Programmeringsuppgift 5

Avsikt

Avsikten med programmeringsuppgift 5 är att du ska träna på att implementera interface, ärva abstrakt klass och eventuellt ärva vanlig klass. Uppgiften handlar om ett enkelt tärningsspel där det gäller att kasta tärningen så många gånger som möjligt utan att få en etta. Själva spelet är givet (klassen Game) men du ska skriva ett antal klasser för att olika speltest ska fungera.

Det är viktigt att du själv skriver all din kod och att du inte kopierar från någon studiekamrat. Men det är alltid bra att diskutera olika sätt att lösa ett problem med andra studenter.

I filen ***DA339A_P5_HT17.zip*** hittar du ett antal filer som används i programmeringsuppgiften. Dessa filer ska placeras i ett paket med namnet ***p5***. Filerna är:

Dice.java, Player.java, Game.java, NegativeSidesException.java, PredictableDice.java, TestSimpleDice.java, TestOrdinaryPlayer.java, TestCheater.java, TestProbabilityDice.java, TestP5.java, TextWindow.java

Redovisning

Inlämning

Din lösning av uppgiften lämnas in via It's learning *senast kl 09.00 måndagen den 7/12*. Du ska placera *samtliga källkodsfiler i paketet p5* i en zip-fil. Det innebär att lösningen kommer innehålla minst följande java-filer:

- Dina lösningar dvs:
  SimpleDice.java, OrdinaryPlayer.java, TestDice.java, Cheater.java och eventuellt *ProbabilityDice.java (extra)*
- Övriga filer som ingår i uppgiften (de som du kopierat från DA339A_P5_HT17.zip)
- Javadoc-dokumentation för paketet *p5*. Klasserna SimpleDice, OrdinaryPlayer och Cheater ska vara javadoc-kommenterade. Dokumentationen ska vara i katalogen *doc* (katalogen som skapas av Eclipse)

Zip-filen ska du ge namnet AAABBBP5.zip där AAA är de tre första bokstäverna i ditt efternamn och BBB är de tre första bokstäverna i ditt förnamn. Använd endast tecknen a-z när du namnger filen.

- Om Rolf Axelsson ska lämna in sina lösningar ska filen heta AxeRolP5.zip.
- Om Örjan Märla ska lämna in sina lösningar ska filen heta MarOrjP5.zip.
- Är ditt förnamn eller efternamn kortare än tre bokstäver så ta med de bokstäver som är i namnet: Janet Ek lämnar in filen EkJanP5.zip

Granskning

Din granskning ska omfatta 1-2 A4-sidor. Granskningen ska vara som pdf-dokument och *lämnas in via It's Learning innan du redovisar tisdagen den 8/12*. Namnet på granskningen ska vara samma som zip-filen, dvs. *AAABBBP5.pdf*.

Granskning av Programmeringsuppgift 5

Lösning inlämnad av Eva Lind

Granskare: Einar Bok

Datum: 7/12-2017

Funktion, lösning:
Hur är funktionen i de olika klasserna, dvs vilka instansvariabler och metoder innehåller klasserna. Hur är funktionen i metoderna? Finns det alternativa lösningar?

Indentering, metodnamn, variabelnamn mm:
Hur välskrivna är klasserna och metoderna? Är identifierarna väl valda?

Kommentarer:
Är klasser och metoder väl dokumenterade? Finns det delar i koden som borde varit kommenterade?

Javadoc-kommentarer:

Är javadoc-dokumentation genererad. Är alla klasser kommenterade? Är alla konstruktorer kommenterade och har @param-taggar för samtliga parametrar? Är alla metoder kommenterade, har @param-taggar för alla parametrar och @return-tagg för alla returvärden?

Redovisning

Redovisning sker *fredagen den 8/12*. Redovisningstid publiceras på It´s learning under *torsdagen den 7/12*. Kom väl förberedd till redovisningen. Kom i god tid till redovisningen så du är beredd då det är din tur. Se till att du är inloggad på en dator (eller har egen dator), att eclipse är igång på datorn och att det går att exekvera dina lösningar.

En redovisning sker genom att:

- Studentens lösningar körs med hjälp av *TestP5*
- Granskaren redogör för sina bedömningar
- Studenten svarar för sina lösningar
- Labhandledaren ställer kompletterande frågor
- De studenter i gruppen som inte redovisar är åhörare.

Godkänd uppgift signeras av läraren på lämpligt papper, t.ex. Redovisade uppgifter (se kurssidan). Du ska spara den signerade utskriften tills kursen är avslutad.

