TENTAMEN: Objektorienterade applikationer

Läs detta!

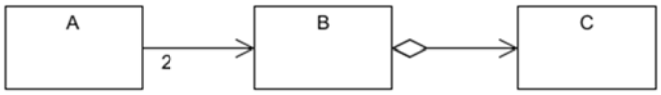
- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad.
- Skriv ditt idnummer på varje blad (så att vi inte slarvar bort dem).
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programmen skall skrivas i Java 5, eller senare version, vara indenterade, 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) Skriv Java-kod som definierar klasser och skapar objekt motsvararande UML-diagrammet



(3 p)

b) I ett enkelt hierarkiskt filsystem finns filtyperna textfil och katalog. En katalog kan innehålla noll eller flera filer. Ett känt designmönster som behandlats i kursen beskriver sådana strukturer. I en objektorienterad modell för filsystemet kan man modellera filtyperna med klasserna Fil, Textfil och Katalog. Rita ett klassdiagram som beskriver designmönstret tillämpat på dessa klasser. Diagrammet skall ha mönstrets namn som rubrik.

(3 p)

Uppgift 2

En `CounterIterator` fungerar precis som en vanlig iterator med skillnaden att den dessutom kan tala om hur många element som returnerats av `next`. Exempel:

```
Collection<Integer> coll = new LinkedList<>();
for ( int i = 3;i >= 0;i-- )
    coll.add(i);

CounterIterator<Integer> it = new CounterIterator<>(coll.iterator());
System.out.println("count=" + it.getCount());
while ( it.hasNext() ) {
    int x = it.next();
    System.out.println("x=" + x + ",count=" + it.getCount());
}
```

Utskriften blir:

```
count=0
x=3,count=1
x=2,count=2
x=1,count=3
x=0,count=4
```

Konstruera klassen `CounterIterator` enligt designmönstret *decorator*. Om argumentet till konstruktorn är `null` skall undantaget `IllegalArgumentException` kastas. Gränssnittet `Iterator` finns i API. Om du tycker att det blir enklare får du definiera `CounterIterator` utan typparameter, dvs anta att iteratorn alltid hanterar en samling av heltal.

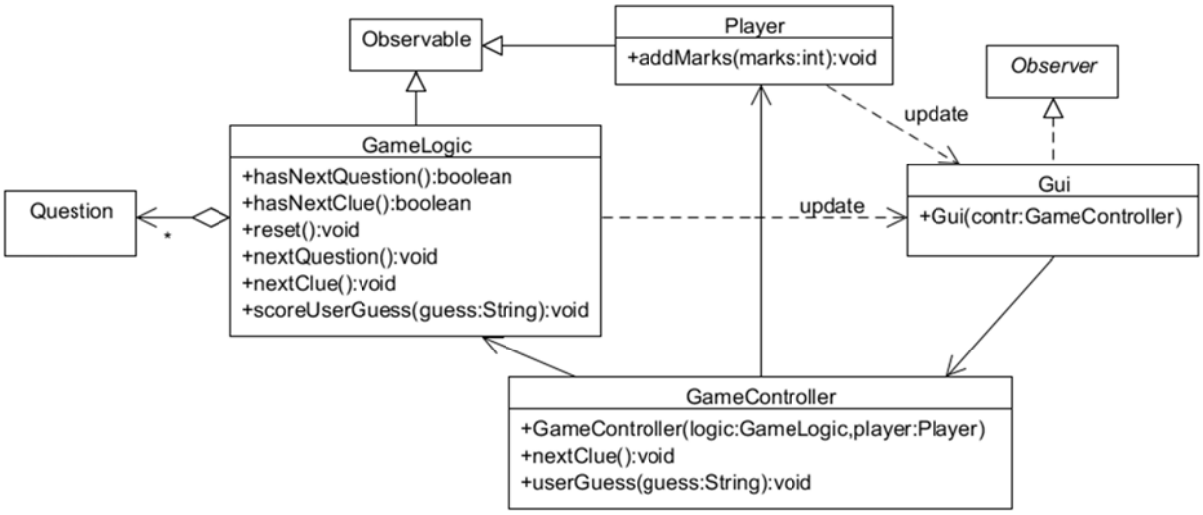
(8 p)

