VOP F15 ReEksamen 17. august 2015

Tilpasset til Eksamenstræning 2019, lektion 12.

Eksamensopgaver

Dette opgavesæt indeholder 4 programmeringsopgaver, hvor opgaverne 1, 2 vægtes med hver 25% , opgave 3 med 30% og opgave 4 med 20%.

Opgave 1 og 2 integreres med javaFX GUI; men del-løsninger kan testes ved hjælp af `main()`-metoder.

Opgaverne 3 og 4, løses og testes uafhængigt af javaFX brugerfladen, vha. `main()`-metoder.

Vigtigt: Der kan være problemer med at eksekvere klasser med en `main()`-metode under et JavaFx projekt. Brug *Shift-F11* eller højre klik på projektet og vælg *Clean and Build*, inden *run* kaldes direkte på en Java main klasse.

Hint: Giv dig tid til at gennemlæse hele opgavesættet, inden du går i gang med løsningerne. Du bør dog starte med at opsætte projektet, som beskrevet i Opg. 1.

Aflevering: Ved afslutningen af eksamen skal løsningen afleveres under *Assignments* på BlackBoard:

- I NetBeans markeres projektet, som indeholder løsningen
- Vælg menuen *File -> Export Projekt -> To ZIP…*
- På den fremkomne pop-up dialog, vælges et passende sted at gemme projektet i feltet *Build ZIP:*
- Sørg for at filen får det rigtige navn. Fx ”abcd14VOPReEksamen.zip”
- Upload filen

God fornøjelse.

Opg. 1 JavaFX-projekt med benyttelse af Facadeklasse 25%

a) Oprettelse af projekt

Formål: At oprette eksamensprojektet som en JavaFX applikation, navngivet korrekt og forberedt til de øvrige eksamensopgaver.

1. Opret et projekt i NetBeans af typen *JavaFX FXML Application*:
    a. *Project Name* skal begynde med jeres SDU-brugernavn, fx " abcd14VOPReEksamen".
    b. Marker checkboksen *Create Application Main Class.*
2. Start *SceneBuilderen* ved at dobbeltklikke på den dannede *FXML*-fil og slet den `Button` og `Label` som er dannet til *Hello World* eksemplet.
3. Det dannede `Anchorpane` kan med fordel gøres større , fx ved at sætte *Pref Width = 500* og *Pref Height = 325* under *Layout*, eller ved at trække i et af hjørnerne med musen.
4. Sæt et `TabPane` på brugerfladen og tilpas dets størrelse til *Fit to Parent*
5. Pak den udleverede zip-fil ud og kopier de 4 mapper til projektets *src*-mappe.
6. Kopier `tourdefrance.txt` til projekt-mappen (udenfor src-mappen).

b) Facade-klasse med 3 metoder til tjek af heltal

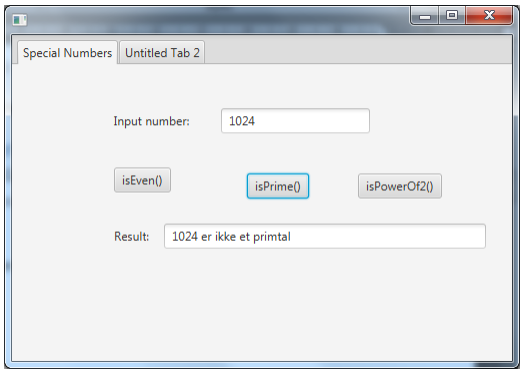
Pakken `opg1_facade` indeholder en ufærdig main-klasse `SpecialNumbers.java`.
De 3 metoder til tjek af heltal, som er erklæret i klassen, skal implementeres:

- `boolean isEven(int x)` skal returnere `true` hvis x er et lige tal
- `boolean isPrime(int x)` skal returnere `true` hvis x er et primtal
  Hint: Find `int y = (√x +1)`. *Hvis der findes et tal* `i > 1` *og* `i < y`, *hvor* `x % i == 0` *er* x *ikke et primtal.*
- `boolean isSquare(int x)` skal returnere `true` hvis x et kvadrattal
  Hint: Find `double y = √x`. *Hvis* `y == (int) y` *er* `x` *kvadrattal.*

De 3 metoder kan tjekkes ved at fjerne udkommenteringerne i klassens `main()`-metode.

c) Integration med javaFX brugerfladen

Klassen `SpecialNumbers` skal benyttes som *Facade Controller* af javaFx brugerfladen. Der skal derfor erklæres en instance variabel af typen i `JavaFxController`-klassen. Initialiser variablen i `initialize()`-metoden.



Opbyg brugerfladen i SceneBuilderen så den indeholder:

- Label og TextFields til input
- Label og *ikke*-editerbart TextField til resultat
- 3 Buttons til kald af de 3 metoder i Facaden.

Erklær en instance variabel af `SpecialNumbers` i javaFx-controlleren og intialiser den i `initialize()`-metoden.

Kald de 3 metoder fra ActionHandlerne på knapperne og skriv resultatet i Resultat-feltet.

Opg. 2 Re-design så Facaden benytter polymorphi 25%

I denne opgave skal de 3 metoder fra opgave 1 gentages, så de kan benyttes vha polymorfi.
Pakken `opg2_polymorphi` indeholder allerede:

- `CheckerInterface.java`
  Interface som erklærer en enkelt metode `boolean check(int x);`
- `NumberCheckerFacade.java` med den ufærdige metode:
  `public List<Integer> checkNumbers(int min, int max, CheckerInterface checker);`
  samt en `main()`-metode til test.

a) Implementationer af `CheckerInterface`

Der skal programmeres 3 implementationer af `CheckerInterface.java`:

- `EvenChecker.java`: `check()`-metode skal returnere `true` hvis input parameteren `x` er et heltal.
- `PrimeChecker.java`: `check()`-metode skal returnere `true` hvis input parameteren `x` er et primtal.
- `SquereChecker.java`: `check()`-metode skal returnere `true` hvis input parameteren `x er` et kvadrattal.

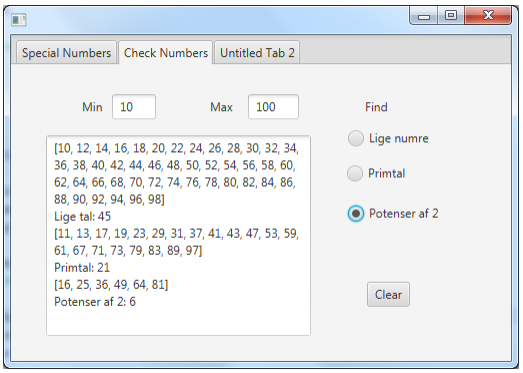
b) Implementation af `checkNumbers()`-metoden

Implementer `checkNumbers(int min, int max, CheckerInterface checker)`i `NumberCheckerFacade`, så alle heltal i intervallet `[min..max[` undersøges med den givne implementation af `check()`-metoden. Returner en `LinkedList<Integer>` indeholdende alle de tal hvor `check()` har returneret `true`.

c) Integration med javaFX brugerfladen

Der skal udvikles en brugerflade til test af ovenstående indeholdende:

- 3 `Labels` med teksterne ”Min”, ”Max” og ”Find”.
- 2 `TextFields` til indtastning af minimum og maximum.
- 3 `RadioButtons` til valg af `check()`-metode (husk en fælles *toggle group*). Erklær en fælles `actionHandler` til de tre radiobuttons.
- 1 `TextArea` til skrivning af resultater (*ikke-editerbart* og med *Wrap Text*).
- 1 `Button` til sletning af tekstarealet og inputfelterne.



Erklær en instance variabel af `NumberCheckerFacade` i controlleren og intialiser den i `initialize()`-metoden.

