Tentamen i Objektorienterad modellering och design (OMD) Helsingborg

EDAF25

30 maj 2022, 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 30 poäng. För godkänt betyg kommer att krävas högst hälften av poängen. Vid bedömningen kommer hänsyn att tas till lösningens kvalitet. UML-diagram skall ritas i enlighet med UML-häftet.

Hjälpmedel: • Andersson: UML–syntax

Uppgift 1 13 P.

Denna uppgift innehåller ett antal deluppgifter, där varje deluppgift har ett påstående och en anledning. För varje deluppgift, svara med ett av följande alternativ:

A Både påståendet och anledningen är korrekta uttalanden och anledningen förklarar påståendet på ett korrekt sätt.

B Både påståendet och anledningen är korrekta uttalanden, men anledningen förklarar inte påståendet.

C Påståendet är ett korrekt uttalande, men anledningen är ett felaktigt uttalande.

D Påståendet är felaktigt, men anledningen är ett korrekt uttalande.

E Både påståendet och anledningen är felaktiga uttalanden.

- Svara inte i uppgiftshäftet, utan skriv ditt svar på ett vanligt lösningspapper.
- Varje korrekt besvarad deluppgift ger 1.0 poäng
- Vi vill inte att ni skall chansa, så ni får *minuspoäng* (-0.25 poäng) på en deluppgift om ni svarar fel, men aldrig minuspoäng totalt sett på hela uppgift 1.

	Påstående	Anledning
T1	Från en matris som representerar en graf kan man direkt utläsa avståndet mellan alla noder i grafen.	För att representera grafen med en matris så sparas varje båges avstånd i matrisen. Till exempel, avståndet för bågen (R, K) finns i cellen på rad R och kolumn K i matrisen.
T2	Bredden-först traversering kan användas för att hitta kortaste avståndet mellan noder i en oviktad graf.	Bredden-först traversering utgående från en nod X kommer att besöka alla noder som ingår i samma komponent som X.
T3	Att koden är stel (Rigidity) innebär att en förändring i en del av programmet inte leder till några förändringar i andra delar av programmet.	I ett stelt program är klasserna hårt kopplade, vilket innebär att de inte är beroende av varandra.
T4	Ett objekt vars tillstånd inte kan förändras kallas muterbart(mutable).	Integer-objekt och String-objekt är muterbara.
T5	Genom att följa OCP principen väl så kan vi utöka en moduls beteende utan att göra ändringar i befintlig kod.	I objektorienterade språk som t.ex. Java kan vi skapa abstrakta klasser som är stängda för förändring men lämnar öppet hur den konkreta klassen implementerar abstraktionen.
T6	Genom att följa SRP minskar man sammanhangsfaktorn (cohesion) i en klass.	SRP kan följas genom att extrahera det fixa beteendet till en basklass (stängd för förändring) och kapsla in det som varierar i termer av utvidgningar av basklassen.
T7	Klassnamn inleds alltid med liten bokstav.	Ett klassnamn är normalt ett substantiv hämtat från uppdragsgivarens beskrivning eller den verklighet som programmet modellerar.
T8	Den abstrakta klassen `DecoratedRobot` i exemplet nedan är ett exempel på hur dekoratörsmönstret (Decorator pattern) kan användas för att undvika duplicerad kod.	Dekoratörsmönstret implementeras i exemplet genom att dekoratören `DecoratedRobot` implementerar samma gränssnitt som ursprungsobjektet, d.v.s. `Robot`, och delegerar den ursprungliga funktionaliteten till det ursprungliga objektet samtidigt som ny funktionalitet adderas till metoden `move()`.
T9	I kommandomönstret (Command pattern) instansierar alla kommando-objekt klasser som implementerar ett gemensamt Kommando-interface.	Med hjälp av kommandomönstret kan man öka kopplingen (coupling) mellan anroparen och objektet som ska utföra handlingen.
T10	Den abstrakta klassen `DecoratedRobot` i exemplet nedan (Figur **??**) visar ett exempel på mallmetod-mönstret (Template method).	I klassen `DecoratedRobot` utgör metoden `move()` en mallmetod.
T11	Strategy är ett bra mönster då man vill kunna ändra beteendet hos ett objekt under exekvering.	En negativ konsekvens av strategimönstret är att man bryter mot OCP.
T12	Kompositmönstrets struktur liknar Dekoratörsmönstret men har ett annat syfte.	Med kompositmönstret vill man kunna följa SRP genom att låta varje komponent ha ett väl avgränsat ansvarsområde.
T13	Den abstrakta klassen `DecoratedRobot` i Figur **??** nedan är ett exempel på hur kompositmönstret har implementerats.	`DecoratedRobot` kapslar in anropet av ett kommando så att man kan separera det från dess konstruktion och exekvera det oberoende av var och när det instansieras.

```
abstract class DecoratedRobot implements Robot {
  protected Robot robot;

  public DecoratedRobot (Robot robot){
    this.robot = robot;
  }

  public final Point2D getPosition(){
    return robot.getPosition();
  }

  public final void move(double dx, double dy){
    robot.move(dx, dy);
    decorate(dx, dy);
  }
  protected abstract void decorate(double dx, double dy);
}
```

