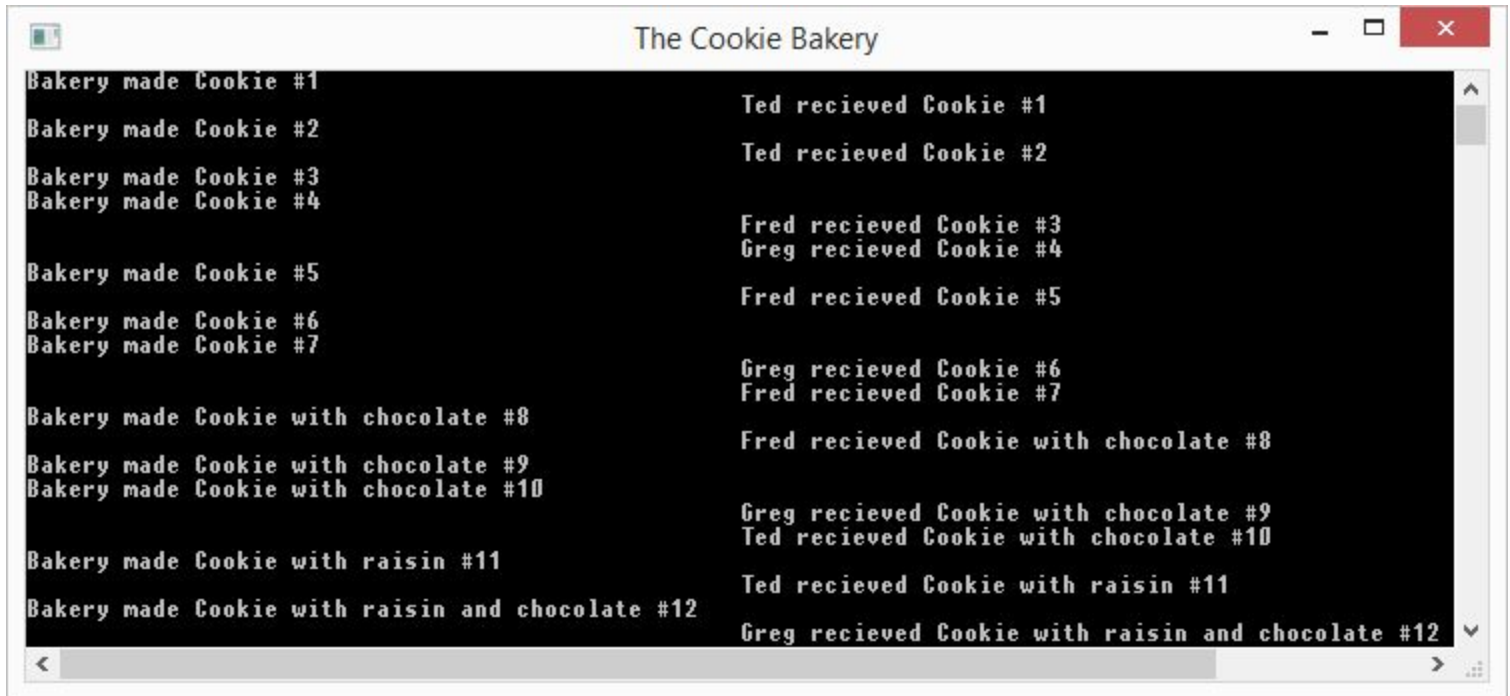


PG3300 Innlevering – Oppgave 3



```
The Cookie Bakery
Bakery made Cookie #1
Bakery made Cookie #2
Bakery made Cookie #3
Bakery made Cookie #4
Bakery made Cookie #5
Bakery made Cookie #6
Bakery made Cookie #7
Bakery made Cookie with chocolate #8
Bakery made Cookie with chocolate #9
Bakery made Cookie with chocolate #10
Bakery made Cookie with raisin #11
Bakery made Cookie with raisin and chocolate #12
Ted recieved Cookie #1
Ted recieved Cookie #2
Fred recieved Cookie #3
Greg recieved Cookie #4
Fred recieved Cookie #5
Greg recieved Cookie #6
Fred recieved Cookie #7
Fred recieved Cookie with chocolate #8
Greg recieved Cookie with chocolate #9
Ted recieved Cookie with chocolate #10
Ted recieved Cookie with raisin #11
Greg recieved Cookie with raisin and chocolate #12
```

Prosess

Som i oppgave 1, har vi også brukt Git, Teamviewer, Facebook Messenger, og slack for samarbeid på denne oppgaven. Vi har også jobbet litt på skolen etter forelesning.

Først leste vi oppgaveteksten grundig for å forstå hva vi skulle gjøre og skrev noen notater på funksjonaliteten.

