Dokumentation DiceGame.

Innehåll:

Kortare sammanfattning av filerna:

Mer ingående sammanfattning av filerna:

Filer som tillhör detta spel:

Die (Class)

- Denna fil innehåller en klass för att kunna skapa tärningar av valfri storlek och tillhörande metoder för att kunna rulla dem och spara värdet. Den innehåller funktioner från den importerade klassen Random som används för att generera ett slumpmässigt nummer mellan ett och max antal sidor den skapade tärningen har. Instansvariablerna för denna klass är privata för att de inte skall kunna ändras på utan att åberopa getter och settermetoder som finns i klassen.

Player (Class)

- Denna klassfil skapar en spelare med ett namn och en poängräknare. Samma klass används även för ett multiplayer spel och därför finns även den importerade funktionen att kunna skapa och lagra flera tärningar till varje spelare med hjälp av ArrayList.
- Utöver detta finns metoder för att spara namn, poäng och tärningar med hjälp av Getter och Setter metoder för namn och poäng samt metoder för att skapa och rulla tärningar och spara samt öka poäng.

DiceGame(main metod)

- I denna fil finns main metoden som låter oss köra spelet och själva spelets kod. Det första som återfinns här är en Scanner för att kunna hämta input från användaren och sen börjar spelet. Först frågas användaren om namn och spelaren skapas från player-klassen.

- Sedan följer ett stycke som frågar om spelaren vill se reglerna för spelet och om de väljer ja förklaras reglerna innan spelaren får välja om de vill spela eller inte. Väljer de ja startar spelet, väljer de nej avslutas programmet. Alternativen är konstruerade med if-satser och den inbyggda metoden **System.exit();** för att stänga programmet om detta alternativ väljs.
- Väljer spelaren att spela frågar programmet hur många rundor de vill spela och hur många sidor de vill ha på sin tärning innan tärningen skapas via metoden för detta i player-klassen.
- Sedan efterföljer ett stycke som kontrollerar om spelaren anger giltiga gissningar. Om gissningarna är ogiltiga, antingen ett otillåtet tecken eller ett nummer de inte kan slå med sin skapade tärning, får de gissa på nytt tills de anger ett giltigt heltal inom gränsen för tärningen. Denna funktion ligger i for-loopen som används för att iterera genom alla rundor och är konstruerad av try/catch som låter programmet bortse från exeptions och fortsätta köra loopen. Den innehåller också en integrerad if-sats för att kontrollera att numret spelaren gissar på är ett rimligt nummer.
- När spelaren matat in en giltig gissning rullas tärningen och programmet berättar om det var rätt eller fel samt ger spelaren ett poäng vid rätt innan det går vidare till nästa runda.
- När alla rundor passerat med giltiga gissningar anges resultatet och lämpliga meddelanden för slutresultat skrivs ut innan programmet frågar om spelaren vill spela igen. Väljs ja får spelaren ange nytt antal rundor och sidor på tärningen. Väljs nej avslutas programmet.

Mer ingående beskrivning av filerna:

Die klassen:

- Tärningsklassen innehåller två int variabler:
    - sides; = variabeln som sparar värdet för hur många sidor tärningen som skapas skall ha.
    - value; = variabeln som sparar värdet tärningen visar när den rullas.
- Utöver dessa finns också en instansvariabel av typen Random som heter dieroll. Mer om den längre ner.
- I klassens konstruktormetod anger vi parametern (int sides) för att bestämma att varje gång en ny tärning skapas behöver objektet få med sig värdet av antalet sidor den skall ha och vi använder också this. för att specifiera vilka variabler som refereras i klassen.
- Efter detta följer två metoder för att kunna ändra de privata klassvariablerna. **getSides()** för variabeln sides och **getValue()** för variabeln value.
- **Random**-klassen är importerad och finns inbyggd i Java. Den innehåller metoder för att generera att slumpat tal mellan 0 och det nummer som anges som parameter. Här används Random i metoden **roll()**. Vi sparar värdet som genereras i variabeln value; genom att kalla på metoden nextInt i Randomklassen och ange variabeln

sides; som parameter. Eftersom Java alltid börjar räkna från 0 ökar vi också det genererade numret med ett när vi sparar in det i variabeln value;.

Player-klassen:

- I playerklassen finns variabler för namn och poäng, name; respektive points;, och en ArrayList som skall hålla värden av typen Die. Den behövs för att kunna skapa och spara flera tärningar till samma spelare även om spelet DiceGame bara använder en. Getter och setter metoder som återfanns i tärningsklassen finns även här för variablerna name; och point; med tillhörande 'this' för att kunna använda samma namn på variablerna uta natt Java blir förvirrat.
- Sedan följer en metod för att skapa och lägga till tärningar i **ArrayList.** Vilken är en importerad klass för att skapa och hantera Collections(en utökad version av Arrays). När tärningarna skapas behöver de få med informationen om hur många sidor de ska ha för att fungera.
- Vi har även en metod för att rulla tärningar som heter **rollDice()** som använder sig av en enhanced for-loop för att iterera genom alla tärningar i en spelares Arraylist och rulla dem. I DiceGame har varje spelare bara en tärning så loopen kan verka överflödig men för att kunna återanvända koden i denna klass så är den en bra lösning att använda här.
- Metoden **getDieValue();** innehåller också en for-loop för att istället iterera igenom resultatet på tärningarna och "spara om" värdet från en Die-variabel till en Int-variabel.
- Sedan följer en metod för att öka spelarens poäng med ett.

