Rapport – AeroScript (IN2031 Oblig 2)

Innholdsfortegnelse

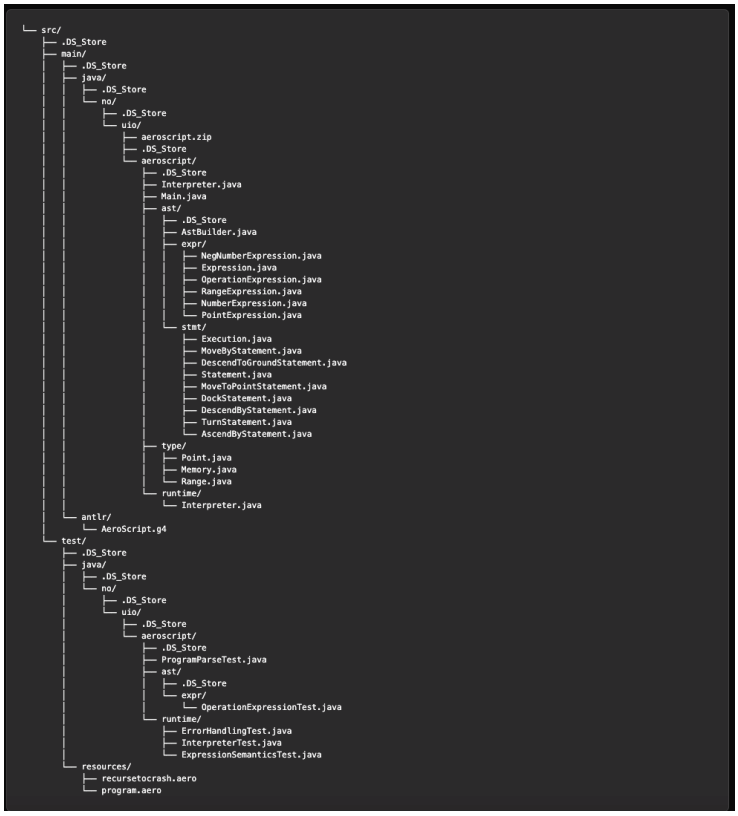
1. Introduksjon

Målet har vært å implementere AeroScript med ANTLR (lexer og parser), en enkel og robust interpreter som evaluerer uttrykk, samt JUnit-tester for både normalbruk og feilhåndtering. Løsningen bygger grønt med "./gradlew clean build", og de gitte testene går gjennom. I tillegg er det lagt til egne små tester for semantikk og feiltilfeller. Rapporten beskriver valg i grammatikk og implementasjon, håndtering av unær minus (negasjon), feilhåndtering, testdekning og forslag til videre arbeid.

2. Arkitektur og prosjektstruktur

Bygg og verktøy:

– Prosjektet bruker Gradle og ANTLR. Grammatikken ligger i "src/main/antlr/AeroScript.g4". ANTLR genererer lexer og parser til "build/generated-src/antlr/...".



Figur 1: lagd via sickrate.com/folder-structure-generator

Kodepakker:

– no.uio.aeroscript: evaluerer uttrykk direkte fra ANTLR sitt parse-tre.

– no.uio.aeroscript.runtime: en lett adapter for testene. Den eksponerer metoden som testene forventer (visitExpression(...).evaluate()).

– no.uio.aeroscript.ast.expr: uttrykksklasser fra precoden (gjenbrukt der det passer).

– no.uio.aeroscript.type: hjelpetypene Range, Point og Memory fra precoden.

Begrunnelse for adapter:

De gitte testene forventer en bestemt klasse og metodeform i pakken no.uio.aeroscript.runtime. For å holde selve tolken enkel og lettlest, er testspesifikke detaljer samlet i en tynn adapter som bare oversetter kall og returnerer et objekt med evaluate(). Dette reduserer koblingen mellom testene og kjernelogikken.

3. Grammatikk (AeroScript.g4)

Grammatikken følger Figur 1 for uttrykksdelen og dekker:

– Tall (heltall/desimal), parenteser.

– Punkt: point (e, e).

– Intervall: [e, e] for range.

– random, både uten intervall (tilsvarer [0,1]) og med eksplisitt intervall [lo, hi].

– Operatorer: unær minus (negasjon), gangetegn, pluss og minus.

– Presedens: unær minus har høyest presedens, deretter multiplikasjon, deretter addisjon/subtraksjon. Binære operatorer er venstreassosiative.