Kodexempel till teorifrågorna **T8, T10 och T13**

Uppgift 2 3 P.

I en sattelit-navigator för bilar finns det en metod `routeTo` som söker en väg från bilens nuvarande position till en adress som användaren skrivit in (`endpoint`). En sådan väg beskrivs med klassen `Route` och `EmptyRoute` används om det inte finns en möjlig väg mellan start- och slut-punkt.

För att hitta en väg används först metoden `addressToGCS` i klassen `GCSDatabase` för att översätta slutpunktens adress till en koordinat i GCS-format (Geographic Coordinate System). Det är möjligt att användaren har skrivit in en address som inte finns i databasen och då returnerar `addressToGCS` en `null`-referens. Detta leder till att man måste ha `if`-satser som testar om GCS-koordinaten är `null` när väg ska hittas med `findRoute` och när slutpunkten visas på kartan med `map.panTo(endpoint)`.

Visa hur koden kan skrivas om med Null Object Pattern så att alla `if`-satser förutom den i `addressToGCS` kan tas bort. Redovisa alla förändringar i koden, inklusive eventuella nya klasser och metoder.

```
class GCS {
  Route findRoute(GCS endpoint) {
    if (endpoint == null) {
      return new EmptyRoute();
    }
    return new Route(this, endpoint);
  }
}

class GCSDatabase {
  GCS addressToGCS(String address) {
    if (found) {
      return new GCS(...);
    }
    return null; // Address not found.
  }
}

class Main {
  GCSDatabase db;
  GPS gps;
  StreetMap map;

  Route routeTo(String address) {
    GCS start = gps.getMyPosition();
    GCS endpoint = db.addressToGCS(address);
    Route route = start.findRoute(endpoint);
    if (endpoint != null) {
      map.panTo(endpoint); // Show endpoint on map.
    }
    return route;
  }
}
```

Uppgift 3 7 P.

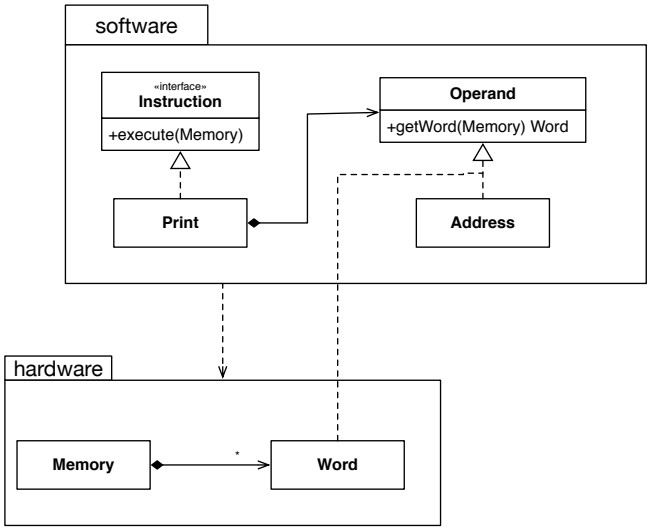
Följande Computer-design, se figur 3 har ett cykliskt beroende mellan paketen. Gör en ny design utan att flytta några givna klasser. Det är tillåtet att lägga till klasser och gränssnitt.

a) Lösningen ges med klassdiagram.

b) Beskriv även ditt tillägg med Javakod.

c) Beskriv hur förändringen påverkar paketens position i A-I grafen. Rör de sig i en bra riktning? Motivera ditt svar.

d) Uppfyller den givna paketindelning de båda sammanhangsprinciperna REP (Reuse equivalence principle) och CCP (Common closure principle)? Motivera ditt svar. Svar utan motivering ger inga poäng.



