RAFLEBÆGER 4 (DIECUP 4)

I dette opgavesæt skal I arbejde videre med raflebægerprojektet fra kursets første tre uger. I skal bruge de ting, som I har lært om test og debugging til at konsolidere projektet.

Start med at se videoen *"Regression tests"*, som I finder under *Uge 10* på siden *Uge 9-15*. Videoen illustrerer, hvordan man laver regression tests for *Raflebæger 1*. Den illustrerer også, hvordan testserveren tester de regression tests, der bliver lavet, herunder hvad der skal til, for at regression testene bliver godkendt af testserveren.

Opgave 1

Download projektet ***DieCup 2*** (zip) og husk at pakke det ud, før I går i gang. Projektet indeholder en (korrekt) løsning af *Raflebæger 2* opgaven, hvor ***TestDriver*** klassen dog er udeladt, idet I nu skal bruge BlueJ's indbyggede testfaciliteter til at lave testklasser og testmetoder for ***Die*** og ***DieCup*** klasserne (dvs. regression tests).

For ***Die*** klassen skal I lave en testmetode for ***roll*** metoden, hvor I tester, at antal øjne ligger i det rigtige interval og har den rigtige middelværdi, samt at alle mulige antal øjne kan slås. Det kan I gøre som vist i ovennævnte video, men nu skal I gøre det for terninger med et vilkårligt antal sider i intervallet [2,10]. Dvs. at koden fra videoen skal indsættes som krop i en ***for*** løkke (der gennemløber intervallets værdier). Koden skal modificeres lidt, da terningerne nu har et variabelt antal sider. I behøver ikke at lave en testmetode for ***getEyes*** metoden, idet den er en "triviel" accessormetoder, der blot returnerer værdien af en feltvariabel.

For ***DieCup*** klassen skal I lave en testmetode for ***roll*** metoden, hvor I tester, at antal øjne ligger i det rigtige interval og har den rigtige middelværdi, samt at alle mulige antal øjne kan slås, og at feltvariablen ***maxEyes*** opdateres korrekt. Det kan I gøre som vist i ovennævnte video, men nu skal I tjekke alle raflebægre, hvor hver af de to terninger har et vilkårligt antal sider i intervallet [2,10]. Dvs. at koden fra videoen skal indsættes som krop i to nestede ***for*** løkker (der begge gennemløber intervallets værdier). Koden skal modificeres lidt, da terningerne nu har et variabelt antal sider. I skal også lave en testmetode for ***resetMaxEyes*** metoden, mens I ikke behøver at teste de trivielle accessormetoder ***getEyes*** og ***getMaxEyes***.

Når I har lavet regression testene, skal I afprøve dem ved at kalde klassemetoden ***test*** i ***TestServer*** klassen med parameteren "DC4-1". Hvis testserveren finder fejl, skal l gennemgå jeres tests og forsøge at rette dem.

Opgave 2

Download projektet ***DieCup 3*** (zip) og husk at pakke det ud, før I går i gang. Projektet indeholder en (ikke helt korrekt) løsning til *Raflebæger 3* opgaven, hvor ***TestDriver*** klassen igen er udeladt. Brug BlueJ's testfaciliteter til at lave testklasser og testmetoder for ***Die*** og ***DieCup*** klasserne.

Die klassen er den samme som i opgave 1 (på nær en lille fejl, som I senere skal finde), så her kan I bruge den samme testmetode som i opgave 1.

For ***DieCup*** klassen kan I også tage udgangspunkt i testmetoden fra opgave 1, men nu skal I tjekke 100 tilfældigt valgte raflebægre, hvor der for hvert raflebæger skal gælde, at antallet af terninger ligger i intervallet [1,4], mens hver terning har et vilkårligt antal sider i intervallet [2,10]. I skal også lave en testmetode for ***resetMaxEyes*** metoden, mens I ikke behøver at teste de trivielle accessormetoder ***getEyes*** og ***getMaxEyes***.

Anvend jeres tests til at finde fejlene i ***Die*** og ***DieCup*** klasserne og ret dem, idet I samtidig indsætter en kommentar i den pågældende linje, der fortæller, hvad I har rettet.

