Objektorienterad programmering med Java, 10 hp
ST 2010

Inlämningsuppgift 2
Spelet Nm

Sebastian Lundström (selu7901)

20 juni 2010

1 Inledning

Denna rapport behandlar design och implementation av spelet Nm. Programmet skall implementeras i programmeringsspråket Java.

2 Installation

Dessa instruktioner förutsätter att du befinner dig i en terminal i samma mapp som programmets Java-filer och Makefile. Du förutsätts även ha tillgång till (minst) Java 5 och programmet make. Allt som beskrivs i denna sektion beskrivs i mer detalj i programmets Makefile.

För att både kompilera och köra programmet skrivs enklast:

```
$ make
```

Detta kommando kompilerar alla .java-filer och kör sedan det resulterande programmet med 20 stickor. Vill du endast kompilera programmet skriver du:

```
$ make compile
```

Vill du endast köra programmet och kunna välja antalet stickor skriver du följande, där 20 förstås kan ändras efter behov:

```
$ java Nm 20
```

3 Körexempel

Följande är ett körexempel på en spelomgång mellan en människa och datorn, med 20 stickor:

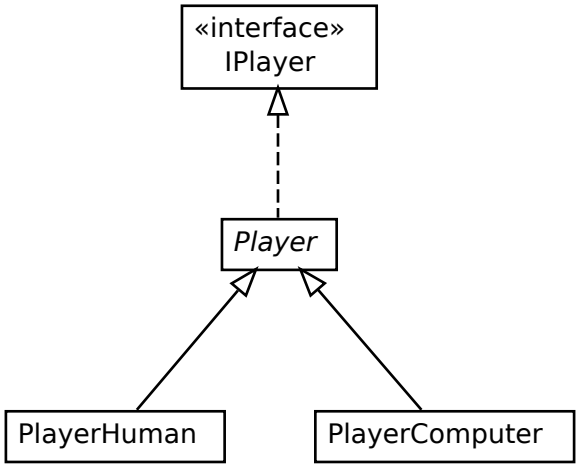
```
$ java Nm 20
Welcome to a game of Nm!
>>> Remaining sticks: 20 <<<
Human 1, your move: 10
Human 1 removes 10 sticks.
>>> Remaining sticks: 10 <<<
Computer 1, your move.
Computer 1 removes 5 sticks.
>>> Remaining sticks: 5 <<<
Human 1, your move: 4
You must remove 1 to 2 sticks.
Human 1, your move: 1
Human 1 removes 1 stick.
>>> Remaining sticks: 4 <<<
Computer 1, your move.
Computer 1 removes 2 sticks.
>>> Remaining sticks: 2 <<<
Human 1, your move: asdf
You must choose an integer!
Human 1, your move: 1
Human 1 removes 1 stick.
The winner is ... Human 1!
Computer 1: Nooo, I lost!
$
```

4 Systemdesign

Här diskuterar jag hur jag valde att utforma klasserna i systemet, och kortfattat om hur de interagerar med varandra.

4.1 Spelare

Enligt specifikationen behövs en abstrakt basklass för spelare, samt två subklasser för människa och dator. Eftersom alla spelare måste bete sig gentemot omvärlden på likartade sätt valde jag att skapa ett spelargränssnitt som basklassen (åtminstone delvis) implementerar. Överallt där någon sorts spelare krävs förväntas alltså ett gränssnitt i stället för någon specifik klass. Se figur:



Figur 1: Klassdiagram över spelare.

IPlayer definieras enligt följande:

void takeTurn() Utför ett drag.

String getName() Returnera spelarens namn.

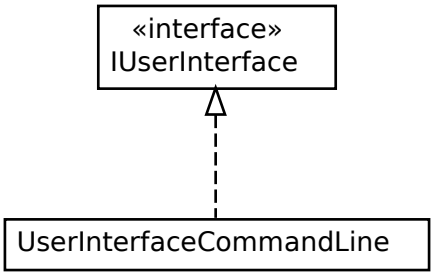
void setName(String name) Sätt spelarens namn. Argumentet *name* måste vara icke-null.

void won() Notifiera spelaren om att den har vunnit.

void lost() Notifiera spelaren om att den har förlorat.

4.2 Användargränssnitt

Användargränssnitt är något som kan tänkas ändras i framtiden (med t.ex. grafiska eller nätverksbaserade sådana), så jag upprättade ett gränssnitt även för detta. Det enda som har implementerats är ett gränssnitt för kommandoraden. Se figur:



Figur 2: Klassdiagram över användargränssnitt.

IUserInterface definieras enligt följande:

String promptForString(String prompt) Visar meddelandet *prompt* och ber användaren mata in en sträng.

