Week 3:

1a: Dit komt voor bij de methodes aantalArtikelen() en hoeveelheidGeldInKassa().

2a: Het is handig om de methodes private te maken omdat dan alleen de klasse waarin die methode gemaakt is de gegevens zal gebruiken. Wanneer je dus iets verandert aan de methodes, zal dit geen invloed hebben op andere klasses. Dit maakt de code een stuk overzichtelijker.

2b: In een HashMap heb je key's en value's. In HashSet heb je alleen value's.

3a: De constructor maakt als eerst een nieuwe Kantine en een nieuwe Random aan. Hierna wordt er een array gemaakt met AANTAL\_ARTIKELEN waardes die allemaal tussen MIN\_ARTIKELEN\_PER\_SOORT en MAX\_ARTIKELEN\_PER\_SOORT liggen. Hierna wordt er een KantineAanbod gemaakt waarbij met de artikelnamen, artikelprijzen en hoeveelheden meegeeft. Tot slot wordt het kantineaanbod geset voor een Kantine

3b: Aangezien er +min achter staat, wordt de waarde opgehoogd met het minimum. Normaal is de minimum waarde 0. Dus nu is de minimum waarde 0 + min = min.

Het maximum komt tussen haakjes te staan. Maar omdat er later nog weer de minimum waarde bij opgeteld wordt, moet de minimum waarde hier weer van het maximum afgehaald worden.

Normaal zou nextInt TOT de ingevoerde value gaan. Maar in dit geval gaat hij tot en met de ingevoerde value gaan. Door de +1.

Week 4:

1c: Er hoeven geen waarden mee te worden gegeven aan het object. Een constructor is in dit geval dus niet nodig. Het enige wat we nodig hebben zijn de methodes.

1d: Deze 2 methode’s voeren niks uit op een instantie van de klasse Administratie. Ze voeren alleen een berekening uit op ingevoerde gegevens en retourneren de uitkomst.

1e: Nu de constructor private is kan men niet meer van ‘buiten af’ een nieuwe instantie van de klasse Administratie aanmaken.

1i: Als het niet static is, hoeft het ook niet overal hetzelfde te zijn. Dus kan hij niet final worden omdat het eventueel nog gewijzigd zou kunnen worden.

1j: Nu zou het nog weer aangepast kunnen worden, wat we niet willen.

2b: Omdat een super aanroep belangrijke info zou kunnen geven of aanpassen. Dit moet dus altijd als eerste gebeuren!

Week 5:

1c: In ieder geval onder deze omstandigheden is er geen verschil tussen het gebruik van .equals() of ==.

1d: In dit geval moet je .equals() gebruiken. == kijkt namelijk alleen of je 2 keer naar hetzelfde refereert. .equals() kijkt ook daadwerkelijk of de inhoud gelijk is.

2c: Om te zorgen dat alleen de subklasses het saldo kunnen benaderen. Je wil niet dat andere klassen het saldo ook kunnen benaderen.

5a: Nee, een interface heeft geen methodes en constructor met body. Ze zijn leeg. Ze zullen eerst geïmplementeerd moeten worden door een andere klasse

5b: Nee, een abstracte klasse heeft ook methoden die nog geïmplementeerd moeten worden

5c: Nee

5d: Ja

5e: Ja

5f: Ja, elke methode in een interface heeft geen body. Een abstracte methode is een methode zonder body.

5g: Ja, als een methode abstract is, moet deze eerst geïmplementeerd worden door een andere klasse.

5h: Polymorfisme is de mogelijkheid te zorgen dat een object meerdere vormen kan hebben.

Je hebt een abstracte klasse dier. En een klasse hond en een klasse kat. Een kat en een hond lopen hetzelfde dus de methode loop() in Dier heeft al een body. Elk dier maakt wel een ander geluid. Dus de klasse maakGeluid() in Dier moet nog geïmplementeerd worden. In de klasse Hond zal deze methode iets doen als return “Woef” en in de klasse kat iets als return “Miauw”.

Kijk ook maar eens naar de zojuist gemaakte interface KortingskaartHouder. Dit interface wordt geïmplementeerd door Docent of KantineMedewerker. In beide gevallen reageert de interface KortingskaartHouder dus anders.

6a: Het is wel mogelijk, maar het zou niet logisch zijn. Over het algemeen heeft een abstracte klasse minimaal 1 abstracte methode die nog geïmplementeerd moet worden.

6b: Dit hoeft niet, je zou ook 2 sub klassen kunnen hebben die allebei een deel van de methodes van de abstracte super klasse implementeren.

6c: Als je een foutmelding krijgt moet je zorgen dat de methode geimplementeerd wordt door een andere klasse. Zodat deze methode toch een body heeft.

6d: Een instantie variabele kan niet abstract zijn, omdat een instantie variabele een instantie is van een klasse. Deze kan niet geimplementeerd worden door een andere subklasse en kan dus niet abstract zijn.

6e: Finale methoden kunnen niet extend worden. Als je een methode dus Final zou maken, en abstract, heb je dus een methode zonder body die niet geimplementeerd kan worden door een andere mehtode.