Semesteroppgave: filbehandling og unntak

DATS1600 oppgavesett 5. Obligatorisk oppgave inkludert.

I dette oppgavesettet skal dere implementere filbehandling for GoL, slik at dere kan laste inn mønstre fra Internett. Filbehandling og håndtering av 'streams' (strømmer) generelt er en vanlig kilde for bugs i programmer. Dere skal bruke Java exceptions for å behandle feil relatert til filbehandling i deres program.

Unntakshåndtering presenteres i forelesning i uke 6, filbehandling presenteres i uke 7, og tekstmanipulering presenteres i uke 9. For studenter med treg progresjon, tilbys repetisjons-presentasjoner i tutorialtimene i ukene 13, 14, og 16. Avansert bruk av IDE (som bruk av debuggingsverktøy) presenteres i tutoriatime i uke 9.

Oppsett

Hittil har vi arbeidet med relativt små spillebrett der startverdiene til cellene er 'hard-kodet' i Javakoden. Vi skal nå generalisere spillebrettet vårt slik at vi ikke trenger å gjøre slik hard-koding.

Deklarer spillebrettet uten å initialisere brettet. I tillegg, deklarer variabler som definerer størrelsen til spillebrettet. For eksempel, hvis deklarasjonen på spillebrettet var:

```
private byte[][] board = {
                          { 1, 0, 0, 1 },
                          { 0, 1, 1, 0 },
                          { 0, 1, 1, 0 },
                          { 1, 0, 0, 1 }
                         };
```

skal dette nå endres til (og kanskje øke størrelsen litt):

```
private final int WIDTH = 4, HEIGHT = 4;
private byte[][] board;
```

Merk at dere har nå gjort endringer på en basisklasse (modellklasse) i koden deres. Denne endringen kan ha produsert bugs andre steder i koden. Test programmet deres, inkludert utførelse av JUnit testing.

Intro til Java exceptions

Klasser og funksjoner kan 'kaste' exceptions (unntak). Kjør følgende kode i et enkelt javaprogram:

```
int[] array = new int[10];

int counter = 0;
while (true) {
        array[counter] = ++counter;
}
```

Denne koden burde skrive ut følgende:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at eksempler.Main.main(Main.java:16)
```

Sett dere inn i try-catch blokken i Java. Gjør endringer i koden over, der du avslutter løkken ved hjelp av en try-catch blokk. Diskuter ulempene med en slik terminering av løkker (hint: spagettikode og ytelse).

Det er to typer Java Exceptions: *checked* og *unchecked*. En ArrayIndexOutOfBoundsException er et eksempel på unchecked exception, der det er valgfritt om en try-catch konstruksjon brukes for å fange unntaket. På den andre siden, i checked exceptions må en try-catch konstruksjon brukes (eller kaste unntaket videre). I dette oppgavesettet vil dere utelukkende bruke checked exceptions.

Dere skal implementere en metode som laster inn et spillebrett fra en 'stream' (f.eks. en fil på harddisken eller en 'fil' fra Internett). Diskuter hvordan denne funksjonaliteten skal inkluderes i programstrukturen. Noen alternativer: en innlastingsmetode i klassen/interfacet relatert til GoL, en (statisk) metode i en hjelpeklasse, eller en egen klasse for håndtering av GoL mønstre eller filbehandling.

Forslag på signatur til relaterte metoder:

```
public void readGameBoard(Reader r) throws IOException {…}

public void readGameBoardFromDisk(File file) throws IOException {…}
```

Java sitt standard bibliotek tilbyr to metoder for aksessering av data: Streams og Readers. En Stream benyttes for aksessering av binær data, som f.eks. et jpg bilde, og en Reader benyttes for aksessering av character data, som en tekstfil. Filer relatert til GoL mønstre er tekstfiler.

Se på dokumentasjonen for den abstrakte klassen java.io.Reader. Forstå hvorfor denne klassen er abstrakt og hva som må implementeres for å gjøre en arvet Reader klasse konkret.

Metoden readGameBoardFromDisk(…) er en hjelpemetode for at brukeren skal slippe å konvertere en sti til en gitt fil på harddisken til et objekt av typen Reader. Forslag til implementasjon:

```
public void readGameBoardFromDisk(File file) throws IOException {
      readGameBoard(new FileReader(file));
}
```

Bruk denne metoden til å la brukeren laste inn filer fra GUIet, via filbehandleren til operativsystemet (JavaFX sin FileChooser kan brukes her). Fang (catch) IOException'et som kastes av readGameBoardFromDisk (og konstruktøren til FileReader) og vis en dialogboks med forklarende feilmelding. Test dette (for eksempel, som å lese en fil som ikke finnes) og vær sikre på at opprettelse av Reader objektet er korrekt (når en fil faktisk finnes) før dere starter med implementasjonen av readGameBoard metoden.

Parsing og tolkning av GoL mønstre

GoL mønstre er formatert i ulike *filformater*. Slik formatering av data er en vanlig måte å lagre data relatert til en spesifikk applikasjon. Les om de ulike filformatene definert for GoL. Eksempler på relativt enkle formater er PlainText og Life 1.06. Formatet RLE har imidlertid mest støtte, og det anbefales at dere implementerer en parser (tolker) for dette formatet hvis dere er komfortable med programmering. Hvis dere ikke er komfortable, anbefales Life 1.06 formatet siden dere vil ha bedre tid til å fokusere på andre oppgaver.

Diskuter hvordan nye innlastinger skal håndteres i programmet. Det er to naturlige valg her:

1. Erstatt nåværende brett med det nye brettet. Dvs., 'tøm' spillebrettet og deretter last inn det nye brettet.
2. Inkluder det nye mønsteret med nåværende brett. Dette er et litt mer avansert alternativ, men gjør det mulig å håndtere flere mønstre samtidig. En problemstilling som må løses er hva som skal gjøres hvis det er overlapp mellom nåværende mønster og det nye mønsteret. En mulig løsning er å la brukeren flytte på det nye mønsteret (se utvidelsesoppgave).

Opprett en klasse som utvider (arver fra) Exception klassen og kall den PatternFormatException. Denne klassen skal brukes til å fange unntak relatert til formatet som brukes til å laste inn nye mønstre (f.eks. feil format etc.). Vis en dialogboks som beskriver denne feilen. Dvs. at dere vil bruke ulike klasser av typen Exception til å fange ulike typer feil (IOException for feil med filinnlasting og fillesing, og PatternFormatException for feil formatering av tekst inne i filen).

Implementering av parsere/tolkere kan være litt kronglete; dette er nok den obligatoriske oppgaven som har høyest vanskelighetsgrad i faget (hvis dere velger RLE). Det kan være behjelpelig å planlegge utførelsen av denne oppgaven, som f.eks. å kladde en beskrivelse av en algoritme som går igjennom filen, 'char-for-char'. I tillegg, burde dere dele opp oppgaven i mindre deler, der dere tester hver del før dere går til neste steg. Et eksempel er å først implementere gjennomgang av header delen, og deretter implementere selve datainnlesingen. Husk at det er spesielt viktig med naturlig formatering av kode, slik at debugging (identifisering av feil) blir enklere.

Den primære fremgangsmåten for å identifisere feil i koden er å bruke en **debugger**, som tilbys av IDEen din. Sett dere inn i hvordan 'break points' fungerer, og hvordan dere kan bruke debuggeren til å identifisere programmeringsfeil.

Vi har nå to verktøy som brukes for feilhåndtering: JUnit, som brukes til å finne ut om det er noen feil i koden og debuggingsverktøy, som brukes til å finne ut hvor i koden feilen ligger.

Opprett en JUnit klasse/metode som tester readGameBoard metoden (eventuelt via readGameBoard-FromDisk). Nå som spillebrettet kan være relativt stort (definert av variablene width og height), kan det være lurt å bare teste celler som vi vet kan være sanne (dette kalles for bounding box). Se Vedlegg 1 for en metode som kan brukes til å hente ut beskrivelse av slike celler. Finn en gruppe med små mønstre på Internett, der det er mest mulig variasjon mellom mønstrene, slik at dere tester flest mulig scenarioer.

Merk at hvis mønsteret er større enn spillebrettet, skal en feil kastes og håndteres i GUIet. Dette er fordi brettet er definert som en statisk datastruktur, der dens størrelse ikke kan endres etter opprettelse. Dynamiske brett skal implementeres i en senere oppgave.

Etter at testmiljøet er satt opp, kan dere starte med å implementere metoden. Lykke til!

Lesing fra web

Dere vil nå se fordelen med at Reader er en abstrakt klasse.

Legg til en metode som leser et mønster direkte fra Internett, via en URL. Signaturen av en slik metode kan være:

```
public void readGameBoardFromURL(String url) throws IOException, PatternFormatException {…}
```

Husk, en stream (strøm) er en sekvens av char, som kan komme fra ulike kilder (harddisk, Internett, etc.). Implementasjonen av metoden over kan se slik ut:

```
public void readGameBoardFromURL(String url) throws IOException,
            PatternFormatException
{
      URL destination = new URL(url);
      URLConnection conn = destination.openConnection();
      readGameBoard(new InputStreamReader(conn.getInputStream()));
}
```

Vi har nå to bruksområder relatert til readGameBoard metoden (som tar den abstrakte klassen Reader som parameter). Forstå hvorfor denne metoden, via polymorfisme, kan brukes til å både hente et spillebrett fra en fil på datamaskinen og fra en URL.

Legg til GUI mekanismer (f.eks. en dialogboks som spør etter URL) for å la brukeren få laste inn et mønster fra Internett (URL). Husk å gi feilmelding hvis noe gikk galt (URL eksisterer ikke etc.).

Bruk av metadata (valgfri oppgave)

Header delen av filen kan brukes til å lagre informasjon som kan være aktuelt for brukeren, som tidspunkt for opprettelse, navn på oppretteren, beskrivelse av mønster etc. Slik data blir ofte referert til som *metadata*.

Opprett et format for metadata. Inkluder felter dere tror kan være interessant for brukeren. Bruk et forhåndsdefinert tegn for å separere ulike felter. Eksempel på format:

#C@metadata:author:time_of_creation:last_edited:message_from_author:description_of_pattern

Legg til metadata manuelt i noen filer dere bruker for testing. Opprett en JUnit test for testing av innlesing av slik metadata. Husk: ukorrekt format for metadata betyr ikke at formatet er ugyldig; mønsteret burde fortsatt leses inn selv om metadata er formatert feil. Hvis dette er tilfellet, burde metadataen ignoreres.

Bruk Javas støtte fra String (som substring) og eventuelt regulære uttrykk for å hente ut metadata. Eventuelt opprett nye klasser/felter for dette. Diskuter hvordan metadata burde inkluderes i programstrukturen.

Legg til elementer i GUIet (som for eksempel en statusbar) som viser eventuelle metadata.

Merk at lagring av metadata, samt lagring av mønstre, er en del av en utvidelsesoppgave (manipuleringsmuligheter av GoL mønstre).

Utvidelsesoppgave

NB: det anbefales å implementere kjernen til programmet før dere starter med utvidelsesoppgaver.

Et nytt mønster må plasseres 'et-eller-annet' sted på spillebrettet. Et naturlig valg er å plassere mønsteret på midten av brettet. Et slik valg kan, imidlertid, motstride brukeren sine ønsker.

Gi brukeren muligheten til å flytte på mønsteret som har blitt lastet inn, for eksempel via pilene på tastaturet. Bruk lineær algebra til å utføre transformasjoner som forflytting og rotering. Dere kan også prøve dere på skalering, men dette er mer komplisert (der vi antar at det skalerte mønsteret følger samme evolusjon som det originale).

I tillegg, burde brukeren kunne flytte på brettet. Dette er spesielt ønskelig når store brett spilles av. Utførelsen av denne oppgaven krever at dere implementerer et 'drag-and-move' interface: når brukeren klikker et sted på brettet og holder knappen samtidig med å flytte på musen, skal brettet (eller vinduet) flyttes etter bevegelsen til musen.

Obligatorisk oppgave 2

Korrekt implementasjon av kjernen til programmet representerer den andre obligatoriske oppgaven i faget. Dere må få godkjent denne oppgaven for å kunne levere inn semesteroppgaven.