Figur 1: cykliskt beroende

Uppgift 4 7 P.

En testklass till en simulator för en liten dator innehåller

```
public class Factorial extends Program {
    public Factorial() {
        Address n = new Address(0),
        add(new Copy(new LongWord(5), n));
        // omissions
    }
}
```

Det första argumentet i konstrueraren `Copy(Word, Address)` skall ha ett argument som implementerar `Word` och är av den typ som datorns minne har. Testklassen går alltså inte att använda för en dator vars minne innehåller `VeryLongWord`. Använd någon fabriksteknik *Factory* så att man kan använda samma `Factorial`-klass för alla minnestyper som implementerar `Word`. Det argument som anger vilket tal som skall representeras av ett `Word` skall vara av typen `String`.

a) Lösningen redovisas med ett Java-gränssnitt som alla `Word`-fabriker skall implementera och en modifierad version av `Factorial` som använder en godtycklig sådan fabrik.

b) Beskriv lösningen med ett UML-klassdiagram, där relevant information framgår.

c) Beskriv med hjälp av ett objektdiagram relationerna mellan de objekt som skapas vid konstruktion av `Factorial`

Lösningsförslag och rättningsmallar

Lösningsförslag till uppgift 1

1p för korrekt svar, 0p för blankt, och -0,25 för fel svar (om inget annat anges i lösningsförslaget)

T1 D En graf representerad med en matris har endast information om avståndet mellan noder som är direkta grannar, dvs. det finns en båge mellan noderna. För att få avståndet mellan två noder som inte är grannar så måste en algoritm såsom Dijkstra användas.

Den ursprungliga formuleringen av påståendet var tvetydig varför även svaret A godkändes.

T2 B

T3 E Det omvända gäller: ett stelt program är svårt att göra ändringar i efter som små ändringar kan leda till en kaskad av ändringar i beroende moduler. Dessa moduler kan sägas vara hårt kopplade det vill säga att det finns ett starkt beroende mellan modulerna.

T4 E (B->0,5) Ett objekt vars tillstånd inte kan förändras kallas omuterbart Integer-objekt och String-objekt är omuterbara.

T5 (A och B) Både påstående och anledning stämmer, anledningen ger exempel på hur man kan följa OCP väl.

T6 E, SRP - En klass ska bara ha en anledning till förändring (låg koppling, hög sammanhangs faktor) Jmfr DRY samma ansvar ska inte finnas någon annanstans i systemet. Beskrivningen i anledningen handlar om en annan princip (OCP).

T7 D Ett klassnamn är normalt ett substantiv hämtat från uppdragsgivarens beskrivning eller den verklighet som programmet modellerar. Namnet inleds med en versal och följs av gemener.

T8 D Den abstrakta klassen `DecoratedRobot` är ett exempel på hur dekoratörsmönstret (Decorator pattern) har använts för att kunna dynamiskt kunna lägga till extra funktionalitet till en robot som implementerar `Robot`-gränssnittet utan att behöva ändra i den ursprungliga koden för roboten. Mallmetodmönstret har också använts för att möjliggöra flera olika dekoratörer utan att bryta mot DRY.

T9 C, Command - Inkapsling av metodanrop, frikopplar (decouple) anroparen från objektet som ska utföra handlingen. Alla kommando-objekt instansierar klasser som implementerar ett gemensamt Kommando-interface med en metod execute()

T10 A Den abstrakta klassen `DecoratedRobot` visar ett exempel på hur dekoratörsmönstret (Decorator pattern) har använts för att kunna dynamiskt kunna lägga till extra funktionalitet till en robot som implementerar `Robot`-gränssnittet utan att behöva ändra i den ursprungliga koden för roboten. Mallmetodmönstret har också använts för att möjliggöra flera olika dekoratörer utan att bryta mot DRY. I exemplet utgör `move()` mallmetod och `decorate()` är en skyddad hjälpmetod.