Uppgift 3

I spelet ”på håret” får man poäng genom att besvara frågor. För varje fråga visas tips på avtagande poängnivåer ner till 1 poäng. När man besvarat en fråga visas nästa fråga. Svarade man rätt adderas dessutom tipsets poäng till totalpoängen. Förbrukar man alla tips visas nästa fråga.

Ex. I denna variant finns tre olika frågor med fem tipsnivåer vardera. Ett scenario åt höger, och ett nedåt.

Det finns en påbörjad implementering av designen nedan, och uppgiften är att komplettera koden med MVC-struktur.



Färdiga klasser

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

`public class Question` (färdig)

Lagrar en fråga, rätt svar samt ett antal tips. Klassen används internt av `GameLogic`.

`public class GameLogic extends Observable` (färdig)

Lagrar frågorna i ett antal `Question`-objekt. Klassen håller reda på vilken fråga som är aktuell, vilket tips som är aktuellt, samt tipsets poängnivå.

Kontruktor

`public` `GameLogic()`

Laddar frågorna från en fil.

Metoder

`public void` `reset()`

Återgår till första frågan.

`public boolean` `hasNextQuestion()`

Returnerar **`true`** om det finns fler frågor, **`false`** annars.

`public boolean` `hasNextClue()`

Returnerar **`true`** om det finns fler tips för aktuell fråga, **`false`** annars.

`public void` `nextQuestion()`

Avancerar till nästa fråga.

`public void` `nextClue()`

Avancerar till nästa tips för aktuell fråga.

`public int` `scoreUserGuess(String guess)`

Returnerar aktuell tipspoäng om `guess` innehåller rätt svar på frågan, annars 0.

Samtliga metoder utom de två accessmetoderna uppdaterar klassens observatörer. För `reset`, `nextQuestion` och `nextClue` skickas en kolonseparerad sträng på formen

<question>:<clue>:<level>

Exempel: `"En stad:Holländare byggde staden:4"`

`public class Player extends Observable` (färdig)

Lagrar spelarens poängsumma.

Metoder

`public void addMarks(int marks)`

Adderar `marks` till spelarens poängsumma. Uppdaterar klassens observatörer med poängsumman i form av ett heltal (`int`).

Uppgifter

Uppgiften är att konstruera kontrollklassen, komplettera Gui:t, samt skriva en `main`-metod som skapar och kopplar ihop objekt av lämpliga typer så att ett fungerande program erhålls.

a) Implementera klassen `GameController`. Inför två metoder som kan anropas från Gui:t när användaren tryckt på knappen för nästa tips, respektive skrivit en gissning i textfältet längst ner i fönstret. `GameController` skall förmedla lämplig information till klasserna som den samarbetar med samt styra spelets kontrollflöde.

(5 p)

b) Klassen `Gui` är påbörjad:

```
public class Gui extends JFrame ...
{
    private JLabel questionLabel = new JLabel();
    private JLabel levelLabel = new JLabel();
    private JLabel marksLabel = new JLabel("Du har 0 poäng");
    private JButton nextButton = new JButton("Nästa tips");
    private JTextArea clueArea = new JTextArea(5,15);
    private JTextField guessField = new JTextField(15);
    private GameController controller;

   ...
```

Vi bortser från hur fönsterlayouten skapas i denna uppgift. Konstruera de lyssnare som behövs och addera dem till lämpliga komponenter i fönstret. Komplettera med eventuell(a) metod(er) som krävs enligt klassdiagrammet. Nedan visas de olika Swing-komponenternas roller i fönstret: (5 p)

c) Skriv en `main`-metod som skapar och kopplar ihop objekt av lämpliga klasser så att ett fungerande "På håret"-program erhålls. (4 p)