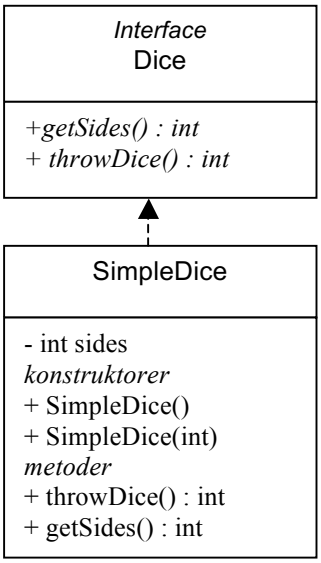
Om labhandledaren anser att det endast krävs *mindre komplettering för att lösningen ska godkännas* kan denna komplettering äga rum direkt efter redovisningen. Labhandledaren granskar kompletterad lösning då tiden medger.

Om labhandledaren anser att det endast krävs *mindre komplettering för att granskningen ska godkännas* kan denna komplettering äga rum direkt efter redovisningen. Labhandledaren granskar kompletterad granskning då tiden medger.

Uppgift 5a

Skriv klassen **SimpleDice** vilken ska implementera gränssnitter **Dice**.

```
public interface Dice {
    public int throwDice();
    public int getSides();
}
```



Klassen **SimpleDice** representerar en tärning med minst en sida. Det innebär att det inte ska gå att konstruera en tärning med mindre än en sida.

Klassen **SimpleDice** ska innehålla följande:

- Attributet **sides** av typen int.
- En **konstruktor** utan parametrar. Om denna konstruktor används ska tärningen vara 6-sidig.
  `public SimpleDice() {…}`
- En **konstruktor** som tar emot antalet sidor som parameter,
  `public SimpleDice(int sides) {…}`
  Om argumentet sides har ett värde <=0 ska undantaget **NegativeSidesException** kastas.
- Implementation av **Dice** (metoderna getSides() och throwDice()).

Testprogrammet **TestSimpleDice.java** kan ge en utskrift liknande denna (inmatningen av antalet sidor sker via inmatningsdialog, utskrifterna i TextWindow-fönstret) om användaren matar in 8, 6, 2, 1, 0, -3 och -10:

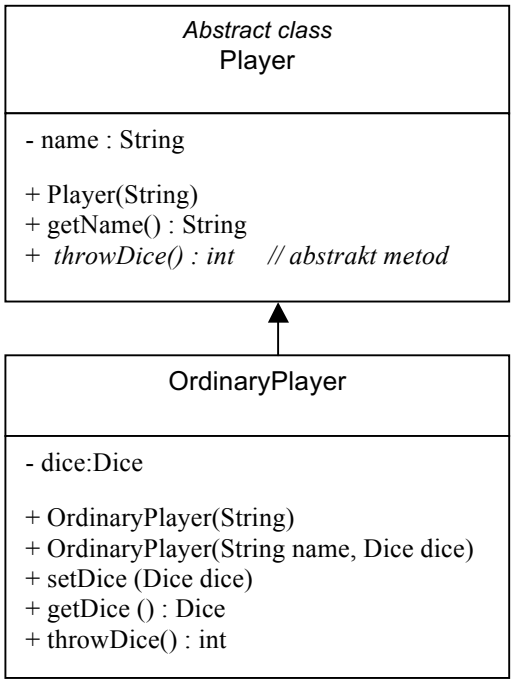
```
----- 100000 kast med 8-sidig tärning -----
1        12549
2        12392
3        12578
4        12490
5        12486
6        12479
7        12555
8        12471
----- 100000 kast med 6-sidig tärning -----
1        16634
2        16727
3        16620
4        16675
5        16688
6        16656
----- 100000 kast med 2-sidig tärning -----
1        50067
2        49933
----- 100000 kast med 1-sidig tärning -----
1       100000
p5.NegativeSidesException: Tärningen måste ha minst 1 sida: 0
p5.NegativeSidesException: Tärningen måste ha minst 1 sida: -3
```

Klassen **PredictableDice** visar en implementering av en förutsägbar tärning. Studera denna innan du löser uppgiften.

Uppgift 5b

Du ska skriva klassen **OrdinaryPlayer** vilken ska ärva den abstrakta klassen **Player**. Ett Player-objekt representerar en tärningsspelare. Spelaren har ett namn.

Klassdiagrammen till höger beskriver vad klassen OrdinaryPlayer behöver innehålla. Konstruktorn med enbart namn som argument ska ge en spelare med en 6-sidig tärning.