T11 C, Strategimönstret är ett bra mönster då man vill kunna ändra beteendet hos ett objekt under exekvering. Strategimönstret är ett sätt att följa OCP eftersom du kan utöka logiken i någon del av koden (som du öppnat upp) utan att skriva om denna del. Det är lätt att skapa en ny strategi genom att skapa en klass som implementerar strategi-gränssnittet.

T12 C Kompositmönstrets struktur liknar Dekoratörsmönstret men har ett annat syfte. Syftet är att man vill kunna bygga hierarkiska datastrukturer och arbeta med dem som en skilda objekt. Genom att använda kompositmönstret bryter man snarare mot SRP än säkrar att den följs, eftersom komponentsinterfacet tillåts omfatta ansvar både för att hantera strukturen och det ursprungliga ansvarsområdet. Komponenterna i mänstret är av samma typ och har således samma ansvar.

T13 E `DecoratedRobot` är en del av en implementering av dekoratörsmönstret och mallmetodsmönstret, men inte komposit. Den syftar inte till att bygga en hierarkisk struktur av robotar. Inte heller kapslar den in något kommandoanrop.

Lösningsförslag till uppgift 2

```
class GCS {
  Route findRoute(GCS endpoint) {
    return endpoint.routeFrom(this);
  }

  Route routeFrom(GCS start) { // Nytt!
    return new Route(start, this);
  }

  void showOnMap(Map map) { // Nytt!
    map.panTo(this);
  }
}

class NullGCS extends GCS { // Nytt!
  Route routeFrom(GCS start) {
    return new EmptyRoute();
  }

  void showOnMap(Map map) {
    map.panTo(this);
  }
}

class GCSDatabase {
  GCS addressToGCS(String address) {
    if (found) {
      return new GCS(...);
    }
    return new NullGCS(); // Nytt!
  }
}

class Main {
  ...
  Route routeTo(String address) {
    GCS start = gps.getMyPosition();
    GCS endpoint = db.addressToGCS(address);
    Route route = start.findRoute(endpoint);
    route.showOnMap(map); // Nytt!
    return route;
  }
}
```

Rättningsmall:

- Endast lösningar med Null Object för GCS-koordinater godkänns. (Andra varianter -> 0p) NullObjekt av samma typ som GCS 1p
- En lösning där endast en if-sats tagits bort ger delpoäng. (map.panTo borttagen -> -0,5p, ok att endast ta bort den omgivande if-satsen)
- För att kunna ta bort `if`-satsen i findRoute behövs en metod som anropas på slutpunkten (`routeFrom`) eftersom `null`-testet görs på slutpunkten. Nullmetod routeFrom 1p
  - ok att att växla start och mål,
  - fel returtyp -0,5p
- addressToGCS returnerar NullObjekt 1p

Lösningsförslag till uppgift 3

Det cirkulära beroendet kan brytas genom att införa WordOperand i software paketet

```
public class WordOperand implements Operand {
  private hardware.Word word;
  // omissions
}
```

och modifiera Word:

```
public class Word {
  // omissions
}
```

Följande går också bra eftersom WordOperand inte utvidgar någon annan klass

```
public class WordOperand extends hardware.Word implements
    Operand {
  // omissions
}
```

Software:

Mer konkret och mer instabil, mot zone of uselessness - abstrakt fil som ingen är beroende av

- SW A1 = 2/4 (2 abstrakta klasser och 2 konkreta) I1 = 4/5
- SW A2 = 2/5 I2=5/5

Hardware:

Mer stabil men fortfarande konkret, mot zone of pain, konkreta klasser som många är beroende av

- HW A1 = 0/2 Inga abstrakta klasser I1 = 1/5
- HW A2 = 0 I2=0

REP: Samma grad av återanvändning i och målgrupp för ett paket -> gemensam plan för underhåll (konfigurationshantering, uppdatering, testning, release)

CCP: Begränsa påverkan av en viss typ av ändringar till ett (helt) eller några få paket -> analysera användarnas förväntade kravändringar. CCP handlar om att separera klasser som kan förväntas ändras vi olika tillfällen i olika paket.

