Tentamen i EDAF25

24 augusti, 2023

Skrivtid: 8-13

- Skriv bara på ena sidan av pappret
- Skriv tydligt – om texten inte går att läsa kan du inte få några poäng.
- Sätt identitet och sidnummer på varje inlämnat blad, kontrollera att sidnumret på din sista sida är samma som det antal blad du markerar på omslagspappret.

Tentamen består av uppgifter med totalt 40 poäng. Dessa poäng är fördelade på två avdelningar: *Grafteori* (10p) och *Design* (30p). För godkänt betyg kommer att krävas högst hälften av den sammanlagda poängen för de två avdelningarna.

Vid bedömningen kommer hänsyn att tas till lösningens kvalitet. UML-diagram skall ritas i enlighet med UML-häftet.

Hjälpmedel:
- Andersson: UML–syntax
- Holm: Java snabbreferens

Grafteori

Uppgift 1

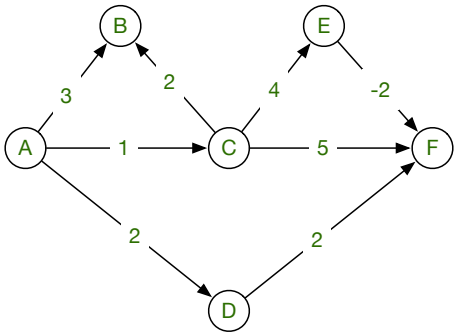
Någon har skojat med Roger och lagt in fel i den pseudokod för Dijkstras algoritm som finns i laborationshandledningen för kursen och som återfinns nedan. Hjälp Roger att hitta felen och ange också hur pseudokoden borde se ut. Varje ändring berör endast en rad, så svara med radnumret för den felaktiga raden och ange hur raden borde se ut för att algoritmen ska bli korrekt. Varje hittat och rättat fel ger 1 poäng. Det finns totalt fem (5) fel.

Förutsättningarna för pseudokoden är följande: Vi vill hitta och returnera längden på den kortaste vägen från en nod u till en annan nod v. Kön som skapas på rad 2 innehåller element med två attribut: en nod i grafen och dess avstånd från ursprungsnoden på den kortaste vägen. I koden betecknas denna typ av element {nod, avstånd}. *5 p*

```
1  markera alla noder som besökta
2  pq = en tom fifokö
3  lägg till {u, 0} i pq
4  markera u som besökt
5  spara avståndet till u = 0
6  så länge pq inte är tom:
7      {x, dist} = ta ut första elementet ur pq
8      om x == v:
9          returnera dist
10     annars:
11         för varje båge e från u:
12             w = e.destination
13             newdist = e.distance
14             wdist = det sparade avståndet från u till w
15             om w är besökt och newdist < wdist:
16                 markera w som besökt
17                 spara avståndet till w = newdist
18                 // Notera: w kan redan finnas i pq med ett längre avstånd.
19                 lägg till {w, newdist} i pq
20 returnera -1 // Om v ej hittats.
```

Uppgift 2

Betrakta följande graf:



(a) Antag att vi använder Dijkstras algoritm för att hitta det minsta avståndet från nod A till nod F. Vilket värde kommer algoritmen i så fall att ge oss? *2 p*

(b) Motivera varför algoritmen kommer fram till detta värde. *2 p*

(c) Vilket värde borde Bellman-Fords algoritm ge för det minsta avståndet från nod A till nod F? *1 p*