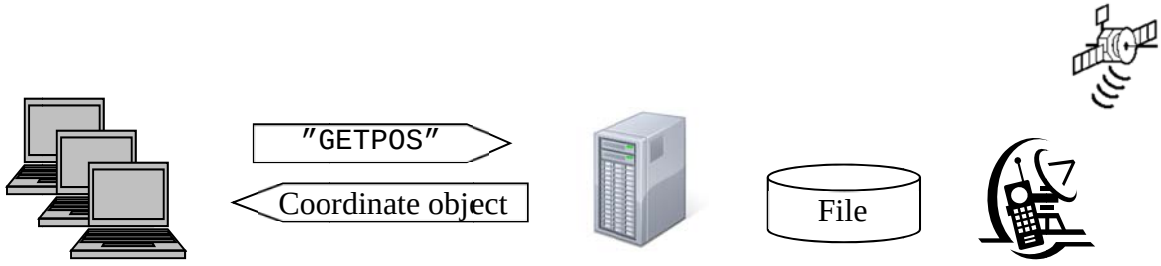
Uppgift 4

En GPS-motagare lagrar med jämna mellanrum den mottagna geografiska koordinaten i en textfil med följande format:

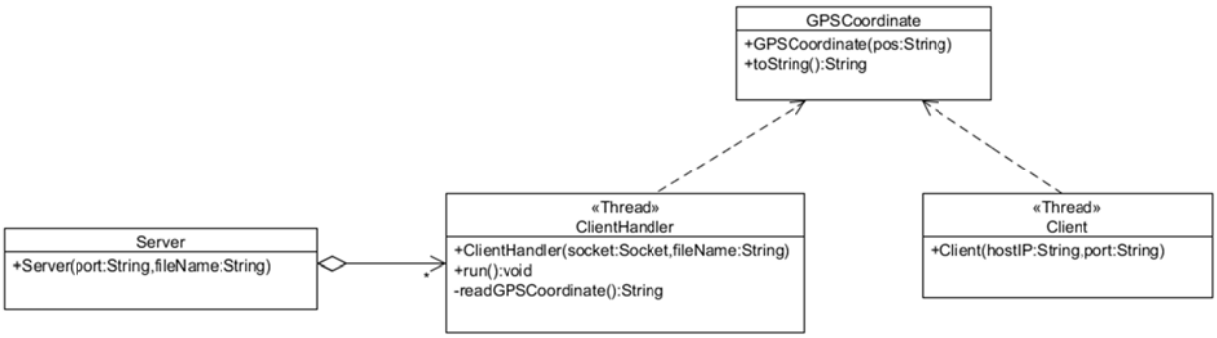
{N/S}<grader> <minuter> {E/W} <grader> <minuter>

Exempel: `N57 37.929 E11 36.068`

På samma dator finns en server som på begäran från klienter läser innehållet i textfilen och skickar det till klienter som begär att få uppgiften. Mellan klienterna och servern finns en uppkopplad förbindelse.



En klient begär att få den senast mottagna positionen genom att skicka strängen `"GETPOS"` till servern. Servern sänder då tillbaka ett objekt av typen `GPSCoordinate`. Mjukvaran beskrivs av följande klassdiagram:



De tre undre klasserna skall konstrueras i uppgiften. Konstruktorn och metoden i `GPSCoordinate` beskrivs enklast med följande exempel:

```
String s = "N57 37.929 E11 36.068";
```

((**new** `GPSCoordinate(s)).toString()).equals(s)` ger **`true`**

a) Implementera klassen `Server` som hanterar klienterna. För varje klient som ansluter skall en tråd av typen `ClientHandler` skapas. Konstruktorn skall ta portnumret som servern skall lyssna på, samt namnet på textfilen som innehåller GPS-koordinaten. Hantera undantag på lämpligt sätt.

