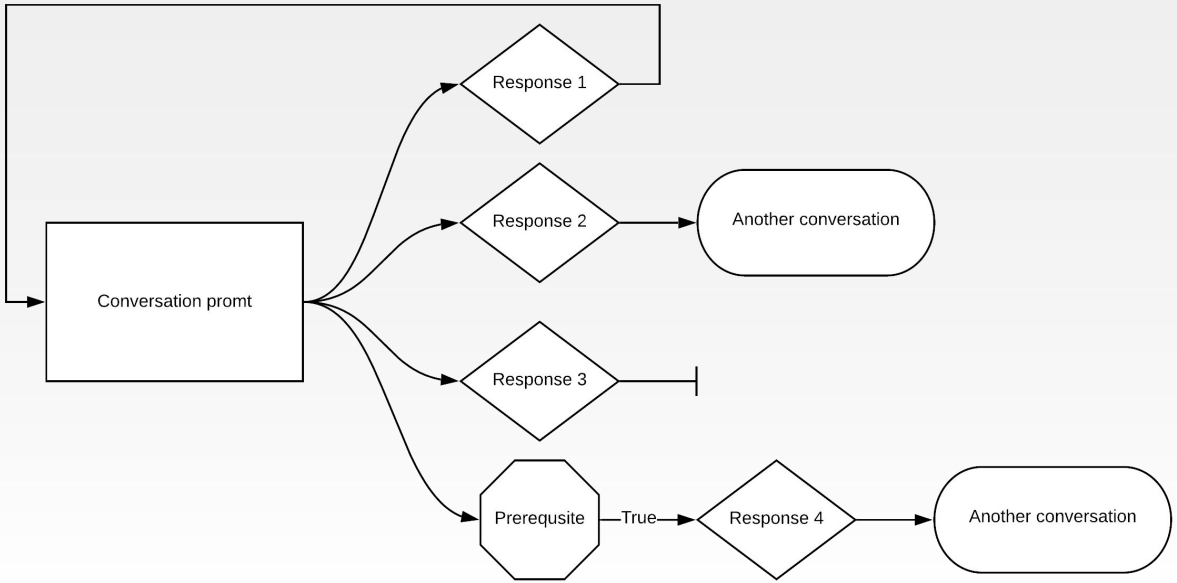
Hvordan serialisere en egenskap

- Egenskaper
- Type informasjon
- Identifikasjon

Conversation Properties		
Property Name	Type	Required
prompt	String	Yes
responses	Collection of Responses	No

Response Properties		
Property Name	Type	Required
text	String	Yes
prerequisite	Prerequisite	No
conversation	Conversation	No

Demo av eksempel



Hvordan fungerer HOT?

Jackson

- Databehandling
- Konfigurerbar
- ObjectMapper

1 minute tutorial: POJOs to JSON and back

The most common usage is to take piece of JSON, and construct a Plain Old Java Object ("POJO") out of it. So let's start there. With simple 2-property POJO like this:

```
// Note: can use getters/setters as well; here we just use public fields directly:
public class MyValue {
  public String name;
  public int age;
  // NOTE: if using getters/setters, can keep fields `protected` or `private`
}
```

We will need a `com.fasterxml.jackson.databind.ObjectMapper` instance, used for all data-binding, so let's construct one:

```
ObjectMapper mapper = new ObjectMapper(); // create once, reuse
```

