Tentamen i EDAF60

28 oktober 2019

Skrivtid: 14-19

- Skriv bara på ena sidan av pappret – tentorna kommer att scannas in, och endast framsidorna rättas.
- Skriv **inte** med färgpenna – de enda tillåtna färgerna är svart/mörkblått/blyerts.
- 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.

Poäng och betygssättning

Delpoängen på tentan räknas till antingen teori eller praktik (markerade med *T* eller *P* nedan), reglerna för betygen är:

Betyg 3: minst 60% på både teori- och praktikdel

Betyg 4: minst 70% på både teori- och praktikdel

Betyg 5: minst 80% på både teori- och praktikdel

Hjälpmedel: • Pennor och blanka papper

Uppgift 1

(a) Beskriv i högst 75 ord mönstret *Template Method*. *T 1 p*

(b) Beskriv i högst 75 ord mönstret *Strategy*. *T 1 p*

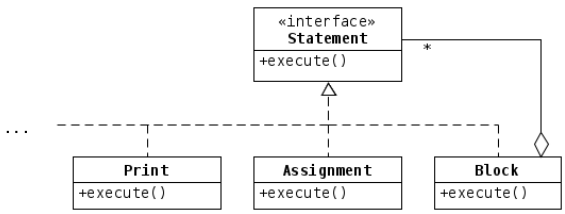
(c) Man kan tänka sig olika sätt att utvidga Computer-projektet med ny funktionalitet. För en design som krävs för att bli godkänd på uppgiften gäller att den följer *Open Closed Principle* (OCP) för vissa slags utvidgningar, men inte för andra utvidgningar.

Ange två naturliga typer av utvidgningar av projektet för vilka vi har OCP – motivera ditt svar. *T 2 p*

(d) Ange en naturlig utvidgning av Computer-projektet för vilken vi inte har OCP – motivera ditt svar. *T 1 p*

Uppgift 2

(a) Vilket/vilka mönster kan du identifiera i denna del av ett större klassdiagram:



Motivera ditt svar. *T 2 p*

(b) Vi har två paket med programkod, i paketet `consumer` har vi:

```
package consumer;
import supplier.Store;
public class Client {
    private int clientId;
    public Client(Store store) {
        clientId = store.generateId();
    }
    // ... utelämnade metoder ...
}
```

och i paketet `supplier` har vi:

```
package supplier;
public class Store {
    public int generateId() {
        // ... utelämnad kod ...
    }
    // ... utelämnade metoder ...
}
```

Klassen `Client` kan inte kompileras utan att vi har tillgång till `Store`-klassen – vilken SOLID-princip strider detta mot? Motivera ditt svar. *T 1 p*

(c) Rita ett klassdiagram för en ny design av programmet i (b), som är i enlighet med den princip som vi bryter mot ovan. Förklara i ord varför vi inte längre bryter mot principen. *T 2 p*

Uppgift 3

Lös följande två uppgifter med hjälp av Javas `stream`-paket:

(a) Skriv en funktion

```
List<String> wordsOfLength(String text, int wordLength)
```

som ger en lista med alla de ord i strängen `text` som innehåller precis `wordLength` tecken. Exempel: för strängen `"to be or not to be"` och `wordLength` 2 vill vi få listan `["to", "be", "or", "to", "be"]`. Vi förutsätter att strängen bara innehåller ord och mellanslag, så att vi kan använda `text.split(" ")` för att bryta upp den till en vektor med strängar. *P 2 p*

(b) Skriv en funktion

```
int partialSum(List<String> strings, int min, int max)
```

som ger summan av de av strängarna i listan `strings` som är heltal och vars värde ligger i intervallet `min` ... `max` (inklusive gränserna). Exempel: för listan `["ett", "4", "8", "4.2", "3"]` skall vi få värdet 7 om vi anger gränserna 3 och 5. Du kan använda de färdigskrivna hjälpfunktionerna:

```
boolean isInt(String s)    // avgör om s innehåller ett heltal
int toInt(String s)        // omvandlar s till ett heltal om det går, kraschar annars
```

P 3 p

Lösningar som inte baseras på `stream` ger inga poäng.

Uppgift 4

Vi har en klass `StandardLever` som beskriver en spak som kan lyftas:

```
class StandardLever {
    // ... attribut och metoder ...
    public void raise() {
        // ... lyfter spaken ...
    }
}
```

Vi vill kunna undersöka vad som händer när en spak används, och vill använda en variant av *Decorator Pattern*. I sin enklaste form skulle Decorator Pattern låta varje decorator själv anropa `raise()` på det inkapslade objektet, men här vill vi istället att det skall skötas automatiskt, och att våra dekoratorer istället skall implementera två metoder, `preRaise()` och `postRaise()`, som anropas precis före respektive efter att spaken lyfts.