Godkjenning av denne innleveringen gjøres ved demonstrasjon for din kontaktperson.

Dere skal allerede hatt et møte med deres kontaktperson etter utførelse av oppgavesett 4. Dette møtet er obligatorisk. Hvis dere ikke har hatt dette møte, så må dere kontakte deres kontaktperson snarest.

Etter utførelse av oppgavene over, skal dere møte kontaktperson enda en gang. Dere må demonstrere følgende punkter for å få godkjent:

- innlasting av GoL mønstre i et gitt filformat
- visualisering av mønsteret i et grafisk vindu i programmet
- animasjon av mønsteret etter spillereglene til GoL
- presentasjon av bruk av exceptions og håndtering av feil
- diskusjon av samarbeidet i gruppen

Merk: det må komme tydelig frem at alle i gruppen har bidratt nok til å bestå faget. Hvis dette ikke er tilfellet, vil studenten som ikke har bidratt nok ikke få godkjent og får dermed ikke levere semesteroppgaven. Det kan hende at studenter blir referert til fagansvarlig for godkjenning hvis det ikke kommer tydelig frem at en student har bidratt.

Absolutt siste frist for møte: uke 16 (husk at frist for møte etter oppgavesett 4 er i uke 14)

Etter at dere har fått godkjent obligatorisk oppgave 2, så har dere nå implementert kjernen til programmet: en fungerende implementasjon av GoL med støtte for innlasting av mønstre fra internett. Hvis dere har tid, kan dere nå se på ulike utvidelser av programmet. Se utvidelsesoppgavene fra oppgavesettene og andre utvidelsesoppgaver. Det anbefales at du fokuserer på oppgaver som passer dine interesser. Husk at det er en balanse mellom hvor mange oppgaver du utfører og hvor dypt du har tid til å dykke inn i hver oppgave. Her gjøres ingen anbefalinger: utførelse av en-to utvidelser (per person) der hver utvidelse er meget grundig utført kan være like bra som utførelse av tre-fire utvidelser som er utført etter minstekrav. Dere kan også velge å starte rett på oppgavesett 6 og 7, men anbefalingen her er å starte på disse etter at relevante emner har blitt gjennomgått i forelesning (uke 13 for sett 6 og uke 16 for sett 7).

Vedlegg 1

Følgende metoder kan brukes til å hente ut beskrivelse av 'bounding box'et til spillet, slik at man slipper å teste med hele brettet. Dette er derfor et alternativ til toString() metoden, som skriver informasjon for hele brettet. For enkelhetens skyld, antar vi at typen til en celle er en boolean. Gjør eventuelle endringer hvis representasjonen av spillebrettet er annerledes. Spør studentassistent om hjelp hvis noe er uklart.

```
public String getBoundingBoxPattern() {
	if(board.length == 0) return "";

	int[] boundingBox = getBoundingBox();
	String str = "";
	for(int i = boundingBox[0]; i <= boundingBox[1]; i++) {
	   for(int j = boundingBox[2]; j <= boundingBox[3]; j++) {
	        if(board[i][j]) {
	          str = str + "1";
	        } else {
	          str = str + "0";
	        }
	   }
	}

	return str;
}

private int[] getBoundingBox() {
	int[] boundingBox = new int[4]; // minrow maxrow mincolumn maxcolumn
	boundingBox[0] = board.length;
	boundingBox[1] = 0;
	boundingBox[2] = board[0].length;
	boundingBox[3] = 0;
	for(int i = 0; i < board.length; i++) {
		for(int j = 0; j < board[i].length; j++) {
			if(!board[i][j]) continue;
			if(i < boundingBox[0]) {
				boundingBox[0] = i;
			}
			if(i > boundingBox[1]) {
				boundingBox[1] = i;
			}
			if(j < boundingBox[2]) {
				boundingBox[2] = j;
			}
			if(j > boundingBox[3]) {
				boundingBox[3] = j;
			}
		}
	}

	return boundingBox;
}
```