En testkörning av programmet **TestOrdinaryPlayer** kan se ut så här:

```
----- 1000000 kast av Stefan med 8-sidig tärning -----
1       125158
2       124975
3       124666
4       125544
5       125070
6       124393
7       125501
8       124693
----- 50 kast av Signe med 6-sidig tärning -----
1            6
2           10
3           10
4            6
5            4
6           14
----- 10 kast av Stefan med 3-sidig tärning -----
1            6
2            2
3            2
```

Nu kan du testköra det du hittills har gjort med följande program:

```
SimpleDice tärning = new SimpleDice( 6 );
Player spelare1 = new OrdinaryPlayer( "Gustav", tärning );
Player spelare2 = new OrdinaryPlayer( "Valborg", tärning );
Game spel = new Game( spelare1, spelare2 );
spel.play( true );
```

Körresultatet kan likna detta:

```
Gustav :6 5 6 1
Valborg :5 2 4 6 4 6 3 6 3 2 5 1
Valborg vann över Gustav  11-3
```

Uppgift 5c

Du ska skriva klassen **TestDice** vilken ska innehålla klassmetoden

```
public static void test( Dice dice, int nbrOfThrows )
```

Metoden ska kasta ett Dice-objekt angivet antal gånger (nbrOfThrows). Metoden räknar hur många gånger olika antal prickar inträffar och skriver sedan ut dessa. Genom att studera frekvenserna kan man sedan avgöra om tärningen är korrekt eller en fusktärning.

För att lösa detta problem behöver du ett sätt att lagra en räknare för varje antal prickar som kan inträffa. Ett smidigt sätt att hantera detta är att använda ett fält, lika stort som antalet sidor på tärningen.

```
int[] res = new int[ dice.getSides() ];
```

res[0] lagrar antalet gången en etta kommer upp
res[1] lagrar antalet gången en etta kommer upp osv.

Om tärningen är 12-sidig så kommer res[11] lagra antalet förekomster av 12 prickar.

Om du vet antalet prickar i senaste slaget så vet du också vilken räknare som ska ökas, nämligen

```
res[ antalPrickar - 1 ]++;
```

Om du kör nedanstående program kan du få ett körresultat som liknar det längre ner på sidan.

Testprogram

```
TestDice.test( new SimpleDice( 6 ), 1000000 );
TextWindow.println();
TestDice.test( new SimpleDice( 4 ), 1000000 );
```

Körresultat

```
1    166571
2    166805
3    166822
4    166699
5    166642
6    166461

1    250397
2    249954
3    249783
4    249866
```

Uppgift 5d

Nu har du ett enkelt men fungerande spel. Nu ska du tillverka en spelare, **Cheater**, som fuskar lite vid tärningsspel. Fusket består av att Cheatern lägger till en prick då och då.

Följande gäller för klassen Cheater:

- Klassen ska ärva Player.
- Cheater-objektet ska ange en prick för mycket vid ca hälften av kasten. Men om tärningen visar max antal prickar så låter Cheatern bli att fuska.

Förutom fusket ska Cheatern fungera på samma sätt som en OrdinaryPlayer.

Testprogrammet **TestCheater.java** kan ge ett körresultat liknande detta:

```
----- 1000000 kast av Stefan med 8-sidig tärning -----
1        62733
2       125227
3       124951
4       124999
5       124496
6       124949
7       125029
8       187616
----- 1000000 kast av Signe med 6-sidig tärning -----
1        83173
2       167102
3       166605
4       166352
5       167051
6       249717
----- 1000000 kast av Stefan med 2-sidig tärning -----
1       249972
2       750028
----- 1000000 kast av Signe med 1-sidig tärning -----
1      1000000
```

Om du kör nedanstående **testprogram** 10 gånger så bör Signe vinna oftare än Viktor.

```
Player spelare1 = new OrdinaryPlayer( "Viktor",new SimpleDice( 6 ) );
Player spelare2 = new Cheater( "Signe", new SimpleDice( 6 ) );
Game spel = new Game( spelare1, spelare2 );
TextWindow.println( "\nResultatet av tio spel" );
for( int i=0; i<10; i++ )
    spel.play( false );
```

Exempel på körresultat

```
Resultatet av tio spel
Oavgjort mellan Viktor och Signe  1-1
Viktor vann över Signe  4-3
Signe vann över Viktor  5-2
Signe vann över Viktor  13-12
Signe vann över Viktor  46-2
Viktor vann över Signe  11-0
Signe vann över Viktor  12-1
Signe vann över Viktor  5-0
Signe vann över Viktor  5-0
Signe vann över Viktor  21-12
```

Uppgift 5e

Nu ska du lägga till klassmetoden

```
public static void test( Player player, int nbrOfThrows )
```

i klassen TestDice. Genom att anropa denna metod kan du avgöra om en Player ger korrekta resultat vid anrop till spelarens throwDice-metod.