Design

Uppgift 3

I ett kalkylprogram finns två klasser för att läsa och skriva på filer. De två klasserna är nästan identiska så när som på fyra rader. Den ena klassen visas nedan med de skiljande raderna inlagda som kommentarer.

```
public class OpenMenuItem extends MenuItem implements ActionListener {
//public class SaveMenuItem extends MenuItem implements ActionListener {
        private final Gui gui;
        private int action = FileDialog.LOAD;
        // private int action = FileDialog.SAVE;
        private String title;
        public OpenMenuItem(Gui gui, String title) {
        // public SaveMenuItem(Gui gui, String title) {
                super(title);
                this.gui = gui;
                this.title = title;
                addActionListener(this);
        }
        public void actionPerformed(ActionEvent event) {
                try {
                        FileDialog fileDialog = new FileDialog(gui,
                                                        title, action);
                        fileDialog.setVisible(true);
                        String file = fileDialog.getFile();
                        String dir = fileDialog.getDirectory();
                        String fullName = dir + file;
                        fileDialog.dispose();
                        if (file == null) { return }
                        gui.load(new BufferedReader(new FileReader(
                                                        fullName)));
                        // gui.save(new BufferedWriter(new FileWriter(
                                                           fullName)));
                } catch (Exception ex) {
                        gui.setMessage(ex.toString());
                }
        }
}
```

Använd Mallmetod(Template Method)-mönstret för att eliminera duplicerad kod. Svara med Java-kod.

8 p

Uppgift 4

I Java görs ofta utskrifter till det globalt tillgängliga objektet `System.out` – det är deklarerat som en instans av standardklassen `PrintStream`. För enkelhets skull antar vi i uppgiften att `PrintStream` har följande specifikation:

```
class PrintStream {
    public void println(String s) {
        // ... omissions ...
    }
}
```

(alltså bara en metod, och klassen ärver direkt från `Object`).

I inlämningsuppgift 1 (Computer) är våra program besvärliga att testa automatiskt, eftersom alla utskrifter hamnar i terminalfönstret, och det är svårt för våra testprogram att kontrollera text som skrivits ut i ett terminalfönster. Vi vill i denna uppgift göra det enklare att testa våra projekt.

Vi förutsätter att ni i er lösning av projektet hade ett interface `Command` som implementerades av alla instruktioner (exakt hur klasserna `PC` och `Memory` fungerar spelar ingen roll i denna uppgift):

```
interface Command {
    void execute(PC pc, Memory memory);
}
```

och att ni i er `PrintCommand`-klass skrev:

```
class PrintCommand implements Command {

    // ... omissions ...

    public void execute(PC pc, Memory memory) {
        // ... omissions ...
        System.out.println(???);
    }
}
```

Här är `???` ett uttryck som beror av hur resten av klassen `PrintCommand` är implementerad, och vi kan ignorera dessa detaljer i uppgiften nedan.

(a) När vi körde våra Computer-program i projekt 1 hamnade alla utskrifter på `System.out`, vi vill nu istället att man skall kunna välja vilken utskriftsenhet utskrifterna skall hamna på. Ett första steg i att göra detta är att införa ett interface för utskriftsenheter.

1. Vilken eller vilka SOLID-principer föreskriver detta?
2. Hur bör interfacet se ut?
3. Hur borde `PrintStream` vara deklarerad om vi hade haft detta interface?

2 p

(b) Vi antar i resten av uppgiften att `PrintStream` är deklarerat som du gjorde i deluppgiften ovan.

För att kunna skicka utskrifterna från våra instruktioner till en given utskriftsenhet måste vi ändra deklarationen av vårt Command-interface ovan så att man kan skriva exempelvis:

```
cmd.execute(pc, memory, System.out);
```

om vi vill skriva till `System.out`, och vi har skapat `cmd`, `pc` och `memory` med instruktionerna:

```
Command cmd = ... PrintCommand-object ...;
PC pc = ... PC-object ...;
Memory memory = ... Memory-object ...;
```

Men observera att vi även vill kunna skicka in andra utskriftsenheter än `System.out` till ett execute-anrop.

För att det skall gå måste vi komplettera deklarationerna:

```
interface Command {
    void execute(PC pc, Memory memory, ...);
}

class PrintCommand implements Command {

    // ... omissions ...

    public void execute(PC pc, Memory memory, ...) {
        // ... omissions ...
        // ersätt System.out.println(???) med ny kod
        // som skriver ut ???
    }
}
```

och eventuellt göra ytterligare någon deklaration (du behöver alltså inte ändra `???` i utskriften ovan, men du måste ange hur resten av utskriftsraden skall se ut, alltså vad som skall stå runt om `???`).

