Oppgavesettet består av 2 sider

# NORGES INFORMASJONSTEKNOLOGISKE HØGSKOLE INNLEVERING PG3300 Programvarearkitektur

Leveranseinfo:

* Skal gjøres i grupper på 2-3 personer.
* Leveres senest søndag 11. november 2012, klokka 23:55.
* Leveres via it's learning.
* Kode skal kompilere og kjøre i Visual C# 2010 Express.
* Tekstdokumentasjon skal være i docx eller pdf format.   
  (UML kan være bildeformat, inklusive screenshot.)

­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Oppgave 1: UML og refactoring av Snake

Se den vedlagte kodefila: SnakeMess.cs

Denne fila inneholder et (for de fleste) kjent spill. Koden kompilerer og kjører, men er ikke særlig lesbar, utvidbar, eller gjenbrukbar.

Denne oppgaven går ut på å refaktorere SnakeMess.cs til bedre arkitektur og kode! Faktisk så bra som dere er i stand til å skrive den. :-)

Tenk god struktur, ikke nødvendigvis hva som hadde vært praktisk for et så lite prosjekt i arbeidslivet. (M.a.o. ta det dere har lært i dette emnet helt ut, ikke lag forenklede løsninger fordi det er lite kode totalt sett).

"Ta det dere har lært i dette emnet helt ut " inkluderer: (men er ikke begrenset til)

* Bytt ut variabel-, klasse- og metodenavn.
* Splitt kode i flere klasser og metoder.
* Implementer arv(?).
* Tenk GRASP.
* Bruk en eller flere design patterns.
* Se forøvrig refactoring lista i forelesning 8.

Diskuter muligheter innad i gruppen. Noter ned stikkord for fremgangsmåte (rekkefølge på oppgaver, hva dere prioriterte), og begrunn valgene deres (f.eks. hvorfor dere valgte variant A fremfor B, der hvor dere kom opp med flere muligheter).

## Oppg. 1 a)

Sett opp UML skjema(er) for hvordan dere tenker dere den refaktorerte løsningen:

1. Implementation Class Diagram er et minimum.
2. Bruk flere av diagrammene vi har lært om hvis dere synes det virker hensiktsmessig.

Husk at ved refactoring skal programmet oppføre seg likt for brukeren før og etter! M.a.o. funksjonaliteten i programmet skal være den samme før og etter endringene.

## Oppg. 1 b)

Implementer den nye arkitekturen i Visual C#.

(Eller sørg i alle fall for at dere bygger og leverer en fungerende Visual C# solution m/kodefiler mot slutten av prosjektet, om dere jobber i et annet verktøy underveis.)

## Oppg 1 c)

Med bakgrunn i notatene, skriv et tekstvedlegg der dere gjør rede for valgene og prosessen deres. Husk at dette er en del av sensuren, så det er viktig at dere "selger" løsningen deres til sensor.

Med andre ord: I dette tekstvedlegget, sørg for å få fram hvorfor akkurat deres løsning er så bra, og vis (med ord, evt. figurer) at dere har tenkt, planlagt og resonnert dere frem til gode valg underveis for løsningen deres.

# Oppgave 2: The Cookie Bakery (multithreading)

"The Cookie Bakery" er et lite bakeri med tre faste stamkunder: De glupske herrene Fred, Ted og Greg.

The Cookie Bakery lager kun cookies (of course), og selv om produksjonen kan være nokså lav (minimum et dusin cookies om dagen, utover det velger dere selv) hindrer ikke det Fred, Ted og Greg fra å storme mot disken javnt og trutt, i håp om å være førstemann til å stå der akkurat idet en ny, fersk kake kommer ut av ovnen! Men uansett hvor samtidig herrene kaster seg mot disken, er det jo bare én som kan få tak i hver kake! Ikke sant?

## Oppg aven

Implementer "The Cookie Bakery" som en multithreaded applikasjon der Fred, Ted og Greg kniver om å få tak i cookies!

Skriv "objektorientert kode", men det er ikke nødvendig å ta det like langt som i Oppgave 1. Følgende klasser (eller tilsvarende) er nok:

* Bakery
* Cookie
* Person
* Program (m/Main)

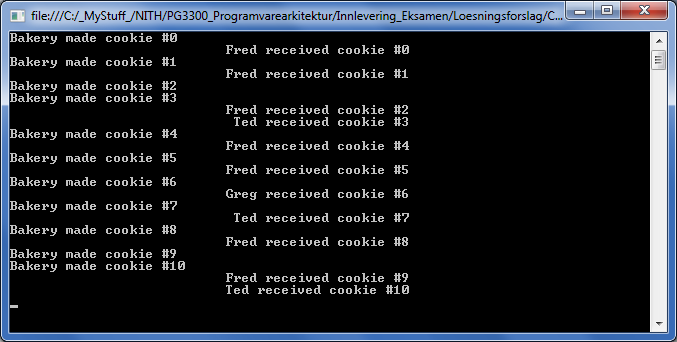
Bakeriet putter sine Cookies på disken etterhvert som de er ferdig bakt (for eksempel hvert 667. millisekund, i en List<Cookie> container eller tilsvarende). De holder åpent til de har bakt en dagskvote, og denne er solgt unna.

Bakeriet selger kakene gjennom sin metode SellCookieTo(Person customer) eller tilsvarende. Fred, Ted og Greg kaller denne metoden så ofte de kan (for eksempel en gang per 1000 millisekunder eller liknende - velg et tall som er litt høyere enn for bakingen).

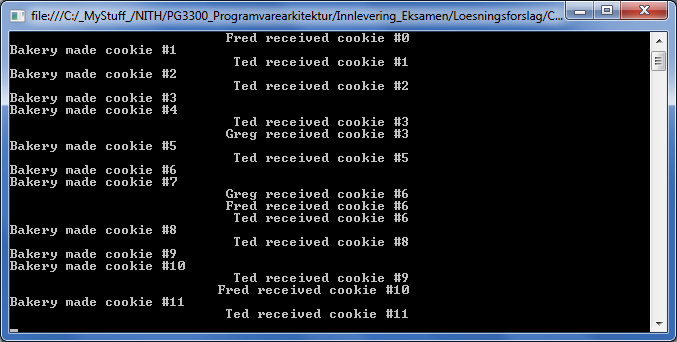
Fred, Ted og Greg prøver alle å få tak i neste kake! Implementer funksjonalitet som gjør de i stand til å prøve å plukke opp kaker i samme øyeblikk (multithreading), men sørg samtidig for at koden er trådsikker slik at bare én kan få fingrene i hver kake som kommer.

Skriv et kort dokument som begrunner valg her og, som for oppgave 1 (men antakeligvis er det færre valg, og derfor mindre å skrive her.)

Under denne gjennomkjøringen er Fred helt klart den beste cookie knabberen:



Som i skjermbildet under skal det ikke se ut! Flere herrer kan nemlig ikke spise den samme kaka! (Ref. cookie #3 og cookie #6.)



- Slutt på oppgaven -