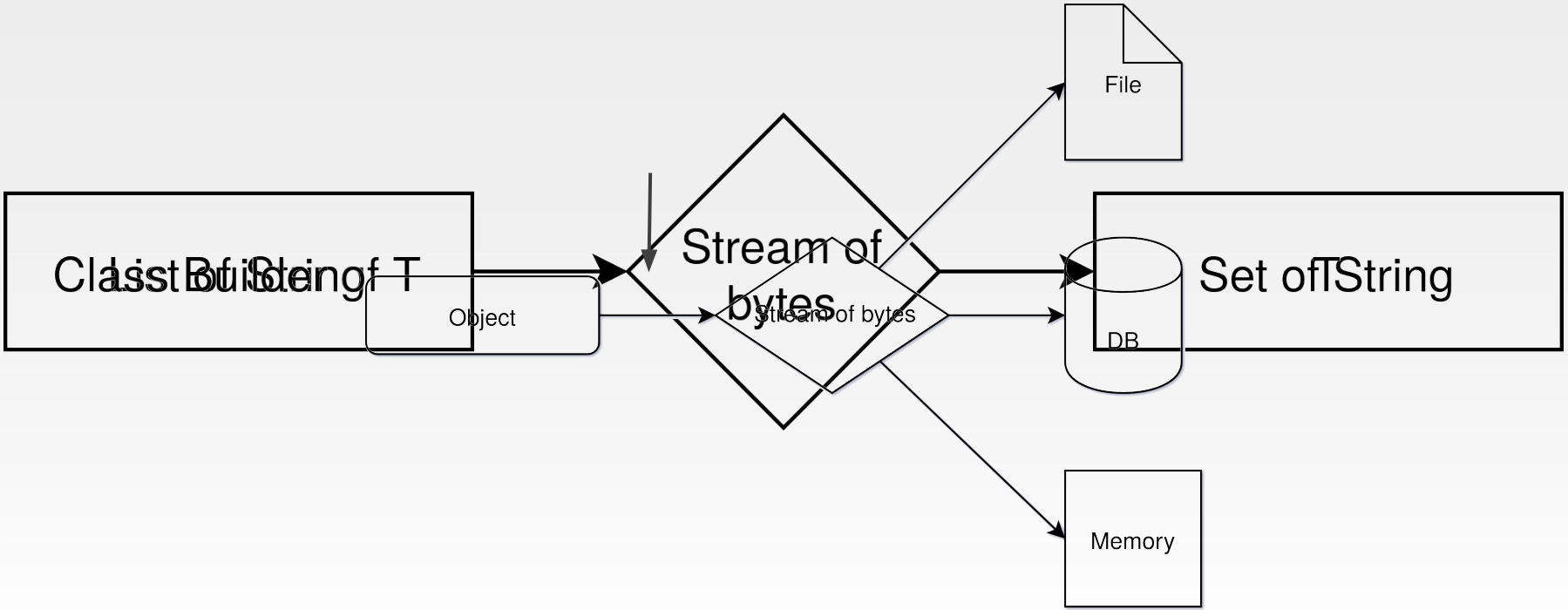
The default instance is fine for our use -- we will learn later on how to configure mapper instance if necessary. Usage is simple:

```
MyValue value = mapper.readValue(new File("data.json"), MyValue.class);
// or:
value = mapper.readValue(new URL("http://some.com/api/entry.json"), MyValue.class);
// or:
value = mapper.readValue("{\"name\":\"Bob\", \"age\":13}", MyValue.class);
```

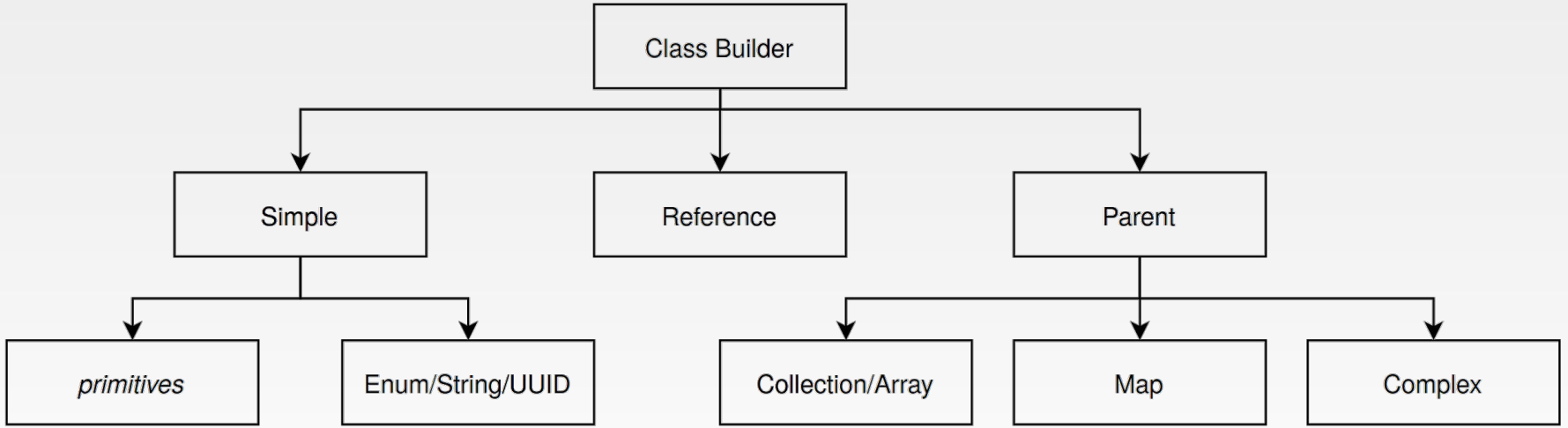
And if we want to write JSON, we do the reverse:

```
mapper.writeValue(new File("result.json"), myResultObject);
// or:
byte[] jsonBytes = mapper.writeValueAsBytes(myResultObject);
// or:
String jsonString = mapper.writeValueAsString(myResultObject);
```

Konvertering av objekter



Klassebyggere



Klassebygger interfacet

```
@JsonIdentityInfo(generator = ObjectIdGenerators.UUIDGenerator::class)
@JsonSerialize(using = ClassBuilderSerializer::class)
interface ClassBuilder {

    /**
     * The object to serialize
     */
    val serObject: Any
    /**
     * The expected [JavaType] this class builder will be serialized to
     */
    @get:JsonIgnore
    val type: JavaType

    /**
     * The parent of this class builder in the property graph.
     * If [parent] the same as `this`, this is the root builder
     *
     * @see no.uib.inf219.gui.backend.cb.isDescendantOf
     * @see ObjectEditorController.RootDelegator
     */
    @get:JsonIgnore
    val parent: ParentClassBuilder

    /**
     * Key of the property to access this from this [parent]
     */
    @get:JsonIgnore
    val key: ClassBuilder
```