Til sidst skal I afprøve det, som I har lavet i opgave 2, ved at kalde klassemetoden ***test*** i ***TestServer*** klassen med parameteren "DC4-2". Hvis testserveren finder fejl, skal l gennemgå jeres tests og forsøge at rette dem.

Opgave 3

Download projektet ***DieCup 4*** (zip) og husk at pakke det ud, før I går i gang. Projektet indeholder en videreudvikling af *Raflebæger 3* opgaven, hvor der er tilføjet en ***Game*** klasse, der gør det muligt at spille mod en computer om, hvem der er bedst til at slå med et raflebæger (dvs. får flest øjne i sine kast).

Man afvikler et spil ved at kalde klassemetoden ***play*** i ***Game*** klassen. Klassemoden tager en heltalsparameter, der angiver hvilke terninger, der er i raflebægeret. Hvert ciffer angiver antallet af sider i en terning, dvs. at *452* angiver, at raflebægeret har tre terninger med henholdsvis *4*, *5* og *2* sider. Parameterværdien må ikke indeholde cifrene *0* og *1* (idet terninger med dette antal sider ikke er tilladt).

Når ***play*** metoden kaldes, afvikles der et spil, hvor hver spiller (dvs. computeren og brugeren) laver 3 kast med raflebægeret, hvorefter resultatet (det samlede antal opnåede øjne) udskrives som en linje i BlueJ's terminal på formen:

```
Computer:  21      User:  17      Winner(s): Computer
```

Herudover returnerer ***play*** metoden det antal øjne, som brugeren har slået (hvilket bruges af ***Testserveren***). Prøv at afvikle nogle spil. I vil så opdage, at computeren altid vinder, hvilket skyldes, at der er nogle fejl i ***Game*** klassen.

Game klassen er med vilje programmeret" kluntet", hvilket forhåbentlig gør det lidt vanskeligere for jer at finde fejlene ved blot at kigge på koden – hvilket I bør lade være med.

For computeren foretages et kast på den forventede måde, dvs. at man skaber et raflebæger med de angivne terninger, kalder ***DieCup*** objektets ***roll*** metode og aflæser resultatet ved hjælp af ***DieCup*** objektets ***getEyes*** metode.

For brugeren foretages kastet på en noget mere omstændelig måde, idet man for hver terning skaber et ***Die*** objekt, kalder ***roll*** metoden og aflæser resultatet ved hjælp af ***getEyes*** metoden. Det antal øjne, som man finder undervejs, summeres og angiver resultatet af kastet.

Start BlueJ's debugger (eller en lignende debugger, f.eks. IntelliJ's) og indsæt et breakpoint umiddelbart under hver af de tre kommentarer i ***play*** metoden. Det er de steder, der starter opgørelsen af, hvor mange øjne computeren får, hvor mange øjne brugeren får, og hvem der vinder.

Start dernæst nogle spil ved at kalde ***play*** metoden med parameterværdierne *452 og 29*. Da antallet af øjne bestemmes ved hjælp af ***nextInt*** metoden i ***Random*** klassen, vil spillene forløbe forskelligt fra gang til gang, selv om parameterværdien er den samme. Hver gang I når et breakpoint, skal I foretage nogle skridt, hvor I inspicerer de efterfølgende kodelinjer og værdierne af de lokale variabler i ***Game*** klassen. Herved bør I kunne finder fejlene i ***Game*** klassen. Når I har fundet en fejl, rettes den i ***Game*** klassens kode, idet I samtidig i den pågældende linje indsætter en kommentar, der fortæller hvad I har rettet.

Til sidst skal I afprøve det, som I har lavet i opgave 3, ved at kalde klassemetoden ***test*** i ***TestServer*** klassen med parameteren "DC4-3". Hvis testserveren finder fejl, skal I gentage jeres debugging og fejlretning indtil testserveren godkender jeres projekt.

Nedenstående afleveres til instruktoren som én samlet zip fil:

- Den færdige version af ***DieCup 2*** projektet i opgave 1.
- Den færdige version af ***DieCup 3*** projektet i opgave 2.
- Den færdige version af ***DieCup 4*** projektet i opgave 3.

Undgå at lave zip-filer inde i andre zip-filer. Det tager for lang tid at pakke ud.