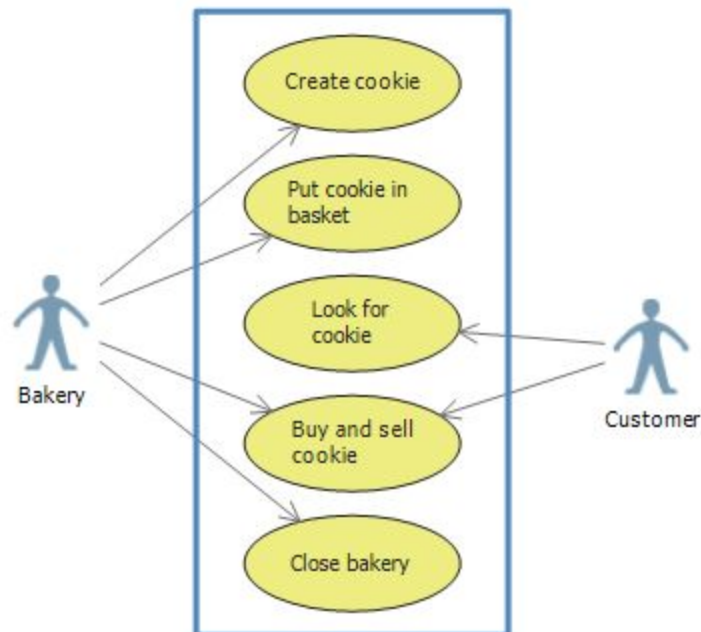
For å få god oversikt og planlegge hvordan programmet skal fungere, laget vi et UML use case diagram

Da vi anså use case diagrammet som ferdig, opprettet vi klasser og metoder for programmet.

Vi gjorde research på det vi var usikker på underveis i kodingen. Blant annet undersøkte vi mer om design patterns og valgte legge til Decorator for å legge til tilbehør på cookies.

Etter programmet var fullført, genererte vi UML klassediagram ut fra koden i Visual Studio 2015.

Use Case Diagram



Use case diagrammet representerer hvordan vi ser for oss ferdig produkt.

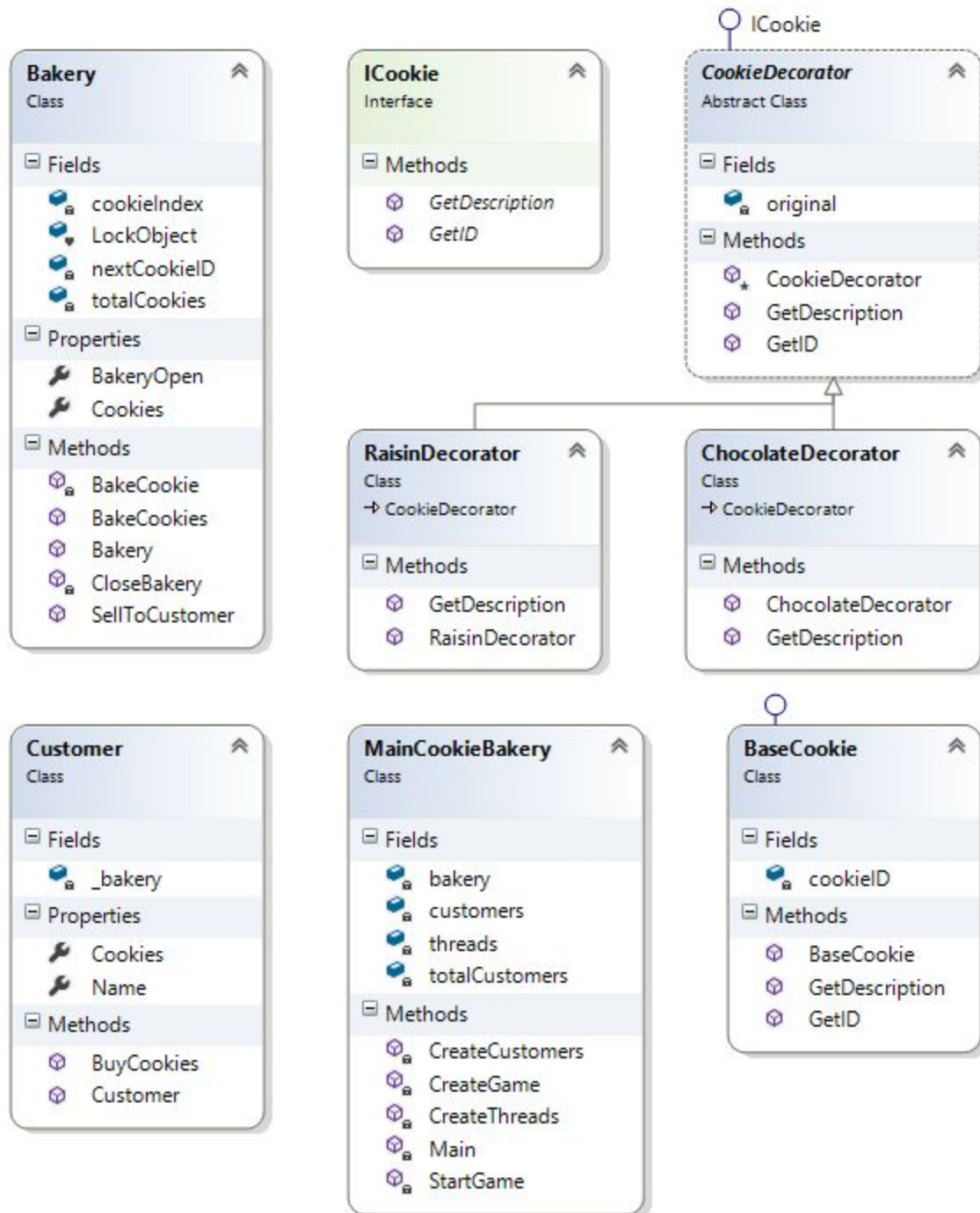
Bakeriet oppretter kaker og legger dem i en kurv.

