NTNU

Oblig 4 Del B

IDATG2001 Programmering 2
Institutt for datateknologi og informatikk (IDI), NTNU

Grensesnitt og unntakshåndtering

I denne oppgaven skal vi fortsatt jobbe videre med BonusMember-prosjektet fra Oblig 3. Vi skal forbedre løsningen ved å:

- Innføre **Exceptions** for å ytterligere øke kvaliteten i løsningen vår ved å håndtere unntak på en trygg måte
- Erstatte det tekstbaserte meny-grensesnittet utviklet i Oblig 2 med et grafisk brukergrensesnitt utviklet i JavaFX.
- Innføre bruk av **Logger** for å logge debug-informasjon.
- Anvende **distribuert versjonskontroll** ved hjelp av GitLab (https://gitlab.stud.idi.ntnu.no/)

Det anbefales sterkt at du aktivt benytter **checkstyle** og **SonarLint**

INNLEVERING

Innlevering foregår **individuelt** i BB ved at du lager en ZIP-fil av prosjektmappen og laster opp. Når dette er gjort, ber du om godkjenning på lab.

Oppgave 1 – Distribuert versjonskontroll

Som student ved IIR/IDI har du tilgang til NTNU sin egen **GitLab** installasjon på https://gitlab.stud.idi.ntnu.no/. Logg på med din NTNU-konto og opprett en egen privat **gruppe (Group)** for emnet **IDATx2001**. Opprett deretter et **repository/prosjekt** som du f.eks. kaller **BonusMemberApp**. Koble ditt lokale repository som du opprettet i Oblig 3 med ditt nye repository i GitLab. Utfør en **Pull** etterfulgt av en **Push** for å synkronisere ditt lokale repository med repositoriet i GitLab. Iviter **faglærer** som medlem i ditt prosjekt i GitLab (gå via «Settings->Members» i menyene til venstre i GitLab).

Videre utover i denne oppgaven kan du gjerne jobbe sammen med andre i gruppe på 2 eller 3 (unngå større grupper enn 3) mot en felles repository i GitLab, selv om innleveringen til slutt skal gjøres individuellt i BB. Dere som da jobber sammen i gruppe leverer inn samme kode i BB, men samtalen på lab vil foregå individuellt ved godkjenning.

Du/dere planlegger selv hvor ofte dere vil lage nye versjoner (**tag**), og hvilken strategi i bruk av greiner (branches) dere velger. Vi anbefaler at du/dere baserer dere på **GitFlow**-arbeidsmetoden (https://nvie.com/posts/a-successful-git-branching-model/). Bruker du/dere **SourceTree** som GUI-verktøy, finner du en egen knapp for å aktivere GtiFlow-arbeidsflyten. Bruk gjerne denne.

Oppgave 2 – Unntakshåndtering

Deloppgave a)

Innfør **Exceptions** i konstruktøren til klassen **BonusMember**. For å forhindre at et objekt av BonusMember kan opprettes med ugyldige verdier i feltene, bør samtlige parametre i konstruktøren til BonusMember valideres. Dersom et parameter har en ugyldig verdi (f.eks. parameteret **personals** er **null**), bør **InvalidArgumentException** kastes.

Lag en **Unit-test** som vist under, som tester at det nå ikke er mulig å opprette objekt av klassen BonusMember (eller strengt tatt av subklasser av BonusMember, siden BonusMember er abstrakt...) med ugyldige verdier.

```
@Test
public void testInvalidParametersInConstructor()
{
  try {
    BonusMember bm = new BasicMember(12, null, null); // Should throw exception
    fail(); // If I get to this line, the test has failed
  } catch (IllegalArgumentException e) {
    //Do not need to add anything here, since if the Exception is thrown, the
    //test is an success
  }
}
```

Du må selv bestemme i hvilken test-klasse du mener denne testmetoden hører hjemme i. Du kan lese mer om hvordan teste **abstrakte klasser** her:

https://gualtierotesta.wordpress.com/2015/01/28/tutorial-java-abstract-classes-testing/

Deloppgave b)

Hvilke andre deler av koden din bør kaste Exceptions? Analyser alle klassene og metodene dine og implementer Exceptions der du mener dette er hensiktsmessig/nødvendig. Skriv en kort vurdering/begrunnelse i Javadoc til klassen/metoden der du velger å kaste exception. Du bør da både begrunne **hvorfor** du velger å kaste exception, og hvorfor du velger å kaste et objekt av akkurat **den spesifikke Exception klassen.** Husk å lage Unit-tester som verifiserer at exception blir kastet ;-)

Prøv gjerne også å lage din egen Exception-klasse. Begrunn da (i JavaDoc) valget du gjør i forhold til om Exception bør være en **checked-** eller en **unchecked exception**.

