

Goal:

Deliver a schema that is according the JsonTypeInfo annotation using a wrapper object.

Works fine for Animals:

After adding a CustomDefinitionProvider with the following code this work pretty fine:

```
@Override
public CustomDefinition provideCustomSchemaDefinition(ResolvedType javaType,
SchemaGenerationContext context) {

    JsonTypeName = javaType.getErasedType().getAnnotation(JsonTypeName.class);
    if (jsonTypeName != null) {
        ObjectNode typeNode = context.createStandardDefinition(javaType, this);
        ObjectNode typeWrapNode = objectMapper.createObjectNode();
        typeWrapNode.set(jsonTypeName.value(), typeNode);
        return new CustomDefinition(typeWrapNode);
    }
}
```

The following Java POJO:

```
@JsonTypeInfo(
    use = JsonTypeInfo.Id.NAME,
    include = JsonTypeInfo.As.WRAPPER_OBJECT)
@JsonSubTypes({
    @JsonSubTypes.Type(value = Cat.class, name = "Cat"),
    @JsonSubTypes.Type(value = Dog.class, name = "Dog")
})
abstract static class Animal {
    public int furLength;
}

@JsonTypeName("Cat")
static class Cat extends Animal {
    public String miauwSound;
}

@JsonTypeName("Dog")
static class Dog extends Animal {
    public String barkSound;
}
```

Gives this schema:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "definitions": {
    "Animal": {
      "anyOf": [
        {
          "Cat": {
            "type": "object",

```

```
        "properties": {
          "furLength": {
            "type": "integer"
          },
          "miauwSound": {
            "type": "string"
          }
        }
      },
    },
    {
      "Dog": {
        "type": "object",
        "properties": {
          "barkSound": {
            "type": "string"
          },
          "furLength": {
            "type": "integer"
          }
        }
      }
    }
  ]
},
"type": "object",
"properties": {
  "animal": {
    "$ref": "#/definitions/Animal"
  },
  "animals": {
    "type": "array",
    "items": {
      "$ref": "#/definitions/Animal"
    }
  },
  "someField": {
    "type": "string"
  }
}
}
```

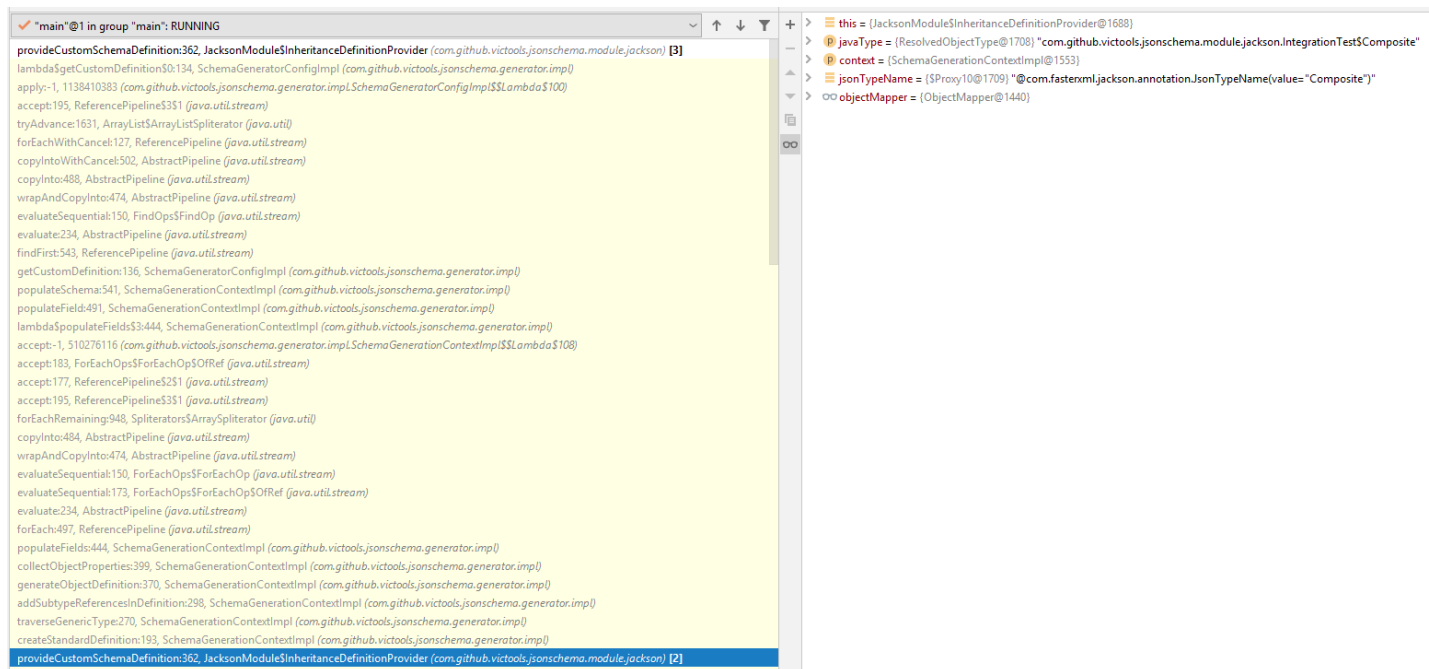
So far so good!

But a standard composite pattern that we are using gives problems:

The following code puts the generator in an endless recursion:

