INFO233: Obligatorisk oppgave 4

Innleveringsfrist: 22. mai, kl. 14:00

Trær, oppslagsverk, og kodeforståelse

Oppgaven denne gangen er å implementere et oppslagsverk som et sortert tre, altså et `SortedTreeMap`. Du kan se hvordan det skal se ut i det vedlagte grensesnittet. Du må selv velge hvilken type tre du skal bruke til å implementere oppslagsverket (`Map`), e.g. `Red-Black Tree`. Du finner også et testsett vedlagt. Du får ikke poeng i denne oppgaven, i stedet så må du besvare alle testene som er vedlagt oppgaven. Testene er skrevet i en stil som heter egenskapsbasert testing (property-based testing). En del av oppgaven er å lese, og forså godt nok, hva testene ber om slik at du kan skrive implementasjonen til det. Det er nok en del uvant å lese, men et hint er å se etter `prop()` og/eller `implies()`. Det man gjør i egenskapsbasert testing er at man lager en spesifikasjon og så genererer man data som er ment til å verifisere den spesifikasjonen. Det er ikke meningen at du skal skrive slike tester selv til denne oppgaven. Se fillageret på mitt.uib for vedlagte filer. Det er kun lov til å bruke de delene av Java som er rimelig å bruke i utviklingen av datastrukturen som du skal implementere i denne oppgaven.

Denne oppgaven er et eksperiment i hvordan vi kan utføre obligatoriske oppgaver. Det er derfor viktig at du starter tidlig og gir tilbakemelding slik at vi kan justere og forbedre de delene som ikke er tydelig nok. Jeg forventer derfor at oppgaven kommer til å bli oppdatert i løpet av de nærmeste dagene etter utlevering.

For å bestå oppgaven så må du bestå alle testene i testsettet.

Om du ikke skulle bestå, så tilbyr vi egenretting. Det er ingen grense for hvor mye du må ha gjort for å få egenretting denne gangen.

Hvordan fungerer egenskapsbasert testing

Slik egenskapsbasert testing fungerer er at vi definerer generatorer som genererer tilfedig data innenfor en spesifikasjon som vi gir. Vi lager så tester som bruker denne dataen til å sjekke om systemet vårt svarer på ulike egenskaper.

Implementasjonen jeg har brukt til å skrive testene i til denne oppgaven er functionaljava sin quickcheck (org.functionaljava:functionaljava-quickcheck:4.7). Du kan bruke pakkebehandleren til intellij for å laste ned dette biblioteket. Dokumentasjonen finner du her `http://www.functionaljava.org/javadoc/4.7/functionaljava-quickcheck/index.html`

Slik jeg anbefaler å jobbe med testene er å kjøre 1 test om gangen. Det kan du gjøre i intellij ved å trykke på den grønne spill-avknappen på venstre side vedsidenav testen. Når du kjører den så vil du få et sett med elementer og en feilmelding når testen ikke virker, og en melding om at testene var vellykket hvis alt gikk bra. Det kan være det står "7 discarded" (eller fler); det betyr bare at det var noen datapunkt som ikke svarte til implikasjonen i den testen. Du kan ignorere dette med mindre tallet er veldig høyt og testen sier den ga helt opp og ingen tester ble vellykket.

Det er også lurt å opprette sin egen test-fil slik at du kan ta de datapunktene som testene sier ikke virker og teste de selv. Da har du mer kontroll over hva som kan gå gale. Du slipper også å oppdatere test-settet hvis jeg skulle finne noen feil eller skulle trenge å legge til noe. Jeg tror også i dette tilfelle at det er bedre å skrive vanlige jUnit tester ettersom du allerede har et punkt som feiler og ikke trenger å generere data.