Øving: Arv

IDATx2003 Programmering 2

Innhold

Innledning

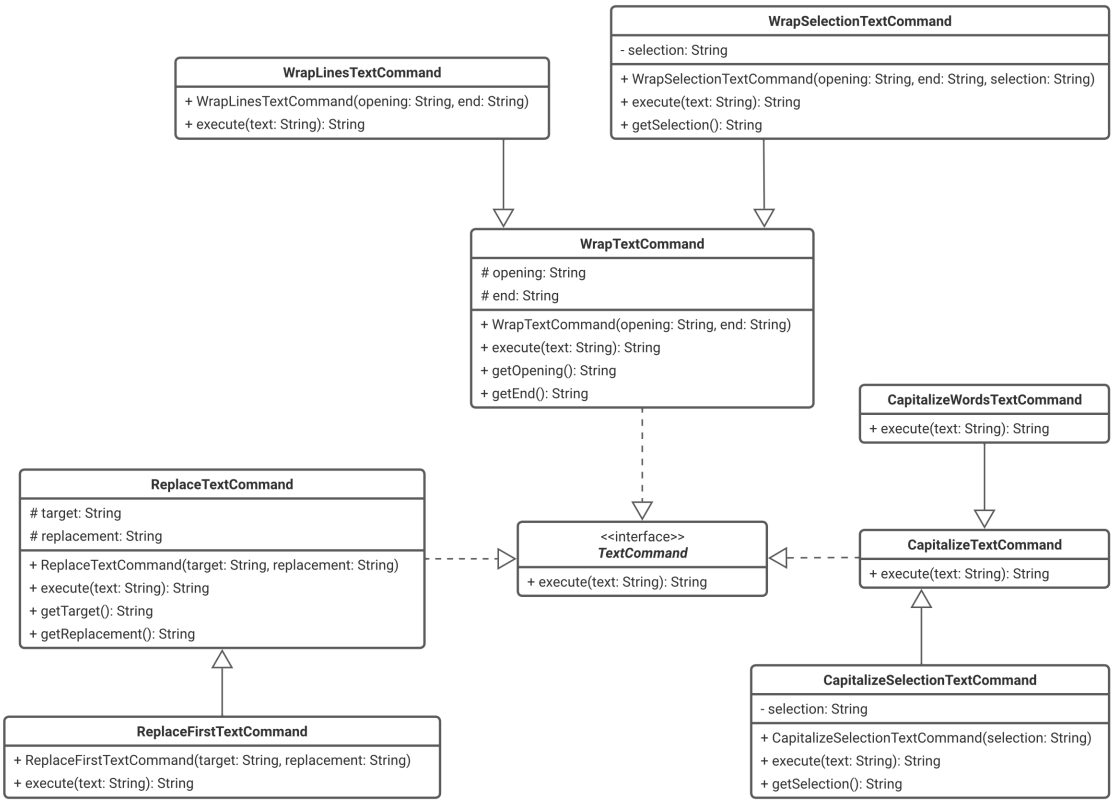
I denne øvinga skal du utvikle et enkelt API for å manipulere tekst. APIet skal ha funksjoner som gjør det mulig å:

- erstatte deler av en tekst (replace...)
- omslutte tekst med annen tekst (wrap...)
- endre tekst slik at enkelte ord får stor forbokstav (capitalize...)

Øvinga gjennomføres individuelt. Når du har løst oppgavene må du levere i BB og vise frem det du har gjort til en læringsassistent for godkjenning.

Vi kan tenke på de ulike funksjonene som kommandoer vi kan utføre på tekst. Vi velger å implementere hver kommando som en egen klasse, med en metode som utfører selve handlingen. Vi baserer oss på arv for å definere et logisk hierarki og gjenbruke kode. Fordelen med en slik løsning er at vi kan lagre kommandoer og utføre dem på et senere tidspunkt, som gjør det mulig å bygge mer avanserte skript.

Følgende diagram viser de viktigste klassene i kommandohierarkiet:



Figur 1: Klassediagram for kommandoer

Oppgave 1: Oppsett av prosjektet

Opprett et tomt **Maven-prosjekt** og gi det en fornuftig groupId og artefactId. Når du svarer på kodeoppgavene under skal filer lagres iht standard Maven-oppsett: enhetstester legges i katalogen "test/java", mens resten av koden hører hjemme i katalogen "main/java". Det skal være mulig å bygge, teste og pakke med Maven fra kommandolinja. Det betyr bl.a. at kommandoen «mvn clean package» skal kjøre uten feil.

Legg så prosjektet under **versjonskontroll**. Gjennomfør en første commit før du gyver løs på selve kodinga.

Følgende krav gjelder for for resten av øvinga:

- **Sjekk inn jevnlig**. Husk at hver commit-melding skal beskrive endringene som er gjort på en kort og konsis måte.
- Lag **enhetstester** for de mest sentrale klassene. Testene skal verifisere at APIet fungerer som forventet, og at feiltilfeller håndteres på riktig måte.
- Det er viktig med solid **unntakshåndtering**. Håndteringen skal være rimelig og balansert, med det formål å gjøre koden mer robust.

Oppgave 2: TextCommand

Et fellestrekk ved alle kommandoer er at de representerer en handling på en tekststreng. Det første du skal gjøre er derfor å implementere grensesnittet TextCommand. Grensesnittet definerer metoden execute(text), som returnerer en streng.