ActionHandleren på de 3 radiobuttons skal erklære en instance af `CheckerInterface` og initialisere den til den implementation, som svarer til den valgte radiobutton. Herefter kaldes Facadens `checkNumbers()`-metode med de indtastede værdier og `CheckerInterface`-implementation. Listen, som returneres udskrives i textarealet efterfulgt af antallet af elementer i listen.

Opg. 3 Resultater fra årets Tour de France 30%

NB: Denne opgave løses uden javaFX-brugerflade.

I denne opgave skal der arbejdes med nogle af resultaterne fra det netop overståede tour-de-France.

Filen `tourdefrance.txt` indeholder de gennemførende rytteres resultater i de 3 konkurencer *Klassementet, Bjerge* og *Pointspurter*. Filen viser én rytter pr. linje med oplysningerne adskilt af et tabulator-tegn ("\t"):

Navn	*Hold*	*Nationalitet*	*Tidstab**	*Bjergpoints*	*Spurtpoints*
A.CONTADOR	TCS	ESP	00:09:48	2	45

*Tidstabet er angivet i fht. den vindende rytter. Fx betyder ovenstående linje at *Contador* tilhører *TCS*-holdet, er *spanier*, har brugt *9min 48sek* mere end vinderen, fik *2* point i bjergkonkurencen og *45* point i spurterne.

a) Implementation af `RacingCyclist.java`

Af det udleverede skelet i pakken `opg3_tour`, ses at klassen skal implementere interfacet `Comparable` og indeholder 6 instance variable, samt en `toString()`-metode. Én instance af klassen repræsenterer en enkelt cykelrytter.

Implementer klassen så:

- Der tilføjes en constructor som tager 6 parametre svarende til de 6 variable og tildeler disse værdier.
- Tilføj `get()`-metoder til de 6 variable.
- Implementer `compareTo()`-metoden så cykelrytterne kan sorteres stigende efter deres *tidstab*. Hvis flere ryttere har samme tidstab, skal de sorteres efter *navn*.
  Hint: Tidstabet er angivet som en String på formen hh:mm:ss og kan derfor sammenlignes leksikalt.

b) Implementation af `TourDeFranceMain.java`

Opret en klasse af typen *Java Main Class* ved navn `TourDeFranceMain.java`

Klassen skal indeholde:

- En privat variabel af typen `java.util.List<RacingCyclist>`
- En privat variabel af typen `java.io.File`
- En constructor med signaturen `public TourDeFranceMain(String fileName).`
  I constructoren skal filen initialiseres med `fileName` og listen som en `ArrayList`.
- En *getter*-metode med signaturen `public List<RacingCyclist> getList()`, som returnerer listen.
- En metode med signaturen `public void readFile()`. Her skal filen læses én linje ad gangen, så der kan dannes instancer af `RacingCyclist`, som tilføjes listen. Eventuelle `exceptions` skal behandles inde i metoden og det skal sikres at filen lukkes efter gennemlæsningen.

Til test kan disse 3 linjer indsættes i klassens main()-metode (udskriver de første 10 elementer):

```
TourDeFranceMain tfm = new TourDeFranceMain("tourdefrance.txt");
tfm.readFile();
System.out.println("List:\n" + tfm.getList().subList(0, 10));
```

Det burde give et resultat som ligner :

```
List:
```

```
[A.CONTADOR TCS ESP 00:09:48 2 45
, A.DELAPLACE BSE FRA 03:11:28 1 15
, A.DEMARE FDJ FRA 04:05:28 0 43
, A.FONSECA BSE FRA 03:53:13 0 18
, A.GENIEZ FDJ FRA 03:42:57 27 31
, A.GERARD BSE FRA 04:02:06 0 0
, A.GREIPEL LTS ALL 04:03:28 0 366
, A.GRIVKO AST UKR 02:38:06 0 51
, A.HANSEN LTS AUS 03:45:18 0 2
, A.KRISTOFF KAT NOR 04:01:06 0 90
]
```

c) Sortering med `Comparable`

Tilføj en sorterings-metode til `TourDeFranceMain` med signaturen `public void sort()`. Benyt en af standard metoderne i `java.util.Collections` til sortering af listen ved brug af `compareTo()`-metoden. Test ved at tilføje disse linjer til main:

```
tfm.sort();
System.out.println("Sort:\n" + tfm.getList().subList(0, 10));
```

