CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
TDA548	Grundläggande Programvaruutveckling			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
TDA548-0041-KEK		06/01-23	11	5

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	3	
2	X	4	
3	X	6	
4	X	6	
5	X	3	
6	X	6	
7	X	4	Σ32 ✱
8	X	6	
9	X	4	
10	X	5	
11	X	6	
12	–	–	
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng på tentamen		53	

CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
TDA548	Grundläggande Programvaruutveckling			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
TDA548-0041-KEK		06/01-13	11	5

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	3	
2	X	4	
3	X	6	
4	X	6	
5	X	3	
6	X	6	
7	X	4	Σ32 ✱
8	X	6	
9	X	4	
10	X	5	
11	X	6	
12	–	–	
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng på tentamen		53	

CHALMERS | Anonymous code / Anonym kod: TDG48-0041-KEK | Points for question (to be filled in by teacher) / Poäng på uppgiften (ifylles av lärare): 3 | Consecutive page no. / Löpande sid nr: 1 | Question no. / Uppgift nr: 1.

A1.

Ett if-statement består av ett förvillkor, tex $x == 10$, där det efter följande kodblocket endast exekveras om förvillkoret är sant, i mitt exempel om $x = 10$. När det efterföljande kodblocket exekverats, hoppar programflödet till nästa block av kod. Om förvillkoret är falskt, i mitt exempel $x \neq 10$, hoppar programflödet över dess kodblock. Så vi använder if-statements när vi vill ha något beteende baserat på ett förvillkor, tex om en längd av en lista > 0 gör detta, tex om $x = 10$ gör detta osv. /

Inte sällan brukar vi vilja ha ett beteende för olika förvillkor, då används ofta följande struktur:

```
If x < 10:
    /// gör något

Elif x == 10:
    /// gör något

Else:
    /// gör något
```

/

Då har vi ett beteende för alla möjliga värden på x.

3

A2.

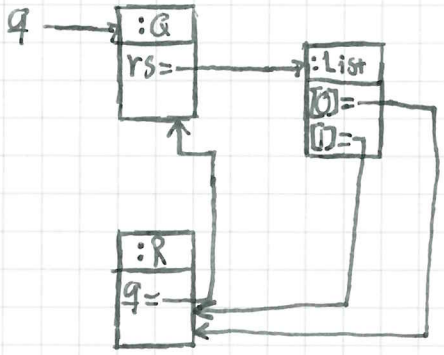
1. x initieras till 6
2. Try-blocket exekveras
3. for-loopen exekveras, i = 0/6, 6 iterationer
4. x ökar med 1 + 0/6 = 1 ⇒ x = 6
5. nästa iteration, i = 1/6
6. x ökar med 1 + 1/4 = 1 ⇒ x = 7
7. nästa iteration, i = 2/6
8. x ökar med 1 + 2/3 = 1 ⇒ x = 8
9. nästa iteration, i = 3/6
10. x ökar med 1 + 3/2 = 2 ⇒ x = 10
11. nästa iteration, i = 4/6
12. x ökar med 1 + 4/1 = 5 ⇒ x = 15
13. nästa iteration, i = 5/6 sista iteration
14. x försöker öka med 1 + 5/0 & division med 0 ⇒ ZeroDivisionError kastas ⇒ try-blocket exekvering slutar
15. Except ZeroDivisonError exekveras ty dess Exception typ matchar det som kastats i try-blocket.
16. x ökar med 7, x = 22
17. Bara ett except block kan exekveras ⇒ programflödet till sista rad
18. Print(x) exekveras, 22 printas.

4

4

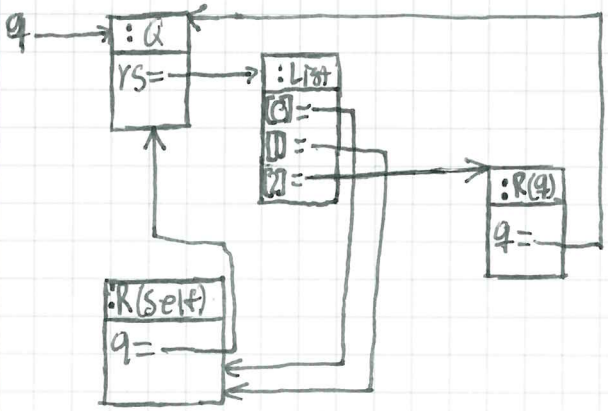
A3

Innan anrop av do_it(q):



Notera att r i class Q är en lokal variabel som anropar R-konstruktor => ett objekt av typ R skapas som tar som argument "self" vilket är en referens till det skapade Q-objektet.

Efter anrop av do_it(q)



Så anropet av do_it(q) kommer lägga till ett nytt element till Listan som q.rs håller referens till. Det nya elementet kommer hålla en referens till ett nytt objekt R(q). Notera att R(self) är samma objekt från innan do_it(q) anropet.

