TENTAMEN: Objektorienterade applikationer

Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad.
- Skriv ditt idnummer på varje blad (så att vi inte slarvar bort dem).
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programmen skall skrivas i Java 5, eller senare version, vara indenterade, renskrivna och i övrigt vara utformade enligt de principer som lärts ut i kursen.
- Onödigt komplicerade lösningar ger poängavdrag.
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Givna deklarationer, parameterlistor, etc. får ej ändras, såvida inte annat sägs i uppgiften.
- Läs igenom tentamenstesen och förbered ev. frågor.

I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa.

Lycka till!

Uppgift 1

a) Rita ett UML-diagram som motsvarar följande Java-kod:

```
public interface Squeezeable { ... }
public class Sponge implements Squeezeable { ... }
public class Jellyfish implements Squeezeable { ... }
public class Snow implements Squeezeable { ... }
public class Hand {
    private Squeezeable sq;
    public Hand(Squeezeable x) { sq = x; }
}
public class Human {
    private Hand left,right;
    public Human(Squeezeable x) {
        left = new Hand(x);
        right = new Hand(x);
    }
}
```

(5 p)

b) Rita ett klassdiagram som beskriver designmönstret Decorator.

(5 p)

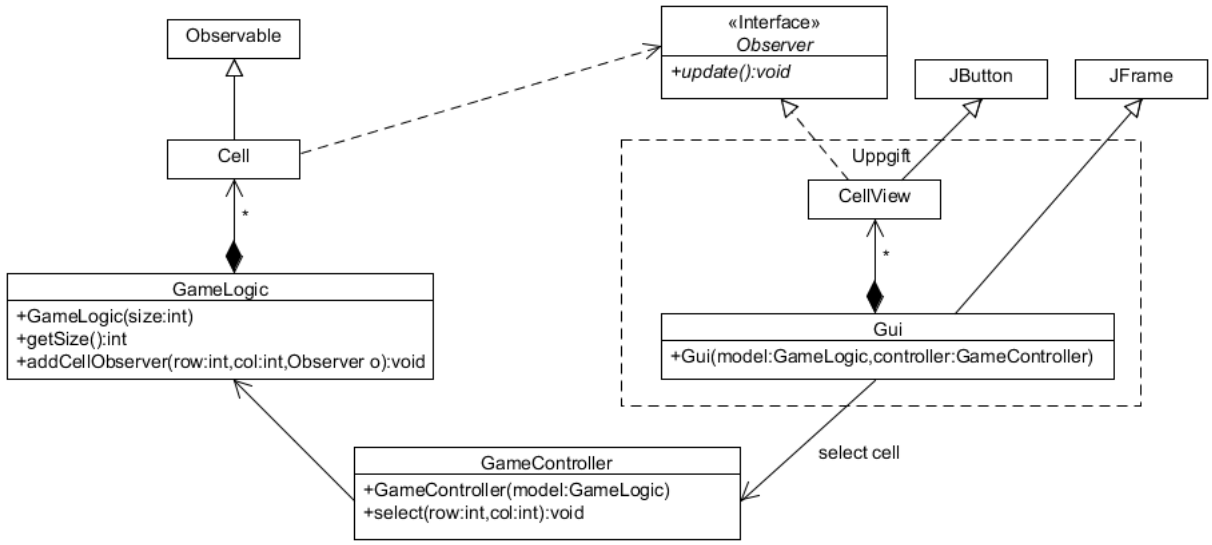
.

Skriv ej här

Uppgift 2

Spelet tre i rad (tic-tac-toe) spelas av två personer på en kvadratisk spelplan med 3 x 3 rutor. Spelarna turas om att sätta ett kryss (X) resp. en ring (O) i en ledig ruta. Den spelare som först lyckas få tre markeringar i rad, vågrätt, lodrätt eller diagonalt vinner.

Vi vill förstås ha en datorbaserad version av spelet. Det finns en påbörjad implementering av följande design, och uppgiften är bl.a. att konstruera GUI-klasserna inom den streckade rutan:



Färdiga klasser

Klasserna `Cell`, `GameLogic` och `GameController` är färdiga. Klasserna beskrivs kortfattat nedan. Endast de konstruktorer och metoder som behövs för att lösa uppgiften är medtagna.

`public class Cell extends Observable` (färdig)

Motsvarar en ruta på spelplanen. Håller reda på om rutan skall innehålla ett kryss, en ring eller vara tom. När objektet ändras uppdateras en observatör med en av tre strängar: `""`, `"X"` eller `"O"`.

`public class GameLogic` (färdig)

Lagrar information om spelplanen. Har ett antal `Cell`-objekt. Klassen kan även räkna ut om någon spelare vunnit.

Kontruktor

`public` `GameLogic(`**`int`** `size)`

`size` anger spelets dimension (3 i exemplet ovan).

Metoder

`public int` `getSize()`

returnerar spelets dimension.

