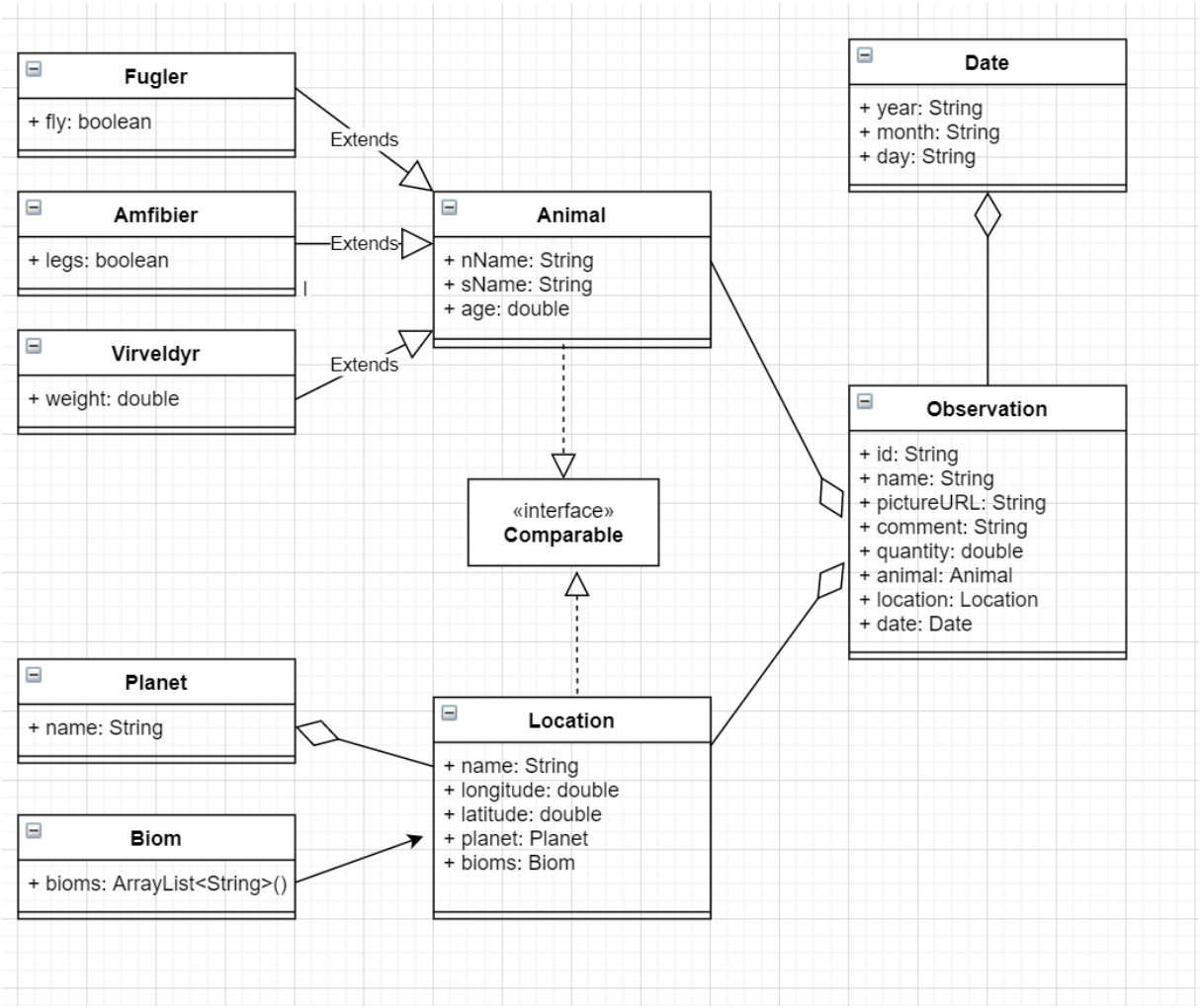
Programmering 2, Oblig 6

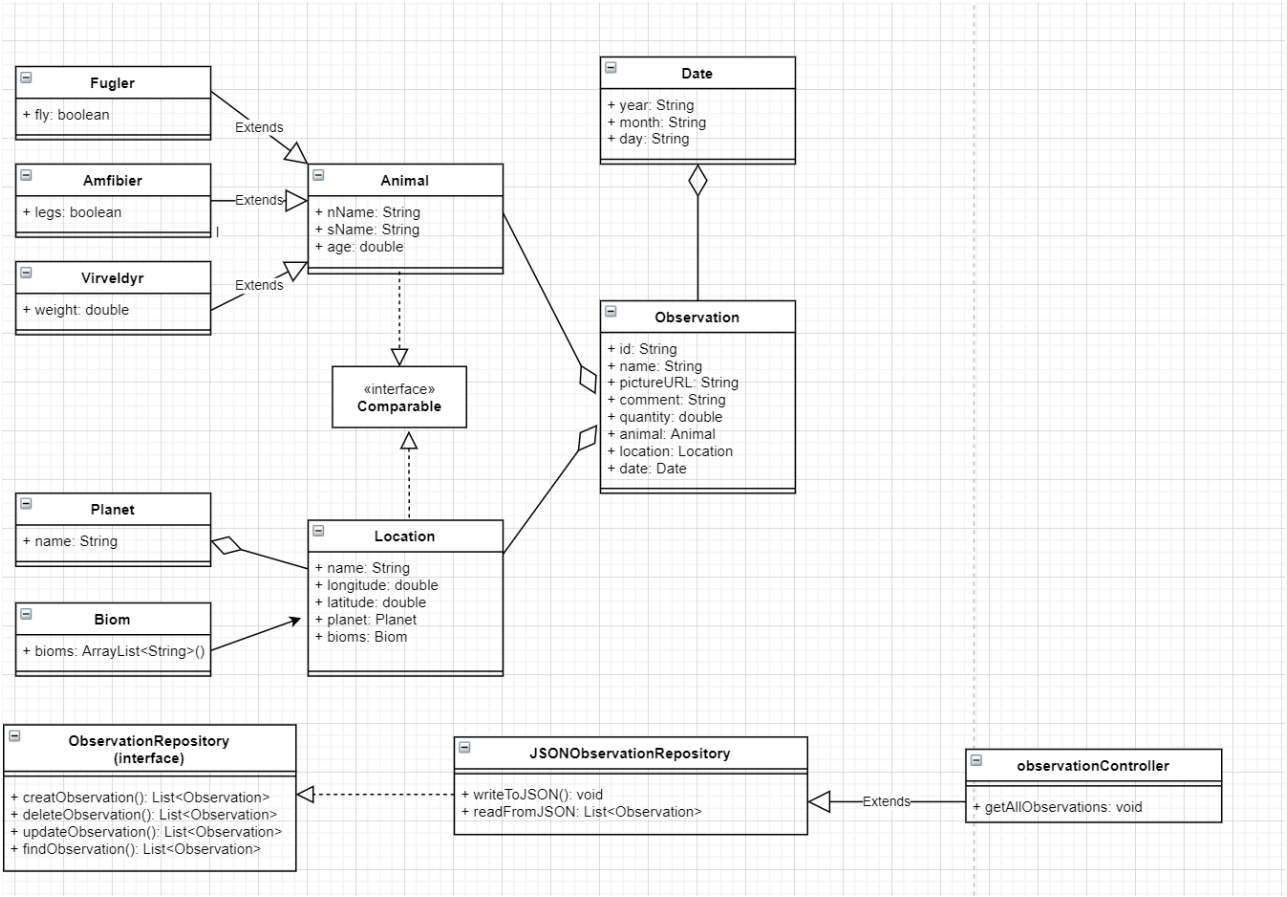
Rapport

UML for klassene:



Ved å lese beskrivelsen kom jeg frem til denne UMLen for prosjektet. Her lagde jeg først de små dyre klassene. Disse *name* og *age* som felles trekk, så jeg lagde en felles abstrakt klasse ovenfor som blir ekstender av de mindre dyreart klassene. Det fører da til at dyreart klasse konstruktørene vil inkludere *name* og *age*, men ved hjelp av super vil de kunne «samles» inn i Animal-klassen. Deretter trenger man en Location-klasse som inneholder en planet og én eller flere biomer. Da må man valgte jeg å opprette egne klasse for både Planet og Biom, som har kun de nyttige variablene. Jeg bruker en aggregasjon mellom Location og Planet-klassen siden en planet kan ha flere lokasjoner, men en lokasjon kan kun ha en planet. Mellom Biom og Location bruker jeg en «normal» relasjons pil, siden både et biom kan ha flere lokasjoner og en lokasjon kan ha flere biomer. Så implementerer jeg både Location og Animal til Comparable, slik at man kan bruke compareTo for å sortere. Tilslutt må man ha en Observation-klasse, som blir aggregert av både Location og Animal, siden en observasjon trenger en animal og lokasjon for å bli opprettet. Så er det koblet til Date, som er en egen klasse for dato (kommer frem til det senere).

Fullstendig UML:



Ovenfor ser man den fullstendige UML, med Repository med Controller. Deet jeg ser nå er at jeg heller skulle ha koblet kontroller til repositoriet via main i stedet for å ekstende fra kontrolleren til repositoriet. Nå vil jeg kunne ha tilgang til unødvendig metoder som kommer både fra interfacet og repositroriet.