4. Unær minus (negasjon)

Unær minus betyr negasjon av ett uttrykk eller tall. Eksempler: –x, –(2+3). I dette prosjektet er negasjon modellert med "NEG" som skrives "--" foran uttrykk, for å holde grammatikken tydelig atskilt fra binær minus. Det viktige er at negasjon har høyere presedens enn multiplikasjon og addisjon/subtraksjon. Evalueringskoden er gjort robust for å håndtere at ANTLR kan produsere litt ulik trestruktur for prefiks-operatorer. Praktiske eksempler som testes: "-- 1" gir –1 og "-- (2 + 3)" gir –5.

5. Interpreter og semantikk

Interpreter går direkte på ANTLR-parse-treet for å evaluere uttrykk. Dette gir lite kode og få feilflater i en liten oppgave.

Støttet semantikk:

– Tall og parenteser evalueres til flyttall.

– Unær minus negere resultatet av underuttrykket.

– Multiplikasjon, addisjon og subtraksjon evaluerer venstre og høyre operand og utfører operasjonen med korrekt presedens.

– point (x, y) skaper en punktverdi som ikke kan brukes direkte som tall.

– range [lo, hi] evaluerer grensene til et intervallobjekt.

– random uten intervall gir et flyttall i [0,1]. random [lo, hi] gir et flyttall i [lo, hi].

Begrunnelse for direkte parse-tre-tolk:

Det er fullt mulig å bygge en egen AST og tolke den, men i denne oppgaven gir direkte evaluering fra parse-treet en enklere løsning som likevel oppfyller kravene og fungerer godt med testene. Dersom det senere ønskes mer semantikk på programnivå (moduser og statements), er det enkelt å legge til en AstBuilder og en program-AST uten å endre uttrykksdelen.

6. Feilhåndtering

Det kastes meningsfulle exceptions for feilsituasjoner:

– Punkt i numerisk kontekst (for eksempel "point(2,3) + 1") kaster IllegalArgumentException.

– Ugyldig random-intervall (for eksempel "random [3,2]") kaster IllegalArgumentException.

– Uventet trestruktur ved ugyldig input kaster IllegalStateException.

Dette gjør det enkelt å skrive negative tester og sikrer tydelige feilmeldinger.

7. Testing

Gitte tester:

Prosjektet inkluderer testene fra handouten, som dekker aritmetikk, unær minus og at visitExpression(...).evaluate() gir korrekte resultater.

Egne tester:

– ExpressionSemanticsTest: verifiserer presedens (2 + 3 * 4 = 14), unær minus (--(2+3) = –5), og at punkt i numerisk kontekst kaster exception.

– ProgramParseTest: enkel parser-test for et representativt uttrykk, for å bekrefte at lexer og parser spiller sammen uten antakelser om full programsyntaks.

8. Utfordringer og løsninger

– ANTLR error(50) oppstod tidlig på grunn av blanding av lexer- og parserregler og feilplasserte symboler. Løsning var å rydde i grammatikken, skille tokenregler og parserregler tydelig, og kjøre "generateGrammarSource" jevnlig for å teste små endringer.

– Unær minus (negasjon) krevde nøye presedenshåndtering og robust evaluering fordi prefiks-operatorer kan gi variasjoner i parse-tre. Løsningen var en klar regel for negasjon og tolerante sjekker i tolken.

– Test-API-kravene ble løst med en egen adapterklasse i pakken no.uio.aeroscript.runtime, slik at kjernen av tolken kan være enkel og gjenbrukbar.

9. Konklusjon og videre arbeid

Løsningen implementerer uttrykksdelen av AeroScript i tråd med Figur 1. Den bygger grønt og består både gitte og egne tester. Grammatikken håndterer korrekt presedens og støtter tall, punkt, intervall, random, unær minus, multiplikasjon, addisjon og subtraksjon. Tolken er enkel og robust, med klar feilhåndtering.

Mulig videre arbeid:

Bygge en full AST for program/modus/statements og en AstBuilder som oversetter parse-treet til AST, og deretter tolke AST.

Utvide semantikk og feilmeldinger, for eksempel for variabler eller strengere typer.

Gjøre random deterministisk i tester (fast seed) for helt reproduserbare resultater.

Flere parser-tester på programnivå dersom ønsket av sensor.