Kunden ser etter kaker og prøver å kjøpe kake hvis en er tilgjengelig.

Bakeriet selger kake til kunde når kunden kjøper den.

Bakeriet stenger når alle kakene er solgt.

Implementation Class Diagram



Implementasjon klassesdiagram generert fra kode i Visual Studio 2015 Enterprise.

Valg for kode

Som i SnakeMess oppgaven har vi også her fulgt C# konvensjoner for navngiving av variabler, metoder og klasser.

Design pattern

For å videreutvikle oppgaven valgte vi å bruke design patternet Decorator for å kunne legge til tilbehør på cookies i runtime. Det gjorde det enkelt å legge til sjokolade, rosiner, eller begge deler på cookies. Bakeriet lager cookie med tilfeldig fyll, enten uten fyll, med sjokolade, med rosiner, eller sjokolade og rosiner. Annet fyll kan enkelt legges til ved å opprette flere decorator klasser som arver fra CookieDecorator og legger til tilbehøret i metoden GetDescription.

Decorator løste vi ved å lage et interface ICookie med en metode med retur string: GetDescription. Så laget vi en CookieDecorator klasse som arver av interfacet, som har en ICookie variabel for original cookie og implementasjon av GetDescription. Klassen BaseCookie arver fra ICookie og er cookie uten fyll og denne må opprettes før fyll kan legges på, denne returnerer kun "cookie" på GetDescription. Deretter laget vi en klasse for ChocolateDecorator og RaisinDecorator, som arver fra CookieDecorator og i konstruktøren tar den i mot original cookie (opprettet med BaseCookie). Disse legger til "with chocolate" eller "with raisin" på GetDescription.

Multi Threading

Vi tok i bruk Multi Threading i oppgaven der en thread skulle representere en handling som en kunde skulle utføre. I vårt eksempel lagde vi tre Customer-objekter og tre tråder. Trådene skulle deretter gjøre en handling samtidig. Problemet med tråder som kjører en metode samtidig er at det kan oppstå race conditions, altså at flere tråder endrer/lagrer verdi i en variabel noe som kan påvirke det endelige resultatet. For å fikse dette problemet, så tok vi i bruk C# sin lock-property. Vi tok deretter og puttet locks rundt Bakery sin SellToCustomer() metode. Dette sørget for tråd-sikkerhet slik at race-conditions ikke skulle skje. I tillegg sikres det at kaken fortsatt finnes i cookies arrayet før kunden mottar kaken. Det sikrer at bare én kunde kan få tak i hver kake som kommer. Det skjer race condition i kunden sin buycookie ved at to/flere kunder leter samtidig og prøver ta samme kake, men kun den første vil motta kaken siden selltocustomer har lock og if setning.

Klasser

Klassene i TheCookieBakery har tatt i bruk elementer fra GRASP-konseptet. De oppfyller high cohesion ved at metodene og objektene kun gjør relaterte oppgaver. For eksempel, så har Customer klassen bare metoder som er relevant til kunden. Dette har tillatt oss å ta i bruk “low coupling” noe som sørger for at det ikke blir mange koplinger mot klassene. I tillegg til å ta i bruk high cohesion og low coupling, så har vi også tildelt MainCookieBakery “Creator” rollen. Det vil si at den har hovedansvaret for opprettingen av objekter som tar seg av programmets funksjoner.