`public void` `addCellObserver(`**`int`** `row,`**`int`** `col,Observer obs)`

adderar `obs` som observatör på `Cell`-objektet i position `row`, `col`.

`public class` `GameController` (färdig)

Styr spelets kontrollflöde och turordningen mellan spelarna. Samarbetar med `GameLogic` för att registrera spelarnas drag, kontrollera om någon vunnit, samt skapa ny spelomgång.

Kontruktor

```
public GameController(GameLogic model)
```

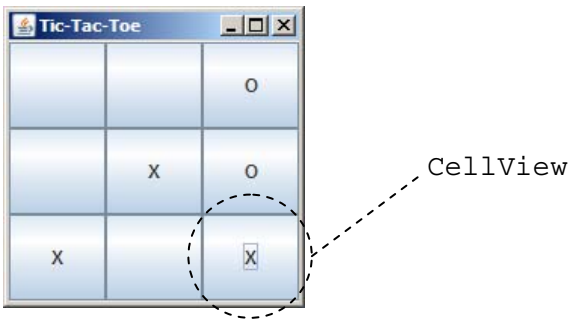
Metoder

```
public void select(int row,int col)
```

Rapporterar ett kryss eller ring för position `row`, `col` till modellen, beroende på vilken spelares tur det är.

Uppgifter

Spelets fönster skall se ut som följande exempel visar, här ett 3 x 3-spel efter fem drag:



Observera att klassdiagrammet ovan kan ge värdefull information som inte nämns här.

a) Implementera klassen `CellView`. Objekt av klassen skall kunna vara komponenter i ett fönster och visa en text. Klassen skall vara observatör enligt designmönstret Observer .

(3 p)

b) Implementera klassen `Gui`.

Kontruktor

```
public Gui(GameLogic model,GameController controller)
```

Visar ett fönster enligt exemplet ovan. Samarbetar med `GameLogic` och `GameController`. Lagrar ett antal `CellView`-komponenter. När användaren trycker på en sådan komponent skall positionen rapporteras till `GameController`.

(10 p)

c) Skriv en `main`-metod som skapar och sätter ihop objekt av `GameLogic`, `GameController` samt `Gui` till ett fungerande program. När `main` exekveras skall alltså fönstret visas.

(3 p)

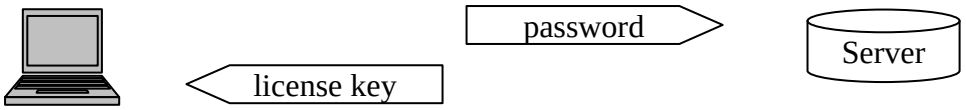
Uppgift 3

Information om licensnycklar för en viss programvara lagras på en server i en textfil med kolonseparerade rader av följande format:

IP-adress:*lösenord*:*licensnyckel*

Exempel: `139.4.2.7:qwerty:DONT-TRYT-OCRA-CKME-YOUF-OOLI-SHHA-CKER`

En licensnyckel gäller för en programinstallation på den dator som anges av IP-adressen. Om man glömt sin nyckel skall man kunna få den skickad till den dator som nyckeln gäller för.



Man begär att få en nyckel genom att skicka ett datagram innehållande lösenordet till servern. Servern kontrollerar om det inkommande datagrammets avsändaradress och lösenord finns i licensnyckelfilen. Om så är fallet returneras ett datagram innehållande den matchande licensnyckeln till avsändaren. I annat fall returneras ett datagram innehållande texten ”Authorisation failure”. Avsändarens adress och port kan erhållas ur datagrammet med metodanrop.

a)

Implementera klassen `Database` som hanterar textfilen med licensnycklarna. Klassens konstruktorn skall ta namnet på licensnyckelfilen som inparameter. Dessutom skall den ha metoden

```
public String lookup(String host,String pwd) throws IOException
```

`lookup` skall söka i filen efter rader som matchar `host` och `pwd`. Om sökningen lyckas returneras det matchande licensnumret, annars **`null`**.

Ex. `lookup(”139.4.2.7”,”qwerty”)` returnerar
`”DONT-TRYT-OCRA-CKME-YOUF-OOLI-SHHA-CKER”.`

Tips: Använd `split` för att analysera textraderna.

(4 p)

b)

Implementera klassen `Server`. Klassen skall ha en konstruktor

```
public Server(int port,Database db)
```

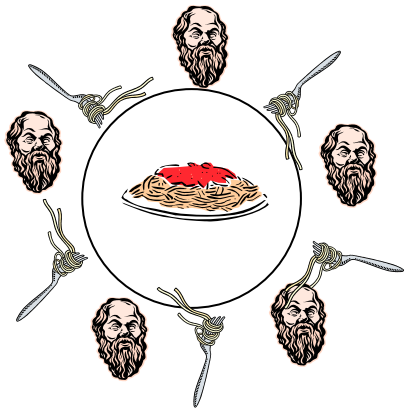
som tar portnumret för inkomande datagram, samt licensdatabasen som inparametrar.