Det burde give et resultat som ligner (de 10 første i klassementet):

```
Sort:
[C.FROOME SKY GBR 00:00:00 119 139
, N.QUINTANA MOV COL 00:01:12 108 80
, A.VALVERDE MOV ESP 00:05:25 72 103
, V.NIBALI AST ITA 00:08:36 53 49
, A.CONTADOR TCS ESP 00:09:48 2 45
, R.GESINK TLJ HOL 00:10:47 32 53
, B.MOLLEMA TFR HOL 00:15:14 0 38
, M.FRANK  IAM SUI 00:15:39 4 24
, R.BARDET ALM FRA 00:16:00 90 74
, P.ROLLAND EUC FRA 00:17:30 74 73
]
```

d) Sorteret `Set` med brug af en `Comparator`

Opret en klasse med signaturen

```
public class CountryMountainComparator implements Comparator<RacingCyclist>
```

Implementer metoden `public int compare(RacingCyclist o1, RacingCyclist o2)`så objekter af typen `RacingCyclist` kan sorteres efter *Nationalitet*. Hvis to ryttere kommer fra samme land, skal der sorteres faldende på *Bjerg*-points.

Tilføje en metode til `TourDeFranceMain` med signaturen:

```
public Set<RacingCyclist> makeSortedSet(Comparator comp)
```

I metoden skal der oprettes et *sorteret* `Set<RacingCyclist>`, hvortil alle elementerne i listen tilføjes ved brug af `comp`.

Test kan udføres ved at tilføje

```
Comparator<RacingCyclist> comp = new CountryMountainComparator();
Set<RacingCyclist> countryMountainSet = tfm.makeSortedSet(comp);
System.out.println("Country/Mountain:\n" + countryMountainSet);
```

Opg. 4 Spillekort 20%

Eksamenstræning 2019: Denne opgave arbejdede i med under øvelsestimerne til Lektion 4. Det anbefales at udvikle Unit test til de to klasser Card og DeckOfCards.

NB: Denne opgave løses uden javaFX-brugerflade.

Udleveret kode: `CardInterface.java`
`Card.java`
`DeckOfCards.java`

I denne opgave skal der implementeres et spil kort. Programmet består af tre klasser:

- `CardInterface.java:` Et interface hvori der er erklæret `int`-konstanter for de fire kulører (*suit* på engelsk), SPADES = 4 (Spar), HEARTS = 3 (Hjerter), DIAMONDS = 2 (Ruder) og CLUBS = 1 (Klør), samt for værdierne ACE = 1 (Es), JACK = 11 (Knægt), QUEEN = 12 (Dame) og KING = 13 (Konge), idet det antages at ACE altid er det mindste kort i en given kulør.
  Desuden er der defineret `String`-konstanter, indeholdende danske betegnelser for de 4 kulører, billedkort og es. Disse kan med fordel benyttes til udskrift. Husk at konstanter blot er synonymer for værdier. Der er ikke erklæret metoder i interfacet.
- `Card.java` skal repræsenterer ét enkelt kort. En instance af Card har to `int`-felter, som definerer hhv. `face` (værdi) og `suit` (kulør).
- `DeckOfCards.java` repræsenterer de 52 kort, som et spil kort uden jokere består af.

a) Klassen Card.java

Implementer en klassen `Card implements CardInterface`. De to felter `face` og `suit` skal tildeles værdier fra parametre til constructoren. Hvis ikke parametrene er lovlige (dvs. `face < ACE`, `face > KING`, `suit < CLUBS` eller `suit > SPADES`), skal der udskrives en fejlmeddelelse.

b) Udskrivning af et kort

Implementer `public String toString()`, så der returneres en tekst med kortes `suit` og `face`. Teksten skal dannes med brug af `String`-konstanterne i interfacet.

