Rapport: Nm-spelet

Namn: Ludvig Fendert

E-post: l.fendert@gmail.com

Inledning

Jag har skapat ett Java-program som hanterar spelet Nm, där två spelare turas om att ta stickor från en hög. Den spelare som inte kan göra ett giltigt drag (när 1 sticka återstår) förlorar. Uppgiften bestod i att utveckla denna kod med tydlig objektorienterad struktur, användarvänlig inmatning och en NPC(datorn, non-playable-character) med en enkel strategi.

Programstruktur

Koden är uppdelad i fem klasser:

Player (abstrakt superklass)

- Beskriver gemensamma egenskaper (namn och metoden takeSticks()) för alla spelartyper.
- Metoden takeSticks (int sticksRemaining) är abstrakt och implementeras av de olika spelartyperna.

HumanPlayer

- Hanterar användarinmatning med `Scanner`.
- Implementerar `takeSticks(int sticksRemaining)` genom att be användaren ange ett giltigt antal stickor och validera inmatningen.

NPC (Datorspelare)

- Implementerar `takeSticks(int sticksRemaining)` genom att välja ett slumpmässigt antal stickor mellan 1 och `sticksRemaining / 2`.
- Detta gör att NPC:n varierar sina drag och inte alltid tar samma antal stickor.

NmGame

- Hanterar spel-logiken, turordningen och kontrollerar spelets regler.
- Validerar varje spelares drag och avgör när spelet är slut.
- Skriver ut aktuell spelstatus efter varje drag.

Main

- Läser in spelarens namn och startantal stickor från kommandoraden.
- Skapar spelare och startar spelet genom `NmGame.play()`.

Exempel på terminalkörning med Maven

Kör följande kommandon i roten av projektet:

Mvn clean compile

Java -cp target/classes com.uppsala.Main 15

Då kompileras koden till `target/classes` och programmet körs därifrån med 15 stickor.

Implementering av spelets regler

Spelets regler implementeras i `NmGame`. Där kontrolleras:

1. Om ett drag är giltigt (mellan 1 och `sticksRemaining / 2`).
2. Om spelet är slut (bara 1 sticka kvar).
3. Vem som har vunnit.

Val av indata hantering och validering

I projektet har jag lagt stor vikt vid att säkerställa att användarindata hanteras på ett robust och användarvänligt sätt. Eftersom spelet bygger på att spelaren anger antalet stickor att ta, är det avgörande att programmet inte kraschar vid felaktig inmatning. För att uppnå detta har jag implementerat följande lösningar:

- Robust inmatnings validering med try-catch:
  I metoden `takeSticks` i klassen HumanPlayer används en try-catch-sats för att fånga upp `InputMismatchException`. Detta innebär att om användaren skriver in något som inte är ett heltal visas ett tydligt felmeddelande "Ogiltig inmatning. Vänligen ange ett heltal." och programmet ger spelaren en ny chans att ange ett korrekt värde. Denna metod förhindrar att programmet kraschar och förbättrar användarupplevelsen genom att direkt fånga upp och hantera inmatningsfel.
- Resurshantering med try-with-resources:
  Istället för att skapa en separat `Scanner` i varje klass har jag centraliserat inläsningen genom att skapa ett `Scanner`-objekt i Main-klassen. Genom att använda

try-with-resources säkerställs att `Scanner` stängs automatiskt när den inte längre behövs, vilket minskar risken för minnesläckor och andra resursrelaterade problem.

Designbeslut och motivering

- Användning av arv och polymorfi: Genom att ha en abstrakt `Player`-klass kan `NmGame` hantera både mänskliga och datorstyrda spelare utan att känna till deras specifika implementationer.
- Separat `NmGame`-klass: För att hålla logiken isolerad från `Main` och spelarna.
- Slumpmässig NPC-strategi: För att göra NPC:n mer dynamisk och oväntad.
- Validering i `HumanPlayer` vid inmatning: För att omedelbart fånga ogiltiga drag och undvika extra kontrollsteg i `NmGame`.
- Jag har också valt att använda konstanter i detta projekt, vilket gör koden enklare att underhålla och refaktorera eftersom förändringar endast behöver göras på ett ställe.

Utmaningar och lösningar

- Hantera ogiltiga drag: Lösningen i HumanPlayer består av en while-loop kombinerad med en try-catch-sats. Detta gör att programmet inte bara tvingar användaren att ange ett giltigt värde inom det tillåtna intervallet utan också säkerställer att inmatning av icke-heltal fångas upp och hanteras, vilket förhindrar att programmet kraschar.
- Göra NPC mindre förutsägbar: Genom att använda `Random` tar den olika drag varje gång.

Vad jag kunde ha gjort annorlunda

- Smartare NPC: Just nu tar den slumpmässigt mellan 1 och max, men en strategi som lämnar motståndaren i en dålig position hade varit mer utmanande.
- Flerspelarläge: Spelet kunde byggas ut för att låta två människor spela mot varandra eller ha flera NPC:er.
- Jag kunde gjort en ytterligare klass som hette Constants.java, där jag samlat alla strängkonstanter på ett ställe. Detta skulle ha gjort det enklare att hantera och ändra meddelanden i programmet. I ett större program hade detta varit en bättre lösning, eftersom antalet strängar annars hade blivit väldigt stort för att hanteras individuellt i varje klass.

Sammanfattning

Spelet Nm är nu implementerat med en tydlig objektorienterad design och en varierande NPC-strategi. Genom att använda arv och polymorfi är koden flexibel och lätt att utöka. Valideringen hanteras effektivt, och spelet följer reglerna enligt uppgiften.

Teknisk information

javac --version
javac 22.0.1

java --version
java 22.0.1 2024-04-16
Java HotSpot(TM) 64-Bit Server VM (build 22.0.1+8 16, mixed mode, sharing)

Jag använde IntelliJ under utvecklingen men spelet kan även kompileras och köras från terminalen.