Oppgave 3: Replace-kommandoer

APIet skal ha to klasser som gjør det mulig å erstatte deler av en tekst.

ReplaceTextCommand

Klassen implementerer TextCommand og erstatter alle treff i en tekst. Den har attributter for teksten som skal erstattes (target) og selve erstatningen (replacement). Disse settes gjennom en konstruktør og brukes i execute-metoden. Du må også legge til aksessor-metoder, som vist i klassediagrammet.

Eksempel på bruk:

```
new ReplaceTextCommand("target", "replacement")
        .execute("text with target and target")
        → text with replacement and replacement
```

ReplaceFirstTextCommand

Klassen arver fra ReplaceTextCommand og erstatter kun det første treffet i en tekst. Den har en konstruktør som tar inn teksten som skal erstattes (target) og erstatningen (replacement).

Eksempel på bruk:

```
new ReplaceFirstTextCommand("target", "replacement")
        .execute("text with target and target")
        → text with replacement and target
```

Oppgave 4: Wrap-kommandoer

APIet skal ha tre klasser som gjør det mulig å omslutte tekst.

WrapTextCommand

Klassen implementerer grensesnittet TextCommand og omslutter en gitt tekst med annen tekst. Den har attributter for åpningsstrengen (opening) og avslutningsstrengen (end). Disse settes gjennom en konstruktør og brukes i execute-metoden. Husk get-metoder, som vist i klassediagrammet.

Eksempel på bruk:

```
new WrapTextCommand("<p>", "</p>")
        .execute("text to be wrapped")
        → <p>text to be wrapped</p>
```

WrapLinesTextCommand

Klassen arver fra WrapTextCommand og omslutter hver linje av en gitt tekst. Den har en konstruktør som tar inn åpningsstrengen (opening) og avslutningsstrengen (end).

Eksempler på bruk:

```
new WrapLinesTextCommand("<p>", "</p>")
      .execute("first line\nsecond line")
      → <p>first line</p>\n<p>second line</p>
```

WrapSelectionTextCommand

Klassen arver fra WrapTextCommand og omslutter et utvalg av en gitt tekst. Den har et attributt for utvalget som skal omsluttes (selection), en konstruktør som tar inn nødvendige parametre, samt aksessor-metoden getSelection().

Eksempel på bruk:

```
new WrapSelectionTextCommand("<p>", "</p>", "selection")
      .execute("text with selection")
      → text with <p>selection</p>
```

Oppgave 5: Capitalize-kommandoer

APIet skal ha tre klasser som gjør det mulig å endre tekst slik at enkelte ord får stor forbokstav.

CapitalizeTextCommand

Klassen implementerer TextCommand og endrer en tekst slik at forbokstaven blir stor. Eksempel på bruk:

```
new CapitalizeTextCommand()
       .execute("text to be caplitalized")
       → Text to be capitalized
```

CapitalizeWordsTextCommand

Klassen arver fra CapitalizeTextCommand og endrer en tekst slik at alle ordene får stor forbokstav.

Eksempel på bruk:

```
new CapitalizeWordsTextCommand()
       .execute("text to be caplitalized")
       → Text To Be Capitalized
```

CapitalizeSelectionTextCommand

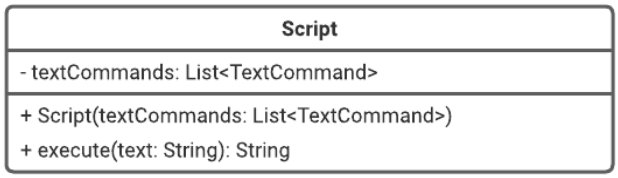
Klassen arver fra CapitalizeTextCommand og endrer en tekst slik at et utvalg får stor forbokstav. Den har et attributt for utvalget (selection). Attributtet settes gjennom konstruktøren, og brukes i execute-metoden. Husk også aksessor-metoden getSelection().

Eksempel på bruk:

```
new CapitalizeSelectionTextCommand("selection")
       .execute("text with selection and another selection")
       → text with Selection and another Selection
```

Oppgave 6: Script

Siden en kommando er et objekt kan vi lagre og utføre den på et senere tidspunkt. Det betyr at vi kan opprette flere kommandoer som vi deretter kan kjøre i en serie, slik at resultatet fra en kommando blir input til den neste. I programmering kalles dette ofte for en pipeline, men her velger vi å kalle det for et skript. Vi gjør det enkelt, som vist i følgende figur:



Figur 2: Skript-klassen

Klassen Script har en liste med kommandoer, som settes gjennom konstruktøren. Metoden execute kjører kommandoene i lista. Resultatet fra den første kommandoen blir input til den neste osv. Til slutt skal den endelige teksten returneres.

Legg merke til at Script har en execute-metode som er lik den som defineres i grensesnittet TextCommand. Det er med andre ord enkelt for Script å implementere grensesnittet, som utvider mulighetsrommet i APIet. Det er imidlertid ikke noe vi trenger å tenke på i denne øvinga.

Oppgave 7: Terminal-klient (frivillig)

I utgangspunktet skal enhetstestene være nok for å verifisere at APIet fungerer som forventet. Men hvis du ønsker kan du også lage en klient som gjør det mulig å kjøre tekstkommandoer og skript fra terminalen.