Til test kan følgende `main()`-metode implementeres i `Card.java`. Ønsket output ses i boksen:

```
public static void main(String[] args) {
    System.out.println(new Card(1, 3));
    System.out.println(new Card(3, 1));
    System.out.println(new Card(13, 4));
    System.out.println(new Card(12, 3));
    System.out.println(new Card(17, 5));
}
```

```
Hjerter-Es
Kloer-3
Spar-Ko
Hjerter-Da
Ulovligt kort: 5 17
```

c) 52 spillekort

`DeckOfCards.java` implementerer også `CardInterface`, så konstantværdierne kendes.
Der er erklæret et `int`-array, en `main()`-metode til test, samt en `toString()`-metode, som udskriver Card-arrayet med 4 kort pr linje.

I constructoren skal int-arrayet initialiseres til størrelsen NUMBER_OF_CARDS og der skal dannes instanser af de 52 mulige kort, som skal indeholdes i arrayet. Benyt nogle af konstanterne fra interfacet til at styre de løkker der udfylder arrayet.

d) Blande kort

Implementer `public void shuffle(int swaps)`, så kortene kan blandes. Parameteren `swaps` angiver hvor mange gange to kort i bunken skal byttes om.

Hint: Benyt en instance af `java.util.Random` *til at vælge to pladser i arrayet, og byt kortene om. Dette gøres* `swaps` *gange.*

Herunder ses et output, som det kan se ud efter opgave c) og d) er løst:

```
Opg 4c:
Kloer Es, Ruder Es, Hjerter Es, Spar Es,
Kloer 2, Ruder 2, Hjerter 2, Spar 2,
Kloer 3, Ruder 3, Hjerter 3, Spar 3,
Kloer 4, Ruder 4, Hjerter 4, Spar 4,
Kloer 5, Ruder 5, Hjerter 5, Spar 5,
Kloer 6, Ruder 6, Hjerter 6, Spar 6,
Kloer 7, Ruder 7, Hjerter 7, Spar 7,
Kloer 8, Ruder 8, Hjerter 8, Spar 8,
Kloer 9, Ruder 9, Hjerter 9, Spar 9,
Kloer 10, Ruder 10, Hjerter 10, Spar 10,
Kloer Kn, Ruder Kn, Hjerter Kn, Spar Kn,
Kloer Da, Ruder Da, Hjerter Da, Spar Da,
Kloer Ko, Ruder Ko, Hjerter Ko, Spar Ko
```

```
Opg 4d:
Hjerter 8, Ruder Da, Kloer 9, Hjerter 10,
Hjerter 3, Hjerter 7, Ruder 6, Spar Da,
Ruder 10, Spar Es, Hjerter Da, Hjerter 5,
Kloer 3, Kloer Es, Hjerter Ko, Hjerter 9,
Spar 4, Spar 7, Spar 9, Kloer 7,
Spar Kn, Ruder Ko, Hjerter 6, Kloer 5,
Ruder Es, Kloer 6, Spar 6, Hjerter Es,
Spar 3, Ruder 9, Spar Ko, Ruder 5,
Ruder 8, Spar 5, Ruder 7, Hjerter 2,
Kloer Kn, Spar 2, Hjerter Kn, Kloer Da,
Spar 8, Kloer 2, Kloer 4, Spar 10,
Ruder Kn, Hjerter 4, Kloer Ko, Kloer 8,
Kloer 10, Ruder 4, Ruder 2, Ruder 3
```