Det skall också finnas en metod som startar servern. Metoden skall oupphörligt vara beredd på att ta emot datagram med lösenord, vilket beskrevs ovan. Sökning skall göras i databasen, samt lämpligt svar i form av ett datagram skickas till avsändaren. Avsändarens adress och portnummer kan fås med lämpliga metodanrop, se API.

(10 p)

Uppgift 4

Ett klassiskt problem i datalogi är följande: Fem filosofer sitter kring ett runt bord. Mellan varje filosof ligger en gaffel. Mitt på bordet står en tallrik spagetti. För att få äta måste en filosof ta upp både gaffeln till vänster och gaffeln till höger. Gafflarna läggs därefter tillbaka på sina platser. Om vi betraktar filosoferna som processer blir problemet att visa att alla blir mätta, d.v.s. att schemaläggningen av processerna sker rättvist. (ursäkta grafiken)



Eftersom det här är en tillämpad utbildning ersätter vi filosoferna med programmerare. Uppgiften är att konstruera en simulator för problemet ovan. Programmerare och gafflar modelleras som objekt. Ett gaffelobjekt får bara användas av ett programmerarobjekt i taget.

a)
Konstruera gaffelklassen `Fork`. Klassen skall ha två `void`-metoder: `grab()` och `drop()`. `grab` anropas av en tråd som vill äga ett gaffelobjekt, och `drop` för att lämna tillbaka det. Om flera trådar anropar `grab` för samma objekt skall bara en i taget ges äganderätt, de andra får vänta på sin tur. Utnyttja Javas mekanismer för synkronisering på lämpligt sätt. (4 p)

b)
Konstruera klassen `Programmer`. Varje objekt av klassen skall exekvera i en egen tråd. En konstruktor skall ta två gaffelobjekt samt programmerarens namn som inparametrar. Tråden skall utföra en oändlig cykel av: tag gafflar, ät, lämna tillbaka gafflar. ”Ätandet” motsvaras av en utskriftsrad enligt exemplet nedan. Talet inom parenteser är antalet gånger processen ätit. (4 p)

c)
Konstruera klassen `Simulator`. Klassen skall skapa en ring av fem programmerartrådar med mellanliggande gaffelobjekt (enligt figuren ovan), samt starta trådarna. Antalet programmerare (och gafflar) skall hanteras generellt så att man enkelt kan variera antalet. (4 p)

```
...
Programmer 4 eats (22179)
Programmer 1 eats (22117)
Programmer 2 eats (22152)
Programmer 3 eats (22161)
Programmer 1 eats (22118)
Programmer 0 eats (22175)
Programmer 2 eats (22153)
...
```

Här framgår att schemaläggningen av trådarna varit någorlunda rättvis.

Uppgift 5

Vi har ju lärt oss att instansvariabler bör deklareras som privata i klasser för att skydda variablerna från felaktig användning. Vill man bara läsa av en variabel kan man deklarera en accessmetod. Men är det så enkelt? Vad händer om instansvariabeln innehåller en objektreferens till exempelvis en lista? Studera följande klass:

```
public class MyClass {
    public ArrayList<String> names;
    private ArrayList<Boolean> truths;
    private ArrayList<Integer> numbers;

    public ArrayList<Boolean> getTruths() { return truths; }

    public Iterator<Integer> getNumbers() {
        return numbers.iterator();
    }
}
```

En användare av klassen har obegränsad access till `names`. Accessmetoden `getTruths` ger access till den andra listan och möjliggör såväl tillägg av nya element som borttagning av element. Det går dock inte att ta bort eller byta ut själva listan. Ett tredje sätt är att returnera en *iterator* till listan vilket görs i den sista metoden. Det kan t.ex. utnyttjas så här:

```
MyClass obj = new MyClass();
Iterator<Integer> it = obj.getNumbers();
while ( it.hasNext() ) {
    int i = it.next();
    if ( i % 2 == 1 )
        it.remove();
}
```

Just det, iteratorn har en `remove`-metod med vilken listan kan muteras! Vad göra?

Uppgiften är att konstruera ”filterklassen” `ReadOnlyIterator` som givet en vanlig iterator ger en iterator utan en fungerande `remove`-metod. Metoden `getNumbers` kan då istället skrivas

```
public Iterator<Integer> getNumbers() {
    return new ReadOnlyIterator<Integer>(numbers.iterator());
}
```

Om nu `remove` anropas, som ovan, skall undantaget `UnsupportedOperationException` kastas. Om argumentet till konstruktorn är `null` skall undantaget `IllegalArgumentException` kastas. *Tips*: Av ovanstående kod kan du dra vissa slutsatser om vilka typer som är inblandade. Gränssnittet `Iterator` finns i API.

(8 p)