(4 p)

b) Implementera trådklassen `ClientHandler`. Klassen skall betjäna en klient via en uppkopplad förbindelse. Inparametrar till konstruktorn är socketen som skall användas för kommunikationen med klienten, samt namnet på textfilen som innehåller GPS-koordinaten. Tråden skall oupphörligt ta emot en sträng, och om denna är `"GETPOS"`, hämta GPS-koordinaten från textfilen och returnera ett objekt av typen `GPSCoordinate` till klienten. För att läsa koordinatfilen kan du anta att klassen har den färdiga metoden

```
private String readGPSCoordinate()
```

Hantera undantag på lämpligt sätt.

(6 p)

c) Implementera trådklassen `Client`. Klassens konstruktor tar serverns IP-address och portnummer som inparametrar. Först skall tråden etablera en förbindelse med servern. Sedan skall den oupphörligt varje sekund skicka ett `"GETPOS"`-kommando till servern och därefter ta emot ett koordinatobjekt från servern. Innehållet i detta skall skrivas ut på skärmen. Hantera undantag på lämpligt sätt.

(7 p)

Uppgift 5

I ett program för bildbehandling implementeras olika grafiska filter med speciella filterklasser. Grundtypen för dessa är

```
public interface Filter {
    void apply(Image im);
}
```

Några implementerande subklasser är t.ex. `RedFilter` och `BlueFilter`. Metoden `apply` genomför den aktuella filtreringsfunktionen på bildobjektet `im`. I denna uppgift skall vi konstruera en klass som gör det möjligt att göra nya filterklasser tillgängliga för programmet, vilka kanske inte fanns när programmet skapades.

a)
`public class` `FilterFactory`

Lagrar ett antal `Filter`-objekt i en intern tabell. Klassen kan på förfrågan returnera lagrade filter, eller ladda in ytterligare filter.

Metoder

`public void` `addFilter(Filter f)`

Adderar `f` till tabellen.

`public` `Filter getFilter(String name)`

Returnerar ett filterobjekt med namnet `name` om det finns i tabellen, annars **`null`**.

`public` `Filter loadFilter(String name)`

Laddar filtret med namnet `name` och adderar det till tabellen om ett sådant finns i aktuell filkatalog. Det nya filtret returneras, eller **`null`** om inget fanns.