6

A4.

kod i python:

```
def Products(in_list):
    Final_list = []
    for i in range(len(in_list)):
        Elem = in_list[i]
        if i > 0:
            Product = Elem * in_list[i-1]
            Final_list.append(Product)
    return Final_list:
```

Förklaring: Tom lista skapas. for itererar för varje index i längden av listan. Elem skrivs för varje iteration till värdet på det index i listan vi är på. if exekveras om vi ej är på index 0 ⇒ vi undviker indexeringerror när vi kollar bakåt i listan. Product skrivs till produkten av värdet i listan på index i och värdet på elementet innan i listan. Final_list, läggar till product till listan.

6

A6:

Konstruktorn tillhör klassobjektet och är den "maskin" som kan skapa instanser av den klass som klassobjektet tillhör. Vi använder en konstruktor för att kunna skapa flera objekt av samma typ, dvs instanser, och det är även i konstruktorn som vi anger instansattributen, vilka ger instanserna unika värden/tillstånd. /1

Vi kan även använda en konstruktor för låta instanserna ärva attribut från en supertyp, genom att ange keyword super. /1

Det är först vid anrop av konstruktorn som instanserna skapas.

om ingen konstruktor deklareras, används en inbyggd default-konstruktorn, men då kan vi givetvis inte skapa instanser med unika värden och attribut som vi bestämt.

Ex, Låt säga vi har en klass player och vi vill ha flera spelare, då kan en konstruktor se ut enligt:

```
def Player(int points, String name)
    self.Points = Points
    self.nam = name.
```

Då kan vi anropa denna konstruktor och skapa instansobjekt av Player med unika namn och points. Tex Player(Max, 60) eller Player(Nicklas, 69). /1

3

CHALMERS | Anonymous code / Anonym kod: TDA648-0041-KEK | Points for question (to be filled in by teacher) / Poäng på uppgiften (ifylles av lärare): 6 | Consecutive page no. / Löpande sid nr 6 | Question no. / Uppgift nr 6.

A6

Abstraktion innebär att lyfta fram relevanta aspekter av komponenter, och dölja intern implementation/information. Detta för att öka förståelsen för koden. 1

Fördelen med abstraktion är att man åstadkommer svagare beroenden mellan komponenter i koden, då de ej beror på interna implementations-detaljer hos varandra. Detta gör att vi enkelare kan ändra komponenter utan att påverka andra delar i koden. /

Utvecklare kan åstadkomma abstraktion med inkapsling, dvs selektivt dölja metoder och attribut inuti objekt. /

Väl genomtänkta och begränsade gränssnitt åstadkommer också abstraktion, genom att selektiv exponera väl vald funktionalitet som är relevanta för respektive komponent. 1

Väl vald namngivning och funktionell nedbrytning åstadkommer också abstraktion, då koden blir mer lättläst. 1

Ett exempel på etisk konsekvens är att koden blir mindre modulär, dvs att olika komponenter beror på interna detaljer hos varandra. Detta leder till problem för både kund och framtida programmerare som ska underhålla koden, då det är svårt att underhålla samt vidareutveckla koden.

Ytterliggare exempel på etisk konsekvens är såklart att koden blir svår att förstå, vilket leder till problem för alla som ska interagera med koden, till och med utvecklaren själv i längden, genom att strukturen och dynamiken är svårförståelig, vilket i sin tur gör koden svår att använda, underhålla och vidareutveckla. 1

6

4

A7.

En abstrakt metod är en deklaration av en metodsignatur och 1
dekorerad med @abstractmethod (i python), men utan någon tillhörande
implementation. En abstrakt metod är alltså en ren komponent
av ett gränssnitt. 1

En abstrakt metod måste alltså implementeras i en subtyp för att
kunna användas. 1

Vi vill använda abstrakta metoder för att definiera en typ genom
ett "interface", där enbart metodsignaturer deklarerats. En typ definieras
just av de publika metoder som går att anropa på ett objekt av typen,
men inte hur dessa implementeras, vilket är just vad en abstrakt metod
består av. Dessa implementationer kan sedan se annorlunda ut för
olika subtyper, som implementerar interfacet, vilket ger upphov till
olika beteenden för samma typ. 1

4

CHALMERS | Anonymous code / Anonym kod: TDA548-0041-KEK | Points for question (to be filled in by teacher) / Poäng på uppgiften (ifylles av lärare): 6 | Consecutive page no. / Löpande sid nr: 4 | Question no. / Uppgift nr: 8

B1.