(a) Skriv ett interface `Lever` och en abstrakt klass `DecoratedLever` enligt texten ovan. Subklasser till `DecoratedLever` skall *inte* implementera `raise()` (den skall anropas automatiskt av `DecoratedLever`), däremot skall de implementera de två metoderna `preRaise()` och `postRaise()`. *P 2 p*

(b) Skriv en klass `LoggedLever` som subklass till `DecoratedLever` – den skall ge utskriften:

```
Before raising the lever
Raising the lever
After raising the lever
```

om den dekorerar ett objekt av klassen:

```
class PrintingLever implements Lever {
    public void raise() {
        System.out.println("Raising the lever");
    }
}
```

Implementera även en klass `Main` som bara innehåller en `void`-metod `run()` – metoden skall skapa en `PrintingLever`, dekorera den med en `LoggedLever`, och låta dem generera utskrifterna ovan. *P 2 p*

(c) Antag att vi skriver två klasser för att logga, en `SwedishLogger` som skriver ut på svenska: 'före' i `preRaise()` och 'efter' i `postRaise()`, och en `FinnishLogger` som skriver ut på finska: 'ennen' i `preRaise()` och 'mukaan' i `postRaise()`. Vilken utskrift får vi om vi skriver:

```
new SwedishLogger(new FinnishLogger(new PrintingLever())).raise();
```

På denna deluppgift får du *0.25 p* om du svarar 'vet ej', och *0 p* om du svarar fel. *P 1 p*

(d) Rita ett klassdiagram som visar din lösning. *T 1 p*

Uppgift 5

Vi skall skriva programkod för att låta en användare kontrollera ett kretskort med en mp3-spelare med hjälp av knappar i ett GUI. Nedanstående klasser och interfaces är färdigskrivna – för exempel på hur dessa klasser och interfaces används, se klassen `OriginalPlayer` nedan.

Klassen `Button` och gränssnitten `ButtonEvent` och `ButtonPressHandler` beskriver knappar och tryck på knappar:

```
public class Button {
    public Button(String label);
    public void addButtonPressHandler(ButtonPressHandler s);
}
public interface ButtonEvent {
    Button source();                    // anger vilken knapp som tryckts
    Instant timeStamp();                // anger tidpunkten
}
```

```
public interface ButtonPressHandler {
    void handleButtonPress(ButtonEvent event);  // anropas vid tryck på knappen
}
```

De mp3-kretsar som vi vill kunna styra implementerar `Mp3Circuit`:

```
public interface Mp3Circuit {
    void togglePlay();              // startar eller pausar uppspelning
    void skip(int step);            // hoppar bland spåren, negativa värden hoppar bakåt
    void changeVolume(int percent); // ändrar volymen, negativa värden sänker den
}
```

Vi har dessutom ett interface för användargränssnittet:

```
public interface Gui {
    void add(Button button);        // lägger in en knapp i GUI:t
}
```

och en klass `OmdGui`, som implementerar `Gui`.

Någon har skrivit en klass `OriginalPlayer`, som låter användaren styra mp3-kretsen:

```
class OriginalPlayer extends OmdGui implements ButtonPressHandler {

    private Mp3Circuit circuit;
    private Button toggleButton;
    private Button previousButton;
    private Button nextButton;

    public OriginalPlayer(Mp3Circuit circuit) {
        this.circuit = circuit;
        toggleButton = new Button("play/pause");
        toggleButton.addButtonPressHandler(this);
        add(toggleButton);
        previousButton = new Button("previous");
        previousButton.addButtonPressHandler(this);
        add(previousButton);
        nextButton = new Button("next");
        nextButton.addButtonPressHandler(this);
        add(nextButton);
    }

    public void handleButtonPress(ButtonEvent event) {
        if (event.source() == toggleButton) {
            circuit.togglePlay();
        } else if (event.source() == previousButton) {
            circuit.skip(-1);
        } else if (event.source() == nextButton) {
            circuit.skip(1);
        }
    }
}
```

(a) Klassen `OriginalPlayer` bryter mot *Open Closed Principle* (OCP), skriv en klass `SolidPlayer` som fungerar som `OriginalPlayer`, men som följer OCP. *P 4 p*

(b) Om du lyckas med omskrivningen i (a), så är det enkelt att lägga till nya knappar utan att ändra i klassen `SolidPlayer`. Skriv programrader som skapar en `SolidPlayer`, och sedan i efterhand lägger till knappar för att höja och sänka ljudvolymen 10% (din `SolidPlayer` måste då ha en publik metod som låter användaren lägga till nya knappar på något sätt). Du kan i ditt program använda ett objekt av klassen `OmdMp3Circuit`, som implementerar `Mp3Circuit`. *P 1 p*