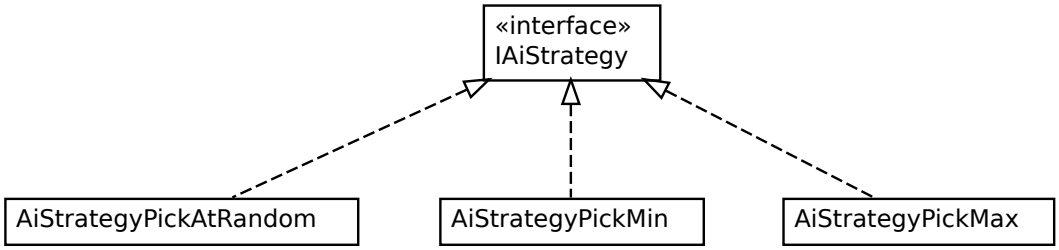
Integer promptForInteger(String prompt) Visar meddelandet *prompt* och ber användaren mata in ett heltal.

void display(String msg) Visar meddelandet *msg*.

4.3 Spelstrategier

En av de mer intressanta aspekterna i spelet är hur datorspelare väljer sina drag. För att hålla nere onödig koppling mellan klasser valde jag även här ett interface. Tanken är att en datorspelare inte vet något mer om sin strategi än vad interfacet (vilket är minimalt) avslöjar.

När en datorspelare skapas skickas en instans av någon strategi in i dess konstruktor. När datorspelaren sedan skall utföra sitt drag överlåts det till strategin. Detta är en implementation av designmönstret ”strategy”, där tanken är att kapsla in olika algoritmer bakom ett interface. Se figur:



Figur 3: Klassdiagram över datorns spelstrategier.

IAiStrategy definieras enligt följande:

Move chooseMove(Rules rules) Väljer ett drag i enlighet med reglerna i *rules*.

4.4 Övrigt

Återstoden av systemet är enstaka klasser:

Nm Spelets startklass, vilken innehåller main-metoden. Skapar instanser av relevanta klasser och låter en instans av GameOfNm ta över kontrollen. Publika metoder:

static void main(String[] args) Anropas när programmet startas.

GameOfNm Sköter själva spelet. Publika metoder:

void play() Genomför en omgång av spelet.

PileOfSticks Högen med stickor som spelarna arbetar med. Publika metoder:

void removeSticks(Integer sticks) Tar bort *sticks* antal stickor från högen.

Integer sticksLeft() Returnerar antalet stickor kvar i högen.

Move Ett drag som består av ett antal stickor som en spelare vill plocka från högen. Publika metoder:

Integer sticks() Returnerar antalet stickor som draget avser.

Rules Spelets regler. Bestämmer om ett drag är giltigt, om spelet är slut och kan svara på hur många stickor som får plockas för tillfället. Publika metoder:

Boolean isIllegalMove(Move move) Returnerar sant om draget *move* är ogiltigt, annars falskt.

Integer minAllowedSticks() Returnerar det minsta antalet stickor som får plockas just nu.

Integer maxAllowedSticks() Returnerar det högsta antalet stickor som får plockas just nu.

Boolean isGameOver() Returnerar sant om spelet är slut, annars falskt.

Util En abstrakt klass som innehåller ett fåtal statiska metoder för t.ex. slumptalsgenerering. Publika metoder:

static void throwIfNull(Object obj) Kastar ett IllegalArgumentException om *obj* är null.

static Integer randomIntegerBetween(Integer min, Integer max) Returnerar ett slumpmässigt heltal i intervallet $[min, max]$.

5 Programflöde

Programmet börjar i main-metoden som återfinns i klassen Nm. Den skapar de objekt som själva spelklassen, GameOfNm, behöver. Sedan startas spelet och GameOfNm-instansen tar över.

GameOfNm består i princip av en enkel spelloop. Först slumpas vem av spelarna som skall börja, och gränssnittet ombeds skriva ut något välkomnande till spelarna. Sedan går programmet in i en loop där nästa spelare ombeds utföra sitt drag. När det bara finns en sticka kvar i högen avslutas loopen och vinnaren annonseras. Spelet avslutas.

När en människospelare skall utföra sitt drag läses ett heltal från gränssnittet. Med detta tal skapas ett drag som direkt undersöks genom att rådfråga reglerna. Är draget ogiltigt efterfrågas ett nytt heltal. Till slut utför spelaren draget (som då är giltigt) och turen går vidare.

När en datorspelare skall utföra sitt drag frågar den sin strategiinstans. Strategin förutsätts returnera ett giltigt drag som utförs direkt.

När en spelare har utfört sitt drag frågar spelloopen reglerna om spelet är slut, annars går turen vidare till nästa spelare.