Oppgave 2 – Grafisk brukergrensesnitt

Implementer et **grafisk brukergrensesnitt** på applikasjonen din ved bruk av **JavaFX**. Den endelige løsningen din skal oppfylle følgende krav:

1. 1) Ved oppstart skal applikasjonen din vise en liste over registrerte medlemmer
2. 2) Fra hovedvinduet skal bruker kunne gjøre følgende:
    1. a) Legge til et basic medlem
    2. b) Slette et medlem
    3. c) Kontrollere og oppgradere medlemmer som er kvalifisert
    4. d) Vise detaljer for et medlem valgt fra listen

Oppgave 2 a) og d) bør implementeres ved at et nytt **dialog-vindu** åpnes i **modal modus** for å vise samtlige detaljer om et medlem. Dialog-vinduet skal være tilpasset **klassen** til objektet som vises. M.a.o. må du lage egne dialog-vindu for hver av sub-klassene GoldMember, SilverMember og BasicMember. Alternativt til dialog-vindu, kan detaljene for et medlem vises i et eget panel, f.eks. til høyre for listen/tabellen av medlemmer.

Øvrige krav:

- For å vise listen av medlemmer **skal** du bruke **TableView**-klassen i JavaFX
- For å koble TableView til *entitets-klassene* BasicMember, SilverMember og GoldMember, er

  det **ikke tillatt** å endre noen av feltene og deres datatyper i disse entitets-klassene.

- Vi anbefaler at du **ikke** lager GUI gjennom å bruke **FXML**-metoden i denne oppgaven, da vi

  mener du lærer mer av prinsippene og klassene i JavaFX ved å «hardkode» brukergrensesnittet. Når du har kontrol på dette, kan du gjerne gå over til **FXML**.

  TIPS/ANBEFALING: IKKE start med kode direkte. Ta deg god tid til å lage skisser over hvordan du tenker/ønsker å designe ditt GUI **før** du begynner med kodingen.

Oppgave 3 – Logger

Med innføring av GUI bør vi unngå å bruke *System.out.println()* i koden vår, hverken til utskrift til bruker (absolutt ikke!!), eller for å skrive ut debug-informasjon eller logg-data. Når det er sagt, vil det lette feilsøkings-arbeidet vårt dersom vi har mulighet til å hente ut detaljer om applikasjonen vår mens den kjører. Spesielt gjelder dette når appliaksjonen er sluppet til kunde/sluttbruker og bruker melder om feil. Da er det uvurderlig å kunne hente ut log-filer for å prøve å avdekke hva som har forårsaket feilen.

Derfor har vi behov for å «skrive ut» eller lagre debug-info.
I Java finnes det et ferdig **rammeverk** for logging av data/informasjon under kjøring.

En god introduksjon til hva logging er, hvilke rammeverk og biblioteker som finnes, samt eksempler på bruk, finner du her: https://www.loggly.com/ultimate-guide/java-logging-basics/

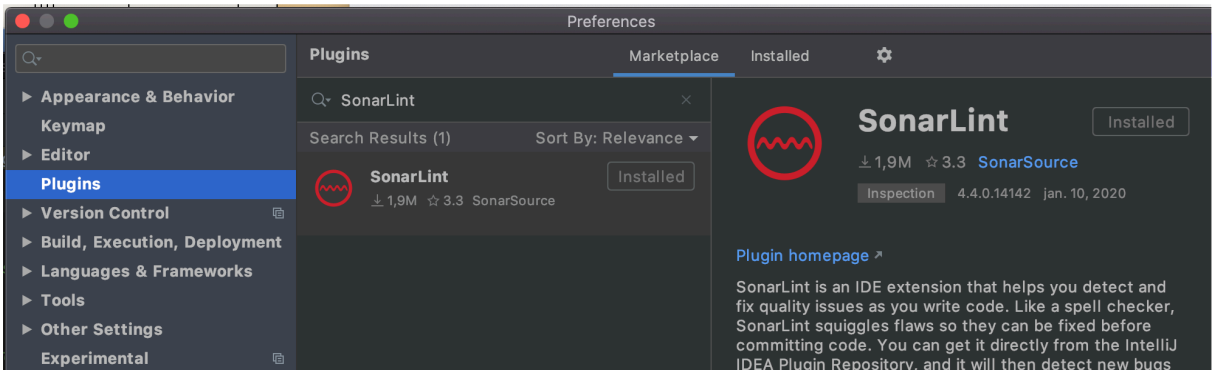
Innfør logging i ditt prosjekt, og sørg for at all bruk av System.out.println() fjernes. **TIPS**: Samtlige steder i koden din der du **fanger en exception** (catch) bør logge til loggeren.

Kodekvalitet – plugins til IntelliJ

SonarLint

SonarLint (https://www.sonarlint.org/) er en plugin til IntelliJ som sjekker koden din for sikkerhetsbrudd, «smelly code» osv og bidrar til å øke kvaliteten i koden din. Integrert i IDEen vil du få god oversikt over hvilke regler du har brutt, og for hver regel får du en detaljert beskrivelse av **hvorfor** denne regele finnes, hvorfor den er viktig, samt eksempel på hvordan du **ikke** bør kode, samt forslag til hvordan du **heller** bør kode.