Hver klasse legges i sin egen .cs fil. Her en beskrivelse klassene vi har opprettet:

- Bakery: Har ansvaret for opprettingen av cookies. Oppretter en array av typen ICookies og fyller den med cookie elementer som har tilfeldig tilbehør og legger dem deretter ut for salg. Selger deretter “kurven” med cookies til Customers helt til det er slutt på kaker.
- BaseCookie: Arver fra interfacen “ICookie” og returnerer en vanlig kake ved bruk av GetDescription().
- CookieDecorator: Er en abstrakt hjelpeklasse som blir brukt av andre decorator-klasser til å dekorere cookies. Dekorator klassene arver fra den abstrakte klassen og bruker dens metoder til å endre på kaken slik de måtte ønske.
- ChocolateDecorator: Arver fra den abstrakte klassen “CookieDecorator” og bruker “override” til å overskrive “CookieDecorator” sin metode slik at en ny verdi blir returnert. Dette er mulig fordi CookieDecorator sin GetDescription() er virtual.
- RaisinDecorator: Arver også fra “CookieDecorator” og gjør det samme som “ChocolateDecorator” bare at en annen verdi blir returnert.
- Customer: Representerer en kunde som er i bakeriet og skal kjøpe cookies. Konstruktøren har navnet til kunden og instansen av “Bakery” klassen som parameter. Har metoden BuyCookies() som senere blir kalt på av de ulike trådene i “Program” klassen. Hver tråd representer en kunde og kaller på BuyCookies() metoden.
- MainCookieBakery: Inneholder selve main-metoden i programmet, eksekvereringen av selve programmet skjer her. Inneholder metoder (CreateGame, CreateCustomers, CreateThreads) som lager Arrays av typen Customer og Thread. Arrayene av typen Customer og Threads blir deretter instansiert og fylt inn med elementer. Threadene blir startet og kjører hver av elementene til “customers” sin BuyCookies metode.

Metoder

- `BakeCookies(Bakery.cs)`: Bruker en for-løkke til å bake cookies og skrive dem ut i konsollen hver 667.millisecond.
- `SellToCustomer(Bakery.cs)`: Selger cookie til en kunde(tar imot en parameter verdi av typen `Customer`). Etter en cookie har blitt solgt til en kunde så blir elementet fjernet fra arrayen, det skrives i konsollen at kunden kjøpte kaken med beskrivelse, og cookien blir returnert.
- `BakeCookie(Bakery.cs)`: Baker en cookie. Typen cookie er basert på en "Random Number Generator" som velger mellom tall fra 0-4. Hver av tallene representerer en type cookie. Kaken blir deretter bakt og returnert.
- `CloseBakery(Bakery.cs)`: Stenger bakeriet. Endrer den boolske variabelen "bakeryOpen" fra true til false
- `GetID()`: Returnerer en int for hvilket nr kaken har. Brukes av bakery og customer for å legge til id på kake og sjekke hva id til kake er.
- `GetDescription()`: Returnerer en String av hva slags type cookie det er. Finnes i alle decorator klassene (`CookieDecorator`, `BaseCookie`, `ChocolateDecorator`, `RaisinDecorator`).
- `BuyCookies(Customer.cs)`: Ser etter og kjøper cookies hver 1000.millisecond så lenge `bakeryOpen` variabelen er true. Sjekker også om det finnes cookies i kurven og kjøper cookie hvis antallet cookies i en kurv er mer enn 0. Metoden bruker deretter `SellToCustomer` sin returverdi til å legge verdiene i en liste.
- `CreateGame(MainCookieBakery.cs)`: Setter opp programmet ved å kalle på metoden `CreateCustomers()` og `CreateThreads()`. Tittelen og størrelsen på konsollen blir også satt.
- `CreateCustomers(MainCookieBakery.cs)`: Kunder blir opprettet ved at Array av typen `Customer` blir instansiert og fylt inn med elementer.
- `CreateThreads(MainCookieBakery.cs)`: Array av typen `Thread` blir opprettet med antall elementer lik antall `Customer`-objekter. Hver av trådene får dermed en `ThreadStart` med `customers` sin `BuyCookies` metode som parameter.
- `StartGame()`: Starter alle trådene. Kaller deretter opp bakery sin `BakeCookie()` metode.

Variabler

C# gir deg også mulighet til å deklarere variabler som “var”, men vi velger å bruke den originale variabeltypen for å gjøre det mer lettlest, unntatt på instansiering hvor det kan være mer lettlest å bruke var, som “var bakery = new Bakery ();”

Kilder

- Vi har brukt msdn for å se etter tilgjengelige features i C#: <https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx>
- C# Coding standarder: <http://www.dofactory.com/reference/csharp-coding-standards>