1. Ett klassobjekt X skapas
2. Ett klassobjekt Y skapas, med ett klassattribut x_1 som anropar X - konstruktorn. konstruktorn exekveras och printar "static in Y". /
3. Ett klassobjekt Z skapas, med ett klassattribut x_1, som anropar X - konstruktorn. X - konstruktorn exekveras och printar "static in Z". /
4. uppgiftB1() anropas.
5. Z initieras till ett anrop av Z - konstruktor med argumen "Run1".
6. Z - konstruktorn exekveras. Ett instansobjekt av Z skapas med instans attribut x_2 som initieras med ett anrop till X - konstruktorn ⇒ "Instance in Z" printas. /
   b1 initieras till anrop av Y - konstruktorn ⇒ instansobjekt av typ Y skapas, med ett instansattribut x_2 som initieras med anrop till X - konstruktor ⇒ "Instance in Y" printas. /
7. Sedan anropas print i Z-objektet ⇒ "Z is done - Run1" printas. /
8. Rad 3, Z initieras med ett anrop till Z - konstruktorn med argumentet "Run 2". Nytt instansobjekt av typ Z skapas, med ett instans attribut x_2, vilket anropar X - konstruktorn ⇒ "Instance in Z" printas.
   Samma instansobjekt av typ Z har även ett instansattribut b_1 som initieras med ett anrop till Y - konstruktor ⇒ nytt instansobjekt av typ Y skapas med ett attribut x_2, som initieras med ett anrop till X - konstruktorn ⇒ "Instance in Y" printas
9. Sedan exekveras print i Z-Objektet ⇒ "Z is done - Run 2" printas. /

6

B2

När vi deklarerar en klass C händer följande:

1. Vi definierar en typ C, vars gränssnitt består av de publika metoder och attribut som deklarerats i klassen. 1
2. Vi deklarerar ett klassobjekt, med namn C, vilken är av typen C. Dess beteende definieras av de metoder och attribut som deklarerats tillhöra klassen. 1
3. Vi definierar att instansobjekt av klassen har typen C, med ett beteende som definieras av publika instansmetoder deklarerade i klassen, dvs de metoder som inte markerats tillhöra klassen. 1
4. Vi definierar hur instansobjekt kan skapas av klassen, genom implementation av en konstruktor. 1

Alltså är en klass som en modul med en sammansättning av olika data och beteenden som antingen tillhör klassobjektet eller instansobjekt. Notera att det är skillnad mellan ett klassobjekt och en klass, klassobjektet representerar klassen vid körning men klassen finns alltid kvar med alla metoddefinitioner och data.

4

CHALMERS	Anonymous code	Points for question (to be filled in by teacher)	Consecutive page no. Löpande sid nr 10
	Anonym kod TDA548 – 0041 – KEK	Poäng på uppgiften (ifylles av lärare) 5	Question no. Uppgift nr 10

B3.

En datastruktur en en struktur för lagring av data. Valet av datastruktur är beroende av hur vi vill lagra och använda den data vi behandlar. En datastruktur är alltså en abstrakt beskrivning till skillnad från datatyper.

Exempel på datorstrukturer: List, tupler, dictionaries.

Exempel på val av datastruktur:

Om vi tex vill kunna ändra ordningen av elementen i en struktur, bör vi välja en List som datastruktur.

Men om vi istället vill lagra data i en struktur där elementen placeras i ökande ordning, bör vi istället använda en tupel.

Dvs, valet av datastruktur är viktigt då valet är beroende av hur vi vill hantera och använda elementen i strukturen.

5

CHALMERS | Anonymous code / Anonym kod: TDA548-0041-KEK | Points for question (to be filled in by teacher) / Poäng på uppgiften (ifylles av lärare): 6 | Consecutive page no. / Löpande sid nr: 11 | Question no. / Uppgift nr: 11

B4.

Komposition handlar om flytta gemensam funktionalitet som flera klasser delar, till en hjälpklass. Man låter sedan de klasser som delar denna funktionalitet ha ett attribut som håller en instans av hjälpklassen. Klasserna kan sedan återanvända kod genom att referera till funktionaliteten i hjälpklassen. På detta sätt återanvänds kod över flera klasser.

Medans arv används om klasserna har samma grund, sedan specialiserade till en viss grad. Då låter man den gemensamma koden deklareras i en superklass, denna funktionalitet ärv sedan av subklasser. Notera att subklasserna ärver samtlig funktionalitet från superklassen. Subklasserna kan själva sedan definiera ytterliggare funktionalitet eller omimplementera ärvd funktionalitet (Override). Arv gör koden mer dynamisk och återanvänder kod.

Nackdelen med arv är att superklassen och subklassen har en "is-a" relation, tex så kan Triangle vara en subklass till polygon, detta medför att om vi vill modifiera en subklass kommer vi att behöva ändra superklassen med, vilket i sin tur förmodligen gör att vi måste ändra övriga subklasser också då de står på samma grund. Arvs nackdel är kompositions fördel, då vi vid komposition enbart behöver byta ut attributet som håller instansen till en hjälpklass, om vi vill modifiera en specifik klass. Komposition erbjuder alltså en mer flexibel relation till övriga klasser, vilket underlättar vidareutveckling och underhåll av koden.

6