DiceGame(main-metod):

- **Scanner:** En Scanner importeras och skapas med namnet input.

- **Sides; och guessedNumber; :** Variabler som skapas för att senare tilldelas värden. De har inga värden från början.

- **Boolean replay; :** En variabel som används för att kunna spela spelet igen. Satt till true; för att den alltid skall köras om inget anges för att ändra på det.

- **Name; och new Player():** Spelaren frågas efter namn och svaret sparas i variabeln name; som används i metoden new Player() för att skapa en spelare med det namnet.

- **Rules; och if-sats:** Variabeln rules; skapas för att kunna testa ifall spelaren vill höra spelets regler. Detta görs med en if-sats. Svarar splearen "yes" kommer reglerna skrivas ut på skärmen och spelaren får ett nytt val om de vill spela eller inte.

Svarar de “yes” igen kommer de till spelet men svarar de något annat kommer programmet att avslutas med System.exit(0);.

  - **toLowerCase(); :** Med den inbyggda metoden toLowerCase() gör Java om användarens input till små bokstäver så att man kan svara med både versaler och gemener. Annars är jämförelser i Java Case-sensitive.

- **While-loop :** För att kunna köra spelet om och om igen utan att programmet avslutas ligger det i en while-loop som körs så länge variabeln replay är satt till true. Längre ner följer en funktion för att ändra på detta och avsluta spelet.

- **Rounds; sides; och addDie():** Har spelaren valt att starta spelet (antingen efter de läst reglerna eller inte valt att se reglerna alls) kommer spelet att börja med att fråga hur många rundor spelaren vill spela och spara antalet rundor i variabeln rounds;. Sen frågas hur många sidor på tärningen spelaren vill ha vilket sparas i variabeln sides; som sedan används för att skapa en tärning med metoden addDie() som finns i player-klassen.

- **For-loop och while-loop:** För att man skall kunna gissa på nummer varje runda används en for-loop för att iterera genom alla rundor. Först i for-loopen ligger dock en while-loop som är satt till true för att den alltid skall köras. I denna while-loop ligger en funktion som heter try/catch.
  - **Try/catch:** Denna funktion kommer från den importerade klassen “InputMismatchExeption” som låter oss fånga upp ett fel som annars hade fått programmet att sluta fungera och istället berätta vad Java skall göra ifall det inträffar. I detta fall fångar det upp händelsen ifall spelaren skulle gissa på något annat än en siffra (MismatchExeption).
  - **If-sats:** I Try har jag också bakat in en if-sats ifall spelaren skulle mata in ett nummer som är utanför ramen för giltiga nummer som återfinns på deras tärning. I båda dessa fall låter spelet spelaren gissa på nytt och ingen runda blir förlorad.

- **RollDice() och ifsats med increaseScore():** När spelaren matat in en giltig gissning kommer tärningen att rullas med metoden rollDice() från player-klassen och resultaten kommer att jämföras med spelarens gissning med en if-sats. Om spelaren gissade rätt kommer ett poäng tilldelas spelaren med metoden increaseScore() från player-klassen innan nästa runda påbörjas. Om spelaren gissade fel kommer loopen bara att börja om och nästa runda påbörjas tills inga rundor återstår.

- **Print(“”), getPoint() och is-sats:** Efter loopen avslutats ligger en tom rad för estetiska ändamål. Raden “calculating results” är också den bara estetisk och likaså de två efterföljande tomma raderna. Sedan skrivs spelarens poäng ut med metoden getPoint() och en if-sats som jämför och skriver ut olika meddelanden baserat på slutresultat.

- **Play again :** Detta stycket börjar med en scanner input för att "äta upp" tidigare input som inte registrerats. Sedan följer en tom rad för formatering innan användaren frågas om de vill spela igen.
    - **If-sats för playagain:** Om användaren svarar ja för att spela igen nollställs poängen och spelarens tärningar tas bort så att en ny tärning kan skapas.
    - Väljer användaren inte att spela igen ändras variabeln replay till false; och loopen avslutas.

- Slutmeddelande och Scanner.close() : Innan spelet avslutas säger programmet hejdå. Scannern input stängs också här.

Beskrivning av hur spelet spelas:

1. Spelet frågar efter spelarens namn. Ange namnet du vill använda i spelet.
2. Spelet frågar om du vill höra reglerna för spelet: Skriv "yes" om du vill höra reglerna. Annars skriv "no".
    2.1. Om du skrev "yes" för att höra reglerna: Spelet förklarar reglerna och frågar om du vill spela: Skriv "yes" om du vill spela spelet eller "no" om du vill avluta utan att spela.
3. Om du svarade "no" på om du ville höra spelets regler eller om du valt att spela efter att reglerna visats: Spelet frågar hur många rundor du vill spela: Ange hur många rundor du vill spela med en siffra.
4. Spelet frågar hur många sidor du vil ha på din tärning: Ange hur många sidor du vill ha med en siffra.
5. Spelet frågar vad du gissar att tärningen visar på första rullningen: Ange en siffra mellan ett och max antal sidor på tärningen du valde att skapa i punkt 4.
6. Fortsätt att gissa på ett nummer för varje runda tills programmet presenterar ditt resultat.
7. Spelet frågar om du vill spela igen. Skriv "yes" för att spela igen. Skriver du något annat avslutas programmet.