Gå til Preferences/Plugins i IntelliJ, velg «Market Place» og søk etter SonarLint:

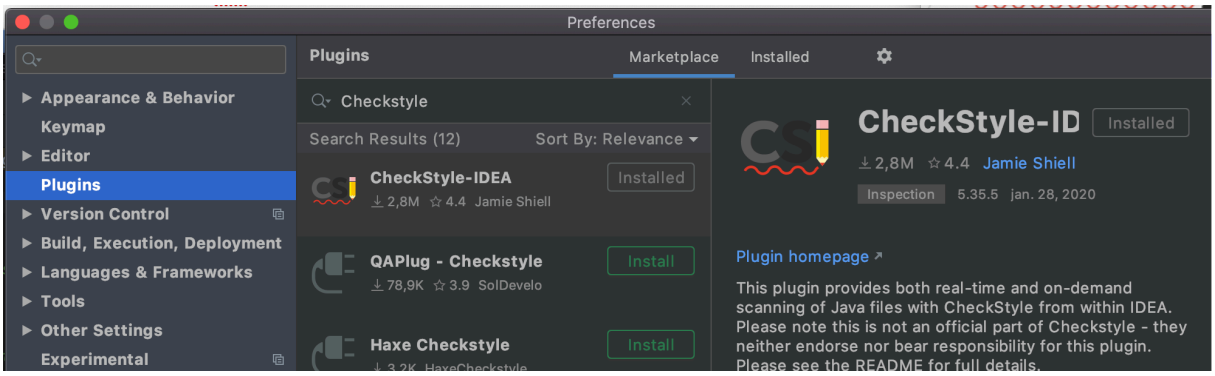


CheckStyle

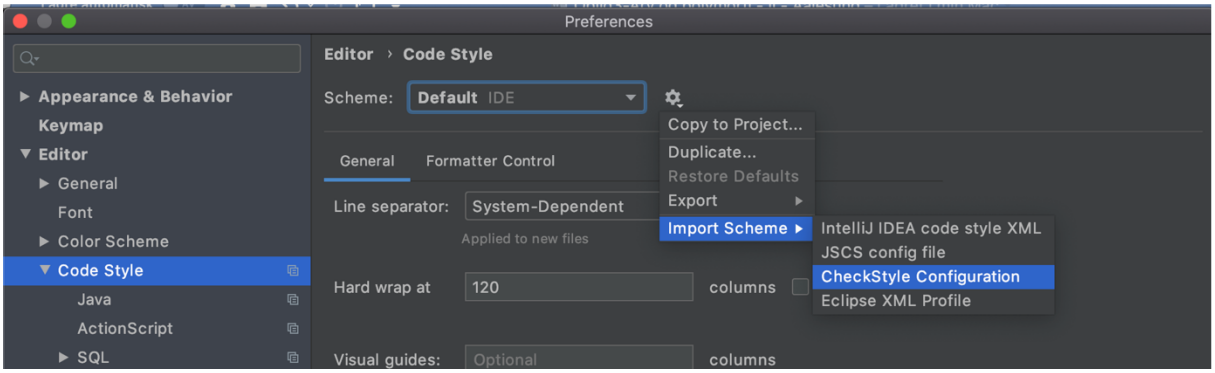
CheckStyle er et verktøy som kontrollerer at du har fulgt **en kodestandard**. Her snakker vi helt ned på detaljnivå som hvor stort et innrykk er (2 eller 4 karakterer), hvor '{'-paranteser skal stå, at det er luft mellom operander og operatorer (ikke a+b men a + b), at **samtlige** metoder som er publi har javaDoc, **og** at JavaDoc er fullstendig skrevet (med @param, @return osv).

Selve kodestandarden er konfigurert i en XML-fil, som du selv kan sette opp.

CheckStyle-plugin i IntelliJ kommer default med 2 kodestiler: Google og Sun (nå Oracle), så da kan du få testet om din kode ville ha passert kodestilsjekken til Google.



NB! Husk at IntelliJ sin editor i utgangspunktet er satt opp til å følge en gitt kodestil. Det er derfor lurt å *kalibrere* IntelliJ slik at den innebygde kodestilen i IntelliJ er i synk med f.eks. Google checkstyle.



Du må da ha tilgang til checkstyle-filen som definerer reglene. Du finner både sun og Google versjonene her:

https://github.com/checkstyle/checkstyle/tree/master/src/main/resources

I tillegg er de lagt ut i BlackBoard under Ressurser.
En detaljert beskrivelse med begrunnelser for de ulike reglene i Google sin checkstyle finner du her: https://google.github.io/styleguide/javaguide.html.

BØR LESES!

Med disse to verktøyene installert, er du meget godt rustet til å skrive god kode med høy kvalitet og som er skrevet i henhold til en etablert kodestil. En enkel måte å sikre noen poeng ekstra til eksamen.