Laboration 2

Grafik, Händelser & Trådar

Sektionen för IT
Högskolan i Borås
501 90 Borås

Patrick Gabrielsson
patrick.gabrielsson@hb.se
Rum: L436

Innehåll

1 Syfte

Laborationen behandlar skapandet av grafiska gränssnitt, händelsehantering, trådar och objektorientering.

2 Uppgift

Laborationen går ut på att konstruera ett objektorienterat Java-program som spelar ”Othello” (https://en.wikipedia.org/wiki/Reversi). Programmet skall fungera på två olika sätt: antingen skall en ensam spelare kunna spela mot datorn eller så skall två spelare kunna spela mot varandra.

Othello är ett tvåmannaspel där den ena spelaren har vitt och den andre svart. De brickor som används är vita på ena sidan och svarta på andra. Spelplanen är 8×8 rutor och utgångsläget är att två svarta och två vita brickor ligger ”diagonalt” på de fyra mittenrutorna. Vit har första draget.

Ett drag innebär att placera en bricka så att minst en av motståndarens brickor ”fångas” av den just placerade brickan dvs. ligger mellan denna och någon annan av den aktive spelarens brickor. Samtliga brickor som på detta sätt fångas vänds. Detta gäller även hela rader av brickor, men naturligtvis bara de som fångas av den brickan som placerades i draget. När draget är genomfört är det nästa spelares tur.

Om en spelare inte kan genomföra ett drag går turen över till motståndaren. Den spelare som har flest brickor i sin färg då brädet är fullt (eller då igen av spelarna kan genomföra ett drag) vinner.

Programmet skall åtminstone innehålla följande klasser:

Othello, Player, HumanPlayer, ComputerPlayer, GameManager, GameGrid, InvalidMoveException, SetUpGameDialog, GameFrame, WinnerDialog och DrawnDialog.

Det är naturligtvis möjligt att skapa fler klasser för att få en bra lösning. Notera att samtliga klasser skall redovisas i ett överskådligt klassdiagram där klassernas relationer till varandra tydligt framgår. Samtliga klasser skall även beskrivas i detalj med hjälp av UML-notation där attribut, metoder, synligheter och typer framgår. De metoder som inte kan anses vara triviala skall kommenteras i koden.

3 Konstruktion av spelare

En spelare kan utformas på olika sätt. Här skall vi tillämpa en ”riktig” objektorienterad teknik med arv och dynamisk bindning.

En spelare skall kunna tala om vilket hans nästa drag är. Man skall om så behövs kunna meddela att draget var felaktigt och måste göras om. Dessutom skall det gå att meddela en spelare att han vunnit.

Eftersom spelet skall utformas så att man antingen kan spela mot datorn eller mot någon annan människa finns det två huvudtyper av spelare: människor och datorer. De två typerna har samma operationer men operationerna utförs på olika sätt. För en mänsklig spelare läser man t.ex. in nästa drag från tangentbordet eller musen (krav att det är möjligt) medan man för en dator använder en algoritm för att beräkna nästa drag. Utgå därför från en abstrakt basklass **Player** och skapa sedan de två subklasserna **HumanPlayer** och **ComputerPlayer**. Antagligen blir det nödvändigt att använda trådar då en spelare beräknar eller väntar på nästa drag.

3.1 Klass: Player

Klassen Player är basklass för HumanPlayer och ComputerPlayer. Klassen innehåller attributen namn och markör-id samt en abstrakt metod för att begära beräkning av nästa drag. Namnet sätts i dialogrutan SetUpGameDialog och används i WinnerDialog. Markör-id anger om spelaren har vita eller svarta brickor.

3.2 Klass: HumanPlayer

Då nästa drag begärs från GameManager väntar man tills användaren utfört draget.

3.3 Klass: ComputerPlayer

Denna klass beräknar nästa drag utifrån den information som finns i klassen GameGrid.

Som du säkert inser är den besvärligaste metoden att konstruera en intelligent datorspelare. För att inte fastna på detta nu, är det tillåtet att låta datorn vara hur dum som helst, dvs. den genererar helt enkelt ett slumpmässigt tillåtet drag. Vi kommer att kika på hur man skapar smartare datorspelare i kursen *intelligenta och lärande system*.