Starten av denne oppgaven var nokså rett frem, siden det var delvis likt grunnlag som de forrige oppgavene. I starten lagde jeg alle dyre klassene for seg selv uten å implementere noe som helst. Flere ganger gjennom kode prosessen merket jeg at jeg ikke hadde lest oppgaven godt nok og endte med å endre på ting. Dette var en av gangene, og endte med å lage en abstrakt Animal-klasse for å forenkle kodedelen, og gjøre det mer oversiktlig. Deretter fortsatte jeg med å opprette de andre klassene. Tostringen var nokså rett frem, og jeg lagde en relativt lik toString til alle klassene, som er på array-form. For å kunne sortere noen ting valgte jeg klassene Location og Animal. Disse implementerte jeg til Comparable og lagde en annen override metode for compareTo, hvor jeg sammenligner alder og latitude. Så opprettet jeg noen observasjoner i main, og skrev det ut.

Neste del av koding dreier seg hovedsakelig om repositoriet. Først må man lage et interface med metoder som var aktuelt for create, delete og update. I tillegg lagde jeg metode for

findObservation, for å kun peke frem til én observasjon. Selve CRUD metodene var nokså enkelt, hvor man hovedsakelig kun trengte gettere og settere. Jeg valgte JSON med «*List*» collection siden det er nok det mest enkle og forståelige. Det fungerer hovedsakelig slik som array, men har mange flere fordeler. Det gir brukeren muligheten til å ha kontroll over hvilken posisjon elementet settes inn i og gir tilgang til elementene ved hjelp av index. Dette brukte jeg til å lage en lese og skrive metode for JSON, ved hjelp av *objectMapper*. I hovedsak hadde jeg brukt *LocalDateTime*, og hadde fått inn compile delen, men alt gikk i vasken når det kom til å leseFraJSON. Jeg hadde lagt inn dependency i objekt mapper, men fikk feil etter feil. Prøvde å skifte fra *LocalDateTime.now* til *LocalDateTime.of*, men funket fortsatt ikke og endte opp med feilmeldingen nedenfor.

```
com.fasterxml.jackson.databind.exc.InvalidDefinitionException: Cannot construct instance of `java.time.LocalDateTime` (no Creators, like
 default construct, exist): no String-argument constructor/factory method to deserialize from String value ('2020-03-20T05:03:30')
 at [Source: (File); line: 19, column: 12] (through reference chain: java.lang.Object[][0]->no.hiof.janaathm.model.Observation["date"])
    at com.fasterxml.jackson.databind.exc.InvalidDefinitionException.from(InvalidDefinitionException.java:67)
    at com.fasterxml.jackson.databind.DeserializationContext.reportBadDefinition(DeserializationContext.java:1592)
    at com.fasterxml.jackson.databind.DeserializationContext.handleMissingInstantiator(DeserializationContext.java:1058)
    at com.fasterxml.jackson.databind.deser.ValueInstantiator._createFromStringFallbacks(ValueInstantiator.java:371)
    at com.fasterxml.jackson.databind.deser.std.StdValueInstantiator.createFromString(StdValueInstantiator.java:323)
    at com.fasterxml.jackson.databind.deser.BeanDeserializerBase.deserializeFromString(BeanDeserializerBase.java:1373)
    at com.fasterxml.jackson.databind.deser.BeanDeserializer._deserializeOther(BeanDeserializer.java:171)
    at com.fasterxml.jackson.databind.deser.BeanDeserializer.deserialize(BeanDeserializer.java:161)
    at com.fasterxml.jackson.databind.deser.impl.MethodProperty.deserializeAndSet(MethodProperty.java:129)
    at com.fasterxml.jackson.databind.deser.BeanDeserializer.vanillaDeserialize(BeanDeserializer.java:288)
```

Dette førte til at jeg måtte skrote LocalDate, og gikk heller over til å lage en egen klasse, «Date» som tar år, måned og dag som parameter, for å lage en string dato. Dette førte til at jeg måtte gå gjennom nesten hele koden igjen for å kunne erstatte variablene. Skrive til JSON funket fint med endringene, og lesingen fungerte også nå. Deretter lastet jeg ned vue-filen man hadde fått og koblet den opp ved hjelp layout-filen jeg hadde fra de tidligere obligatoriske oppgavene. Så startet jeg opp javalin ved å
Javalin app = Javalin.create().start(7100);
app.config.enableWebjars();

Så brukte jeg *app.get* til å koble opp vue og lage pather til local host. Deretter lager jeg en handler for API, hvor jeg kobler til kontrolleren, som får tak i observasjonene fra JSON filen, slik at det kan vises på nettstedet. Siden jeg hadde brukt de rette variabel og klassenavnene greide vue-filen å kobles til med en gang uten noen problemer, og API hadde fått inn observasjonene.