String klassebygger

```
/**
 * Note that the default value is the empty String `""` and not the default value `null`
 */
class StringClassBuilder(
    initial: String = "",
    key: ClassBuilder?,
    parent: ParentClassBuilder?,
    property: ClassInformation.PropertyMetadata? = null,
    immutable: Boolean = false,
    item: TreeItem<ClassBuilderNode>
) : SimpleClassBuilder<String>(String::class, initial, key, parent, property, immutable, StringStringConverter, item) {

    override fun createEditView(
        parent: EventTarget,
        controller: ObjectEditorController
    ) = parent.textarea {
        bindStringProperty(textProperty(), converter, serObjectObservable)
    }
}
```

Kompleks klassebygger

```
@JsonSerialize(using = ComplexClassBuilderSerializer::class)
class ComplexClassBuilder(override val type: JavaType, override val key: ClassBuilder, override val parent: ParentClassBuilder,

    internal val propInfo: Map<String, ClassInformation.PropertyMetadata>

    //...

    init {
        val (typeSer, propInfo, valueDelegator) = ClassInformation.serializableProperties(type)

        //...

        //initiate all valid values to null or default
        for ((key, v) in this.propInfo) {

            //...

            } else if (v.hasValidDefaultInstance()) {
                //only create a class builder for properties that has a default value
                // or is primitive (which always have default values)
                this.createChild(key.toCb(), item = TreeItem())
            } else {
                this.serObject[key] = null
            }
        }
```

Demo av HOT

Hvor kommer klassene fra?

- Dynamisk klasse laster
- Jackson verdi serialiserer
- Egenskap Metadata

```
object DynamicClassLoader : URLClassLoader(emptyArray()) {

    /**
     * Load all classes from the given [File], if file is a
     *
     * @param file The file to load
     */
    fun loadFile(file: File) {
        try {
            addURL(file.toURI().toURL())
        } catch (e: Exception) {
            LoggerView.log("Failed to load jar file ${file
            LoggerView.log("$e")
            e.printStackTrace()
        }
        LoggerView.log("Successfully loaded jar file ${fil
    }
```

Hvor kommer egenskapene fra?

- Dynamisk klasse laster
- Jackson verdi serialiserer
- Egenskap Metadata

Hvor kommer egenskapene fra?

- Dynamisk klasse laster
- Jackson verdi serialiserer
- Egenskap Metadata

```
data class PropertyMetadata(
    val name: String,
    val type: JavaType,
    val defaultValue: String,
    val required: Boolean,
    val description: String,
    val virtual: Boolean
) {
    private var validDefInst: Boolean? = null

    fun hasValidDefaultInstance(): Boolean {
        if (validDefInst == null) {
            validDefInst = getDefaultInstance() != null
        }
        return validDefInst!!
    }
```

Serialisering av klassebyggere

- Generelt
- Kart
- Kompleks

```
override fun serializeMaybeWithType(
    cb: MapClassBuilder,
    gen: JsonGenerator,
    provider: SerializerProvider,
    typeSer: TypeSerializer?
) {
    val map = cb.serObject.map {
        //find the real serializer and delegate to it
        val ser: JsonSerializer<Any> = provider.findValueSerializer(cb.serObject::class.java)
        // ?: error("Failed to find serializer for ${cb.serObject}")
        //This is because jackson is using a different type of serializers with keys as not everything is allowed
        // to be a key in json. So to make this happen we need to make key serializers for all existing cb
        // serializers and use in those to allow for referencing.
        if (cb.serObject === cb) {
            error("Endless cycle detected, class builder is referencing it self")
        }
        //value is allowed to be referenced and be a reference!
        val value = it.serObject[ENTRY_VALUE]
        if (typeSer != null)
            ser.serializeWithType(cb.serObject, gen, provider, typeSer)
        else
            ser.serialize(cb.serObject, gen, provider)
        key to value
    }.toMap()

    val ser = provider.findValueSerializer(map::class.java, null) ?: error("Failed to find map serializer")

    if (typeSer != null)
```

Videre arbeid

Laste inn objekter med referanser

- Hvorfor er dette et problem nå?
- Hvordan kan dette bli løst?

Tredjeparts klassebyggere

- Hvorfor er dette et problem nå?
- Hvordan kan dette bli løst?

Egendefinert validering

- Hvordan?
  - Annoteringer med JSR 380
  - Json schema validation

```
@Min(1)
@Max(20)
private Int id;
```

```
"id": {
  "type": "number",
  "minimum": 1,
  "maximum": 20
}
```

Andre små justeringer

- Laste inn klasser med json schema
- Definere nye klasser i HOT
- Andre serialiserings bibliotek
- Skille logikk fra det visuelle


UNIVERSITAS
BERGENSIS