Som svar på denna deluppgift vill vi ha de nya versionerna av `Command` och `PrintCommand` enligt texten ovan, och eventuell kod som behövs för att komplettera dem. *2 p*

(c) Baserat på deklarationerna ovan, använd *Decorator Pattern* för att implementera en klass som kan kapsla in en utskriftsenhet, och som gör det möjligt för oss att senare hämta de strängar som skickats till utskriftsenheten – vi vill få dem tillbaka som en sammansatt sträng (en sträng som består av alla strängar som skrivits ut hopslagna till en enda). *4 p*

(d) *Denna deluppgift kräver att du har lite koll på Computer-projektet – man måste inte klara deluppgiften för att klara tentan, men för att få överbetyg bör man göra en rimlig lösning.*

I Computer-projektet hade vi ungefär följande kod i ett huvudprogram:

```
var wf = new LongWordFactory();
var program = new Factorial("5", wf);
var computer = new Computer(new Memory(1024, wf));
computer.load(program);
computer.run();
```

Hur skall vi göra i detta program för att få utskrifter till `System.out`, och dessutom i efterhand kunna kontrollera att vi aldrig fick texten ”ERROR” i någon av våra utskrifter?

Du kommer att behöva skapa ett lämpligt objekt, och skicka in på lämpligt sätt, och sedan använda detta objekt för att i huvudprogrammet testa om texten ”ERROR” förekommit eller ej – om texten ”ERROR” förekommit skall programmet göra en lämplig utskrift på `System.out`. Du kan behöva ändra anropet till någon av metoderna ovan, men behöver inte skriva om den aktuella metoden (all kod du skall skriva i denna deluppgift är den som finns i huvudprogrammet. *4 p*

Uppgift 5

Vi vill i denna uppgift använda *Observer/Observable Pattern* för ett väldigt litet exempel. Javas standardbibliotek har sedan version 1.0 en implementation av mönstret (kommer ni ihåg den från projekt 2?), och även om denna implementation idag är lite omodern, så vill vi använda den i uppgiften (eftersom den väldigt väl illustrerar principen bakom mönstret, och alternativet hade varit att använda ett mycket mer komplicerat bibliotek). Om du känner dig osäker på någon detalj i biblioteket kan du skriva en kommentar och använda ett namn eller en typ som känns rimlig. Vi är inte ute efter att testa om du kommer ihåg exakt hur biblioteksimplementationen ser ut utan snarare om du kommer ihåg hur mönstret fungerar.

Vi vill ha en klass `Person` som beskriver en person med ett namn och en metod för att ändra namnet. Det skall vidare vara möjligt för omgivningen att med hjälp av Observer-mönstret få veta när namnet ändras.

Vi vill även ha en klass för myndigheter (`Agency`) som skall lyssna på namnändringar. Myndigheter skall göra det genom att hela objektet (myndigheten själv) lyssnar på ändringar, och alltså implementerar något standardgränssnitt. När en person uppdaterar sitt namn skall myndigheten informeras och skriva ut det nya namnet på `System.out`. Klassen `Agency` skall ha en metod med signaturen:

```
public void register(Person p);
```

som gör att vi noterar alla uppdateringar som en given person gör i framtiden.

(a) Skriv klasserna `Person` och `Agency` enligt ovan – när vi ändrar namnet på en person som de (på något sätt) observerar, så skall myndigheten skriva ut namnet på personen. Skriv även ett huvudprogram som skapar en person och en myndighet, samt låter myndigheten hålla koll på personen (med hjälp av metoderna ovan). *4 p*

(b) Rita ett sekvensdiagram som visar vad som händer när en person ändrar namn, och en myndighet hanterar ändringen genom att skriva ut namnet på personen. *3 p*

Uppgift 6

Förklara kortfattat begreppen cohesion (sammanhang) och coupling (koppling), och principerna SRP, DIP och OCP. Förklara även hur och varför var och en av de tre principerna påverkar begreppen cohesion och coupling.

Du behöver inte göra någon lång beskrivning – försök sammanfatta beskrivning i en eller några få meningar, men den skall vara tydlig. *3 p*