Du kan använda samma lösningsstrategi i denna uppgift som i uppgift 5c men det är inte helt enkelt att ta reda på hur många sidor spelarens tärning har. För att ta reda på detta kan du:

1. kontrollera om Player-objektet som levereras är en OrdinaryPlayer eller en Cheater (med instanceof)
2. typkonvertera Player-objektet till referens till OrdinaryPlayer eller Cheater
3. erhålla referens till tärningsobjektet genom att anropa metoden getDice.
4. använda Dice-referensen till anrop av getSides-metoden.

Nu återstår ungefär samma jobb som gjordes i uppgift 5c. Men det är Player-objektet som ska kasta tärningen.

Om du kör nedanstående program kan du få ett körresultat som liknar det längre ner på sidan.

Testprogram

```
TestDice.test( new OrdinaryPlayer( "Rut", new SimpleDice( 6 ) ), 1000000 );
TextWindow.println();
TestDice.test( new Cheater( "Fuffe", new SimpleDice( 6 ) ), 1000000 );
```

Körresultat

```
1    167391
2    166523
3    166336
4    166804
5    166427
6    166519

1     83545
2    166432
3    166320
4    166893
5    166647
6    250163
```

Uppgift 5f

Javadoc-kommentera klasserna SimpleDice, OrdinaryPlayer och Cheater. Gererera sedan javadoc-dokumentation för paketet p5.

Vardera klass ska innehålla följande javadoc-kommentarer:

- Kommentar om klassen, inklusive @author-tagg.
- Kommentar om varje konstruktor, inklusive @param-tagg för alla parametrar.
- Kommentar om varje metod, inklusive @param-tagg för varje parameter och @return-tagg då metoden har returvärde.

Uppgift 5g (extra)

Nu ska du tillverka en fusktärning, **ProbabilityDice**, vilken kan ge helt olika utfall för olika antal prickar. ProbabilityDice ska **implementera Dice**. Detta kan ske genom en direkt implementation eller genom arv av SimpleDice.

- En ProbabilityDice ska ha en konstruktor med en int-array som parameter. int-arrayen beskriver sannolikheten att olika antal prickar ska inträffa. Ett par exempel:

  ```
  int[] prob1 = {25,20,35,20};
  ProbabilityDice dice1 = new ProbabilityDice(prob1);
  ```

  ger en tärning som i 25 % av kasten ger en 1:a, 20 % av kasten ger en 2:a, 35 % av kasten ger en 3:a och i 20 % av kasten ger en 4:a.

  ```
  int[] prob2 = {20,0,0,0,30,0,50};
  ProbabilityDice dice2 = new ProbabilityDice(prob2);
  ```

  ger en tärning som i 20 % av kasten ger en 1:a, aldrig ger en 2:a, 3:a eller 4:a, 30 % av kasten ger en 5:a, aldrig ger en 6:a och i 50 % av kasten ger en 7:a.

- Summan av sannolikheterna ska alltid vara 100. Om så inte är fallet ska ett **BadProbabilityException** kastas. Det innebär att du även måste skriva klassen BadProbabilityException.

Om du kör **TestProbabilityDice.java** kan du erhålla ett körresultat liknande detta:

```
{20,10,10,10,10,40}
----- 10 kast med 6-sidig tärning -----
1            2
2            1
3            2
4            0
5            1
6            4

{0,25,0,25,0,25,0,25,0}
----- 1000000 kast med 9-sidig tärning -----
1            0
2       250120
3            0
4       249677
5            0
6       249953
7            0
8       250250
9            0

{40,30,20,10}
----- 1000000 kast med 4-sidig tärning -----
1       398997
2       300505
3       200156
4       100342

p5.BadProbabilityException: Total sannolikhet måste vara 100: 60
p5.BadProbabilityException: Antal sidor = 0
```

Nu kan du testa det enkla spelet med t.ex. två fuskare varvid den ena har en fusktärning:

```
int[] prob = {5,15,20,20,20,20};
Player spelare1 = new Cheater( "Viktor", new SimpleDice( 6 ) );
Player spelare2 = new Cheater( "Signe", new ProbabilityDice( prob ) );
Game spel = new Game( spelare1, spelare2 );
TextWindow.println( "\nResultatet av tio spel" );
for( int i = 0; i < 10; i++ ) {
    spel.play( false );
```