Mittuniversitetet

MID SWEDEN UNIVERSITY

DT063G, Designmönster med C++

Skriftlig Tentamen

Innehållsförteckning

Tentamensdata....3
Studentdata....3
Tentamensfrågor....4
Uppgift 1 (2p)....4
Svar....4
Uppgift 2 (6p)....5
Svar....5
Uppgift 3 (5p)....6
Svar....6
Uppgift 4 (2p)....7
Svar....7
Uppgift 5 (3p)....8
Svar....8
Uppgift 6 (2p)....9
Svar....9
Uppgift 7 (10p)....10
Redovisning....11

Tentamensdata

Ämne, nivå	Datateknik GR (B)
Kurs	DT063G, Designmönster med C++, 7.5hp
Tidpunkt	Torsdag 18 mars 2021, 09:00 – 15:00

Betygsgränser (poängdistribution)

A	**28-30**	
B	**25-27**	
C	**22-24**	
D	**19-21**	
E	**16-18**	
F(x)	**15**	*Underkänd, kan kompletteras!*
F	**0-14**	*Underkänd!*

Tillåtna Hjälpmedel

Kursmaterial + utförda laborationer!

Studentdata

Namn	
StudentID	
Personnummer	

Tentamensfrågor

Uppgift 1 (2p)

Nedan finner du två implementationer av **Singleton DP**. Beskriv vad som skiljer dem åt, samt när och varför en bör föredras över den andra. ***Dina svar måste motiveras!***

```
class Singleton {
  ... // Attributes
  Singleton() = default;

public:
 Singleton(const Singleton&) = delete;
 void operator=(const Singleton&) = delete;

  static Singleton& getInstance() {
    static Singleton instance;
    return instance;
  }
  ... // Operations
};
```

```
class Singleton {
  ... // Attributes
  static Singleton* instance;
  Singleton() = default;

 public:
  Singleton(const Singleton&) = delete;
  void operator=(const Singleton&) = delete;

  static Singleton* getInstance() {
    if (instance == nullptr)
      instance = new Singleton();
    return instance;
  }
  ... // Operations
};
```

Svar

Innehåll…

Uppgift 2 (6p)

Ange vilket designmönster du skulle tillämpa för varje scenario beskrivet nedan. Varje punkt ger totalt `1` poäng, där korrekt mönster ger `0.5` och en korrekt motivering `0.5` poäng.

a) Byta ut / alternera beteenden under exekvering beroende på aktuella omständigheter.

b) Reducera komplexiteten av ett existerande gränssnitt.

c) Utöka specifikationen av en existerande klass för att stödja ny funktionalitet.

d) Att utifrån en basimplementation av en algoritm låta detaljer kring exekveringsordning variera beroende på situation.

e) Utöka funktionalitet av objekt under exekvering.

f) Du önskar ersätta komplexa villkorssatser med en mer förenklad och skalbar lösning.

Svar

Innehåll…

Uppgift 3 (5p)

Du har blivit kontaktad av den topphemliga organisationen **Spies'R Us** som behöver din hjälp att begränsa hur information delas mellan agenter verksamma i fält. Det finns två typer av agenter; **spion** och **Mästarspion**. Varje spion tillhandahåller en liten del av ett större meddelande, men som är krypterat för att försäkra att ingen av dem förstår innehållet. Mästarspionen är den ende som har nyckel för att dekryptera meddelandet, och en spionhanterare på **Spies'R Us** är den enda som vet i vilken ordning informationens delar skall sättas samman för att meddelandet skall bli komplett.

Din uppgift blir att tillhandahålla en design som uppfyller ovan beskrivna krav på beteende och ansvarsfördelning, och det skall inte spela någon roll hur omfattande ett meddelande kan vara. Hur möter du detta problem och på vilket sätt skulle en lämplig design se ut? **Förklara** och **motivera** ditt tillvägagångssätt och det / de mönster du tillämpar. Bifoga även ett klassdiagram och / eller klassdefinitioner (i kod) som illustrerar din lösning.

