a) olion luominen, mitä vaiheita olion luomiseen sisältyy (1p)

1. Luo luokka (Nimeä se esim. Auto)
2. Luo oliolle jäsenmuuttujia (
    - private int max_speed = 10;
    - private int speed;)
3. Luo oliolle ominaisuuksia (
    - public void Speed(){
    - get{return value;}
    - set{(if value => max_speed)speed=max_speed}
    - else{ speed = value}

    }
4. Luo oliolle konstruktori/konstruktoreita tai älä(default)
    Public Auto(){}
    Public Auto(int vauhti){
    speed = vauhti;
    }
5. Luo oliolle methodeja esim:
    - Public int Accelerate(){
    - Speed += 1

    }
6. Luo oliolle ideksoijia (Jos tarve)
7. Tuhoa olio
    - ~Auto(){ConsoleWriteLine(”Auto on tuhottu”)}

Jäsenmuuttujia ja ominaisuuksia määrittäessä on muistettava määrittää suojaus taso(public, private, protected, internal, protected internal)

b) perintä ja sen merkitys olio-ohjelmoinnissa (1p)

Perintä on oleellinen osa olio-ohjelmointia. Voidaan hyödyntää jonkin luokan ominaisuuksia aliluokasssa. Esimerkiksi jos meillä olisi Luokka ”Ihminen” niin ihmis luokassa olisi ominaisuuksina ikä, paino, pituus, jne. Tämän jälkeen voitaisiin luoda luokat ”Oppilas” ja ”Opettaja”. Oppilaalla ja Opettajalla on ihmisen perus ominaisuudet(toivottavasti) ja voidaan hyödyntää tätä perimällä yläluokasta(”Ihminen”). Tapahtuu näin:

Class Oppilas : Ihminen

Perimällä meidän ei tarvitse uudelleen määrittää samoja asioita saman tyyppisten olioitten kanssa. Ideana on että voimme hyödyntää tiettyjä asioita uudellleen.

Opettajalla on esimerkiksi aliluokassa ominaisuutena kellokortti ja kulkukortti, Oppilaalla on ominaisuutena masennus.

c) abstrakti-luokka (1p)

Liittyi periytyvyyteen, mutta nyt en kyllä muista valitettavasti.

d) mitä on kapselointi, ja millä tavalla voit toteuttaa sitä (1p)

Kapseloinnilla lukitaan olion jäsenmuuttujat tai ominaisuudet niin että oliota ei voida väärinkäyttää/rikkoa. Suojaamme jäsenmuuttujat esimerkiksi private suojaustasolla, milloin muuttujan arvoa ei voida muuttaa muualla kuin kyseisessä luokassa. Muodostamalla aksessorin public suojaustasolla, voimme muuttaa jäsenmuuttujan arvoa meidän määrittelemiin arvoihin. Esim:

```
//jäsenmuutujat
- private int max_speed = 1000;
- private int speed;)

// aksessori(ominaisuus)
- public void Speed(){
- get{return value;}
- set{(if value => max_speed)speed=max_speed
- else{ speed = value}
  }

//methodi
- Public int Accelerate(){
- speed += 100
```

Näin ollen pääohjelmassa kun annetaan autolle nopeus tai autoa kiihdytetään niin jäsenmuuttujan arvo ei mene ”rikki” vaan arvot pysyvät asetuissa rajoissa.

e) kuinka toteutat C#-ohjelmassa poikkeuksien käsittelyä ja mitä merkitystä sillä on ohjelmalle (1p)

Poikkeuksien käsittely on oleellinen osa ohjelmointia, on olemassa kääntämisen, ajon, sisäisten poikkeuksien hallintaa. Poikkeuksien hallinnalla voimme pitää huolen siitä, ettei ohjelma kaadu(”Cräshää”) tai tuota ei haluttuja toimenpiteitä. Hyvänä esimerkkinä toimii numerolla 0 jakaminen mikä aiheuttaa ohjelman kaatumisen, jos laskutoimitukselle ei ole tehty poikkeuksien käsittelyä. Käsittely toteutetaan ensin kokeilemalla ”try”, sitten ”catch” eli jos tapahtuu poikkeus ja lopuksi ”finally” eli mitä tapahtuu joka tapauksessa olipa exceptionia tai ei.

```
try:

result = x/y

catch (DividedByZero):

if (x == 0 || y == 0) result = 0;

finally:

ConsoleWriteLine(“Result is {0}”,result)
```