`public` `Set<String> getKeys()`

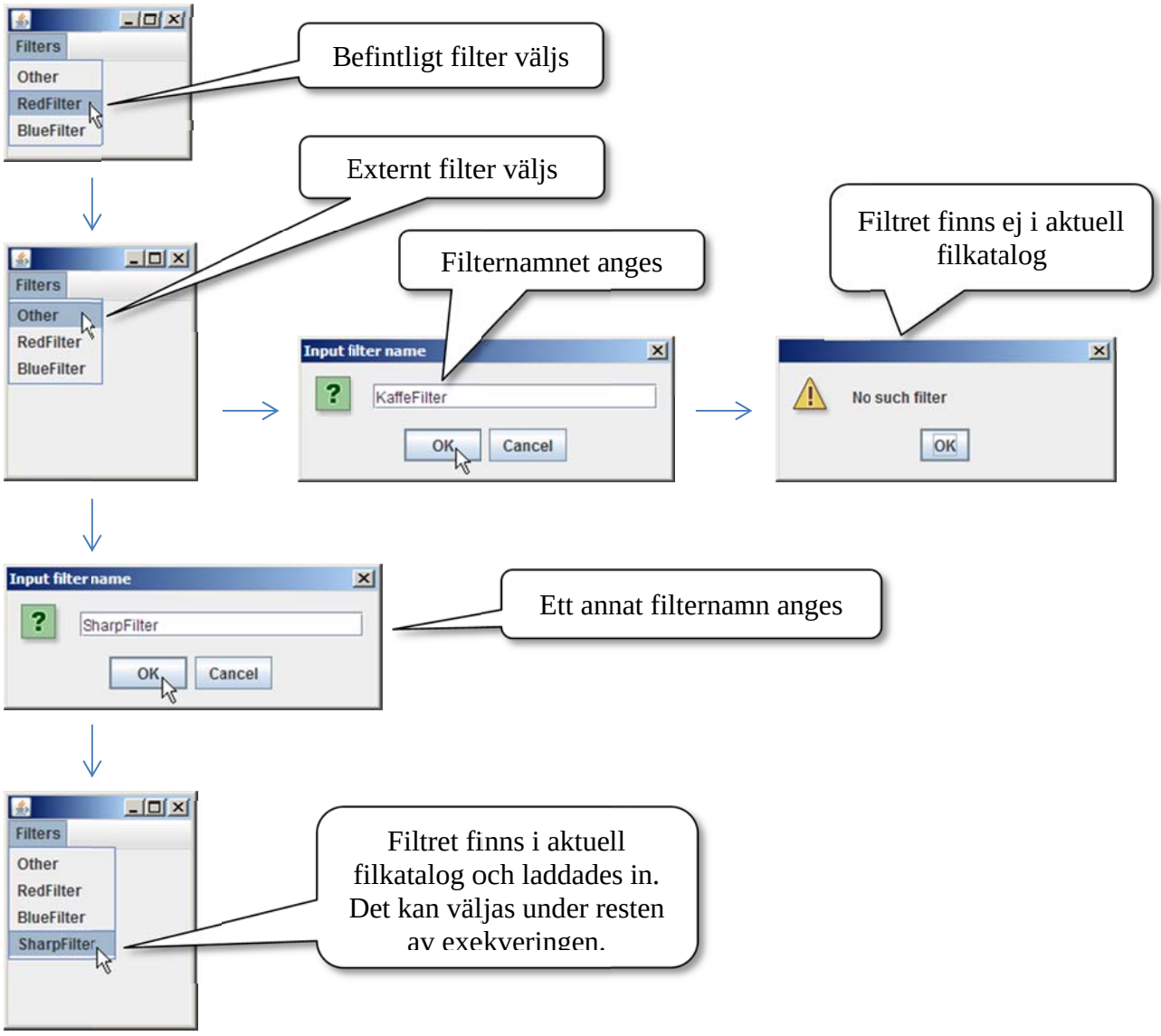
Returnerar en mängd av namnen på de lagrade filterobjekten.

Implementera klassen! Lagra filterobjekten i en lämplig datastruktur.

(5 p)

b)
I denna uppgift skall vi konstruera en utbyggbar filtermeny som kan användas i ett tänkt program för bildbehandling. Menyn skall ha några standardfilter när programmet startas, men också kunna byggas ut med fler filter under exekveringen. Om menyalternativet `Other` väljs skall användaren kunna mata in ett filternamn i en pop-up-meny. Om filtret inte redan är lagrat men finns i aktuell katalog så skall det laddas in och ett extra menyalternativ för det nya filtret adderas sist i filtermenyn. Är filtret redan lagrat händer ingenting. Är det inte lagrat och saknas i aktuell katalog skall man få ett felmeddelande. När ett filter valts i menyn skall motsvarande filterobjekts `apply`-metod anropas med en bild som inparameter. Ett exempelscenario finns på nästa sida.

Exempelscenarion:



Du kan utgå från att följande instansvariabler finns:

```
public class Gui extends JFrame {
    private FilterFactory filterFactory;
    private JMenuBar menuBar;
    private JMenu filterMenu;
    private Image image;
...
private JMenu makeFilterMenu(String menuName) { ... }
```

Implementera denna metod. Inför ytterligare lämpliga privata hjälpmetoder. Grundalternativen i menyn skall hämtas från `filterFactory`.

(10 p)