Svar

Innehåll…

Uppgift 4 (2p)

Avsluta nedanstående mening med någon av alternativen, samt en kort motivering för detta val. Korrekt val av alternativ ger **1** poäng och tydlig motivering **1** poäng.

Template Method DP liknar **Factory Method DP** eftersom...

a) … de båda definierar abstrakta operationer som ska implementeras i subklasser.

b) … de båda är inblandade i instansiering av objekt.

c) … de båda delegerar ansvar för instansiering av objekt.

Svar

Innehåll…

Uppgift 5 (3p)

Studera klassen `MyComplex` och hur den är implementerad.

```
class MyComplex {
public:
  static MyComplex fromCartesian(double real, double imag) {
    return MyComplex(real, imag);
  }

  static MyComplex fromPolar(double length, double angle) {
    return MyComplex(length*cos(angle), length*sin(angle));
  }

private:
  MyComplex(double a, double b) :x(a), y(b) {}
  double x, y;
};
```

Denna klass är exempel på en variant av ett vanligt designmönster. Vilket av nedanstående alternativ handlar det mest troligt om? Du får enbart välja ett alternativ, och ditt svar måste motiveras!

a) Singleton DP

b) Abstract Factory DP

c) Strategy DP

d) Factory Method DP

Svar

Innehåll…

Uppgift 6 (2p)

Ange vilka två av nedanstående designmönster som tillämpar rekursiv komposition. Notera att du enbart får välja två alternativ, och varje korrekt svar ger **1** poäng. Ingen svarsmotivering är nödvändig!

a) Strategy DP

b) Composite DP

c) Proxy DP

d) Decorator DP

e) Abstract Factory DP

Svar

Innehåll…

Uppgift 7 (10p)

Ange huruvida nedanstående påståenden är korrekt eller ej. Varje rätt svar ger +1p, varje felaktigt svar ger -1p medan varje uteblivet svar ger 0p (minsta möjliga poäng på uppgiften är 0p)!

	Påstående	Svar
a)	Begreppet ***sammanhållning*** relaterar enbart till Objekt Orienterad Programmering.	
b)	En av de främsta anledningarna att föredra komposition före arv är för att undvika ***kombinatorisk explosion*** av antalet nödvändiga klasser.	
c)	Utifrån ett högre designperspektiv så ökar sammanhållningen (cohesion) av en modul om dess ingående entiteter har stark koppling (tight coupling) till varandra.	
d)	Principen ***find what varies and encapsulate it*** kan bli uppfyllt genom att definiera likheter mellan subklasser i en gemensam abstrakt bas.	
e)	Strategy DP kombineras ofta tillsammans med andra mönster för att abstrahera implementationsdetaljer och möjliggöra flexibla designlösningar.	
f)	Composite DP använder både arv och komposition mellan klasser.	
g)	En klassadapter konstrueras i C++ genom multipelt arv från minst två icke-publika basklasser.	
h)	Utökning av det publika gränssnittet i deriverade klasser strider mot **Liskov's Substitution Principle** (LSP).	
i)	**Single Responsibility Principle** (SRP) kan summeras såsom: *det bör finnas enbart en anledning till förändring*.	
j)	Enligt **Interface Segregation Principle** (ISP) så ska en design tillämpa generiska gränssnitt som täcker in ett stort spektrum av möjliga användningsområden.	

Redovisning

Skriv dina svar under avsedda sektioner direkt i tentadokumentet. Vid eventuella problem där detta inte är möjligt, är det accepterat att svaren skrivs i separata textdokument. Lägg då till ditt namn och studentid i varje dokument och relatera tydligt varje svar till respektive fråga / uppgift.

Kodexempel / implementationer kan förläggas till källfiler och eventuella diagram kan sparas i externa bildfiler. Dock måste relationen mellan dina svar och dessa filer tydligt framkomma (inget är underförstått gällande dessa delar).

Alla filer packas till en **zip**-fil som skickas in i avsedd inlämningslåda i Moodle senast angiven deadline. Lycka till!

Genom att du skickar in denna tentamen försäkrar du att alla svar är dina egna, att dina lösningar inte innehåller plagiat från andra källor samt att samarbete med andra studenter inte har förekommit under provtiden!