Tentamen

Data- och informationsvetenskap:
Objektorienterad programmering och modellering för IA (DA361A)
Teknik och samhälle, Malmö högskola

2017-12-15

Hjälpmedel

Penna och papper.

Anvisningar

Tentamen omfattar 14 uppgifter.
Maxpoäng är 38.5, följande gränser finns:
Godkänt: 20p, **Väl godkänt:** 30p

Markera varje sida med dina initialer, samt sidnummer. Skriv läsligt och kommentera utförligt vad du gör, det kan ge dig poäng även om resultatet är fel.

Glöm inte att fylla i försättsbladet, inklusive att kryssa i vilka uppgifter du svarat på.

Jag kommer att komma förbi två gånger under tentamen för att svara på ev. frågor, den första gången cirka **09.00** och den andra gången cirka **10.30**.

Lycka till!

Uppgifter

Del 1 - Kortare frågor, här förväntas ni ge kortfattade svar på frågorna. (14p)

1. Förklara ”instans” och ”klass” och hur de relaterar till varandra inom området objektorienterad programmering. (1p)

2. Vad innebär attribut och metoder inom området objektorienterad programmering? (1p)

3. Vad är *PEP* (och mer specifikt *PEP8*), och hur påverkar det er programmering i ”vardagen”? (2p)

4. Vad är en konstruktor inom objektorienterad programmering? När behöver man använda sig utav en sådan, och när behöver man inte det? (2p)

5. Vad är det för skillnad mellan metoder, klassmetoder och statiska metoder i en klass? (2p)

6. I Python skickar man med *self* som parameter i alla metoder i en klass, varför gör man detta? (2p)

7. Skapa en valfri klass i UML med minst två attribut och två metoder med tillhörande Python-kod. Beskriv sedan hur klassdiagrammet relaterar till Python-koden. (2p)

8. Beskriv med ord eller visa med ett kodexempel för vad termen “polymorphism” har för innebörd inom objektorientering.? (2p)

Del 2 – Fokus källkod & UML (16.5p)

9. Vi har i kursen pratat om *inkapsling* (eng. *encapsulation*).
    a. Vad innebär inkapsling? (1p)
    b. Ge ett konkret exempel på hur detta ser ut i UML (klassdiagram) och i Python-kod. En klass räcker. (3p)

10. Beskriv begreppen *arv, aggregation*, *komposition* och *association g*enom ett konkret exempel för varje begrepp, i ord och UML. (4p)

11. I bilaga 1 ser ni klasserna Person, Student, StaffMember, Lecturer och Course. Skriv Python-kod för att utföra följande uppgifter. (2.5p)
    a. Skapa kursen ”Objektorienterad programmering”
    b. Skapa en student
    c. Skapa en lärare
    d. Ange att studenten går kursen ”Objektorienterad programmering”
    e. Ange att läraren är delaktig i kursen ”Objektorienterad programmering”

11. Utifrån bilaga 1, skapa ett klassdiagram med tillhörande relationer och kardinalitet. (4p)

12. Utifrån bilaga 1, skapa ett sekvensdiagram för att lägga till en student på en kurs (2p)

Del 3 - Diskuterande frågor, här förväntas ni föra lite längre resonemang och diskussion kring frågorna. (8p)

13. Ni har i denna kurs använt er av objektorienterad programmering (till skillnad från procedurell/ strukturerad programmering som ni tidigare använt). Diskutera för- och nackdelar kring dessa olika sätt att programmera, samt när det är lämpligt att använda de olika tillvägagångssätten. (4p)

14. Du har fått som uppgift att introducera UML på ett företag. Argumentera för varför / varför inte företaget skulle tjäna på detta. Vilka konkreta för- och nackdelar skulle det innebära? (4p)

Bilaga 1.

```
class Person:
    def __init__(self, name, surname, computer_id):
        self.name = name
        self.surname = surname
        self.computer_id = computer_id


class Student(Person):

    def __init__(self, *args, **kwargs):
        self.classes = []
        super(Student, self).__init__(*args, **kwargs)

    def enrol(self, course):
        self.classes.append(course)
        course.add_student(self)


class StaffMember(Person):

    def __init__(self, academic_title, *args, **kwargs):
        self.academic_title = academic_title
        super(StaffMember, self).__init__(*args, **kwargs)


class Lecturer(StaffMember):
    def __init__(self, *args, **kwargs):
        self.courses_taught = []
        super(Lecturer, self).__init__(*args, **kwargs)

    def assign_teaching(self, course):
        self.courses_taught.append(course)
        course.add_lecturer(self)

class Course:
    def __init__(self, name, code):
        self.name = name
        self.code = code
        self.students = []
        self.lecturers = []

    def add_student(self, student):
        self.students.append(student)

    def add_lecturer(self, lecturer):
        self.lecturers.append(lecturer)
```