4 Konstruktion av GameManager, GameGrid och InvalidMoveException

4.1 Klass: GameManager

GameManager är den som administrerar pågående parti. *GameManager*ns uppgift är enkel. Den skall gång på gång, omväxlande, be de två spelarna att ange sitt nästa drag. Den skall också tala om för spelarna om de vunnit eller gjort ett felaktigt drag. Eftersom *GameManager*n inte skall veta vilken typ av spelare den har att göra med skall dynamisk bindning användas vid anrop av spelarnas operationer.

4.2 Klass: GameGrid

GameGrid är den klass som innehåller partiets tillstånd (exempelvis en matris). När en spelare utför ett drag kommer detta att registreras i *GameGrid*en. Om en spelare utför ett otillåtet drag, så skall *GameGrid*en kasta ett undantag med namnet *InvalidMoveException*. Då *GameBoard* ritar om spelplanen hämtas informationen om partiets aktuella ställning från denna klass.

4.3 Klass: InvalidMoveException

Detta undantag kastas då någon försöker göra ett otillåtet drag. Detta gäller både mänskliga spelare och datorspelare. Programmet skall inte låsa sig utan låta spelaren göra ett korrekt drag istället.

Er nästa uppgift blir nu att konstruera och testa dessa klasser. Ett spel avslutas när hela spelplanen är fylld med brickor eller när ingen av spelarna kan utföra ett drag. I båda fallen är det spelaren med mest brickor i sin egen färg som har vunnit.

5 Konstruktion av gränssnittsobjekt

En av huvudprinciperna vid objektorienterad design är att man skall bygga upp sitt program i form av separata, väl avgränsade, klasser. Detta underlättar förändringar i programmet. Det är därför viktigt att bibehålla inkapsling och tänka noggrant på att separera logik från det grafiska gränssnittet. De klasser som innehåller logiken skall inte känna till eller på något sätt vara beroende av gränssnittsobjekt. Detta medför att ett byte av det grafiska gränssnittet inte påverkar några av de övriga delarna i programmet.

Tänk på att GUI objekt skall modifieras på JavaFX-applikationens tråd, dvs om ni har skapat en egen tråd i erat program som ni utför något jobb på och vill presentera resultatet i ett GUI objekt, så måste ni först växla över till JavaFX-applikationens tråd innan ni ändrar på GUI objektet. Exempelvis, så anropas nedanstående kod från en ”worker” tråd för att skriva text till ett *TextArea* objekt via JavaFX-applikationens tråd:

```
Platform.runLater(() -> ta.appendText("Text Meddelande"));
```

Tyngre beräkningar, t.ex. då *GameManager* ber *Player* (framförallt *ComputerPlayer*) att ange sitt nästa drag, skall köras i en egen tråd så att inte det grafiska gränssnittet ”fryser” medans beräkningen utförs.

5.1 Klass: SetUpGameDialog

I denna dialogruta namnges de två spelarna samt om de skall vara *HumanPlayer* eller *ComputerPlayer*. Då dialogrutan stängs skapar (instantierar) man de två spelarna och skickar referenser till *GameManager*. Det exakta utseendet på dialogrutan får väljas fritt. Det är även tillåtet att använda två dialogrutor; ett för att hämta information om spelare 1 samt ett för att hämta information om spelare 2.

5.2 Klass: GameFrame

GameFrame är en instans av en *javafx.stage.Stage* och finns uppe så länge programmet körs. Den innehåller, bland annat, en instans av *GameBoard* samt knapparna ”Nytt parti” och ”Avsluta”.

5.3 Klass: GameBoard

GameBoard kan t.ex. vara av typen *javafx.scene.layout.Pane* och utgör den grafiska representationen av spelplanen och används för att presentera den aktuella ställningen. *GameBoard* fångar dessutom upp användarens (*HumanPlayer*) val av ruta då denne gör ett drag.

Observera att trots att gränssnittsobjekten ritar spelplanen och brickorna i den, så är det inte deras uppgift att hålla reda på den aktuella ställningen. När spelplanen skall ritas om hämtas all information från *GameGrid*en.