```
@JsonTypeInfo(  
    use = JsonTypeInfo.Id.NAME,  
    include = JsonTypeInfo.As.WRAPPER_OBJECT)  
@JsonSubTypes({  
    @JsonSubTypes.Type(value = Leaf.class, name = "Leaf"),  
    @JsonSubTypes.Type(value = Composite.class, name = "Composite")  
})  
abstract static class Component {  
    public String id;  
    public Composite parent;  
}  
  
@JsonTypeName("Leaf")  
static class Leaf extends Component {  
    public String content;  
}  
  
@JsonTypeName("Composite")  
static class Composite extends Component {  
    public Set<Component> children;  
}
```

Frames of the stack showing one cycle of the endless recursion:



The screenshot shows a stack trace in an IDE. The left pane displays the stack trace, and the right pane shows the current frame's variables. The stack trace shows a cycle of recursive calls between `provideCustomSchemaDefinition` and `populateFields` in the `SchemaGeneratorContextImpl` class. The right pane shows the following variables:

- `this` = `(JacksonModule$InheritanceDefinitionProvider@1688)`
- `javaType` = `(ResolvedObjectType@1708) "com.github.victools.jsonschema.module.jackson.IntegrationTest$Composite"`
- `context` = `(SchemaGenerationContextImpl@1553)`
- `jsonTypeName` = `(SProxy10@1709) "@com.fasterml.jackson.annotation.JsonTypeName(value="Composite")"`
- `objectMapper` = `(ObjectMapper@1440)`

Key elements:

Call for the standard type definition, excluding this CustomDefinitionProvider:

```
JacksonModule.java x
356
357 @Override
358 public CustomDefinition provideCustomSchemaDefinition(ResolvedType javaType, SchemaGenerationContext context) {
359
360     JsonTypeName jsonTypeName = javaType.getErasedType().getAnnotation(JsonTypeName.class); jsonTypeName: "@com
361     if (jsonTypeName != null) { jsonTypeName: "@com.fasterxml.jackson.annotation.JsonTypeName(value="Composite"
362     ObjectNode typeNode = context.createStandardDefinition(javaType, ignoredDefinitionProvider: this); context: S
```

Call for a definition of a type that is calling the same CustomDefinitionProvider, where this is not expected.

```
SchemaGenerationContextImpl.java x
528 }
529
530 /**
531  * Preparation Step: combine the collected attributes and the javaType's definition in the given targetNode.
532  *
533  * @param javaType field's type or method return value's type that should be represented by the given targetNode
534  * @param targetNode node in the JSON schema that should represent the associated javaType and include the separately collected attributes
535  * @param isNullable whether the field/method's return value the javaType refers to is allowed to be null in the declaringType isNullable:
536  * @param collectedAttributes separately collected attribute for the field/method in their respective declaring type collectedAttributes:
537  * @see #populateField(FieldScope, Map, Set)
538  * @see #populateMethod(MethodScope, Map, Set)
539  */
540 @ private void populateSchema(ResolvedType javaType, ObjectNode targetNode, boolean isNullable, ObjectNode collectedAttributes) { javaType:
541     final CustomDefinition customDefinition = this.generatorConfig.getCustomDefinition(javaType, context: this, ignoredDefinitionProvider: null);
542     if (customDefinition != null && customDefinition.isMeantToBeInline()) {
```

Please advice on a solution forward.

Regards,

Jan