Det är rimligt att tänka att båda dessa principer följs väl. klasserna som finns i de respektive paketen hör tätt ihop och kan förväntas användas av samma användare mjukvaruutvecklare resp. hårdvaruutvecklare (REP) Det är inte troligt att någon användare endast är intresserad av en delmängd av klasserna i något av paketen. Däremot kan man tänka sig att det inte alltid är samma utvecklare som jobbar med båda paket så det finns en poäng med att konfigurationshantera dem separat. (REP) Man kan förvänta sig olika typer av kravändringar av de två målgrupperna. (CCP) I den givna designen har vi dock ett cirkulärt beroende vilket innebär att det inte går att ändra i något av paketen utan att ändra i båda. Därför skulle de kunna slås ihop i ett paket. Med tillägget av WordOperand kan man å andra sidan tänka sig ändringar av denna (beroende på ändringar av Word) som inte bör påverka övriga delar av paketet (brott mot CCP).

Rättningsmall:

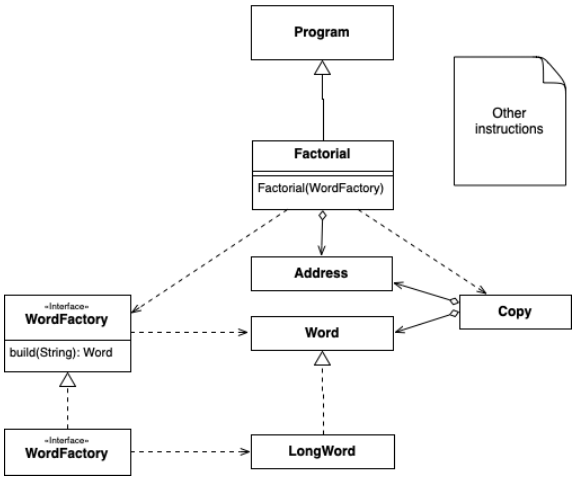
- cirkulärt beroende brutet 1p
- givna klasser kvar i samma paket 1p
- korrekta beräkningar/resonemang kring stabilitet 0,5p och abstraktheten 0,5p
- korrekt resonemang kring förändradrade positioner i AI-grafen HW 1p SW 1p
- korrekta motiveringar sammanhangsprinciper REP 1p och CCP 1p
- vaga resonemang -> -0,5p

Lösningsförslag till uppgift 4

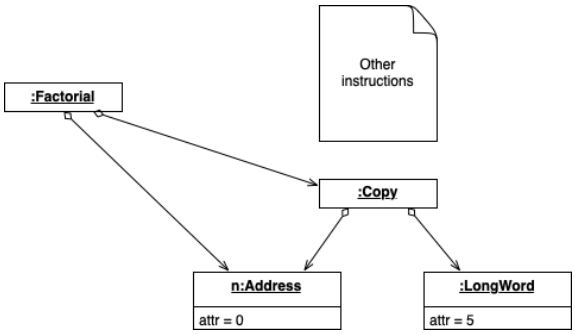
lösning: användning av ordfabrik (Enkel fabrik med interface)

```
public interface WordFactory {
    public Word build(String string);
}
```

```
public class Factorial extends Program {
    public Factorial(WordFactory factory) {
        Address n = new Address(0),
        add(new Copy(factory.build("5"), n));
        // omissions
    }
}
```



Figur 2: klassdiagram Factorial



Figur 3: objektdiagram Factorial

Rättningsmall:

- Interface ordfabrik med byggmetod med strängparameter 1p (fel/saknade parametrar returtyper -0,5p)
- Fabrik som parameter till Factorial 1p
- Anrop av fabrikens byggmetod istf. new, med “5” som parameter -> 1p missar copy -0,5
- korrekt och relevant klassdiagram 2p (småfel -0,5p)
- korrekt objektdiagram 2p (småfel -0,5p)