5.4 Klass: WinnerDialog

Denna klassen utgör en dialogruta som informerar om en spelare har vunnit. Spelaren skall tilldelas med namn. Dialogrutan innehåller också en ”Ok” knapp som stänger dialogrutan.

5.5 Klass: DrawnDialog

Denna klassen utgör en liknande dialogruta som informerar om att spelet blev oavgjort, dvs. brädet är fullt eller inga av spelarna kan utföra ett drag.

6 Konstruktion av Spelets Huvudklass

6.1 Klass: Othello

Othello är spelets huvudklass och (eftersom spelet är en JaxaFX applikation) även från *javafx.application.Application*.

Nu kan hela programmet testköras!

7 Tips

Tänk igenom lösningen innan ni börjar koda. Visa gärna upp det klassdiagram ni har skapat samt redogör för hur ni har utformat klasserna och de viktiga metoderna för handledaren.

Någonstans måste en eller flera trådar användas för att programmet skall fungera bra.

8 Kravspecifikation

Strukturen ovan skall följas. Var noga med att låta objekten vara ”så inkapslade som möjligt”. Utnyttja de verktyg som objektorientering och Java erbjuder.

Klassdiagram (UML) och dokumentation av klasser (Javadoc) samt viktiga metoder (kommentarer) skall lämnas in tillsammans med källkod.

9 Övrig information

Arbeta i grupper om 2-3 studenter (samma grupp som vid laboration 1) .

9.1 Handledning

Handledning sker vid tre tillfällen enligt de tider som finns i schemat. Tänk på att komma till handledningen med väl förberedda frågor. Bokningslistor för handledningen finns i PingPong under Aktivitet/Innehåll/Handledning/Laboration 2. Där hittas Doodle länkar för varje handledningstillfälle (OBS! En Doodle länk för bokning av ett visst handledningstillfälle blir först synlig i PingPong dagen efter det förra handledningstillfället).

För varje handledningstillfälle (Handledning 1, 2 samt 3), boka endast **ett** handledningspass per grupp. Med reservation för ändringar (kolla alltid schemat), så gäller för närvarande nedanstående datum (och rum) för varje handledningstillfälle:

- Handledning 1 sker måndagen den 12 oktober i rum L433 13:00-15:30.
- Handledning 2 sker fredagen den 16 oktober i rum L433 09:00-11:30.
- Handledning 3 sker onsdagen den 21 oktober i rum L433 13:00-15:30.

9.2 Redovisning

Laborationen redovisas via PingPong, under Aktivitet/Inlämning/Laboration 2, där samtliga filer lämnas in som **en** arkivfil (zip-fil eller rar-fil). Följande filer skall finnas med i arkivfilen:

- Designdokumentation (UML klassdiagram)
- Javadoc (klassspecifikation)
- Källkod (beskriv relevanta metoder m.h.a. kommentarer i koden)
- En PDF som listar vilka metoder varje gruppmedlem har skrivit.

Notera att det är varje enskild person ansvar att denna bidrar med relevant kod. Gruppen måste dock ge alla möjlighet att göra relevanta bidrag. Medlemmar som inte bidrar till laborationens lösning kommer underkännas även om gruppen blir godkänd. Uppstår det konflikter i gruppen skall detta tas upp med ansvarig lärare direkt och inte efter att laborationen lämnats in.

Inlämning och examination av laborationen sker vid två tillfällen enligt schemat. Båda inlämningstillfällen får utnyttjas. Om ni fortfarande inte har blivit godkända på laborationen vid andra inlämningstillfället, så ges en ny (modifierad) version av laborationen ut som skall lämnas in senast efter sommaren 2016. OBS! Ingen handledning ges för den nya (modifierde) versionen av laborationen. Nedanstående datum gäller för varje examinationstillfälle (OBS! En länk för ett inlämningstillfälle blir först synlig i PingPong dagen efter det förra inlämningstillfället):

- Deadline för inlämningstillfälle 1 är fredagen den 23 oktober 09:00.
- Deadline för inlämningstillfälle 2 är måndagen den 18 januari 09:00.
- Deadline för inlämningstillfälle 3 (nya versionen) är måndagen den 15 augusti 09:00.

9.3 Betyg

Endast betygen Underkänd eller Godkänd förekommer på laborationen.