**Week 1**

Opdracht 1b:

Declaratie: het aanmaken van een variabele, het aangeven van het type en de naam van de variabele, zonder deze een waarde toe te kennen (deze wordt standaard null, 0 of false, afhankelijk van het type)

Initialisatie: de eerste toekenning van een waarde aan een variabele

Opdracht 4:

Het verschil waarom de verwijzing van j naar i bij het primitieve type wel gelijk blijft maar voor het object niet zit hem in het feit dat je met een primitief type feitelijk een kopie maakt van de parameter (dit is een directe opslag van de waarde in het ‘vakje’ van de variabele met het primitieve datatpe), terwijl je bij het object met zowel a als b een verwijzing naar hetzelfde object maakt. De waarde van het object wordt dus niet rechtstreeks in de variabele opgeslagen, dit is slechts een referentie naar het object.

**Week 2**

Opdracht 5a:

Het is handiger deze rij te bewerken met een while lus dan met een for lus omdat je dan makkelijker gebruik kan maken van een iterator. Je kan de while lus laten lopen terwijl de iterator een volgende waarde heeft (iterator. hasNext()). Met een for lus kan dit ook maar is het minder handig. Met een foreach lus kan het niet omdat een ArrayList niet bewerkt kan worden terwijl het ‘doorlopen’ wordt.

**Week 3**

Opdracht 1a:

Dat methodes dubbel voorkomen komt bij de klassen Checkout en Cafetaria voor bij de methodes:

**Cafetaria:**

amountOfMoneyInCheckout(): Bij deze methode wordt enkel een getter van checkout aangeroepen en gereturnd.

numOfArticlesPassed(): Ook bij deze methode wordt enkel een getter van checkout aangeroepen en gereturnd.

resetCheckout(): Bij deze methode worden 2 setters van checkout aangeroepen.

Al deze drie methodes zouden veel eenvoudiger vervangen kunnen worden door bij de klasse Cafetaria een methode getCheckout() toe te voegen. Als je later dan nieuwe methodes toevoegt aan de klasse checkout hoef je niet nieuwe ‘doorgeefluiken’ te maken bij de klasse cafetaria. Je kan ze eenvoudigweg aan roepen door de getter te gebruiken.

Opdracht 2a:

Dit is een goed idee omdat dit de encapsulatie bevordert. Andere klassen hebben niks aan deze methodes, dit maakt het alleen maar onoverzichtelijk. Het is beter dit af te schermen en de klasse voor 1 belangrijk doel te gebruiken.

Opdracht 2b:

Je gebruikt een HashSet wanneer je een collectie wil van Objecten die uit items bestaat die uniek zijn en allen maximaal 1 keer voor mogen komen. Je gebruikt een HashMap wanneer objecten ook vaker voor mogen komen. Een belangrijk verschil is dat de HashMap een ‘sleutel’ heeft. Je kan Objecten een ander Object meegeven die als sleutel dient. Hierdoor is het makkelijker bepaalde objecten terug te vragen.

Opdracht 3a:

cafetaria = new Cafetaria();

Dit maakt een nieuwe instantie van de klasse Cafetaria aan en koppelt een verwijzing er naar in de variabele cafetaria.

random = new Random();

Dit maakt een nieuwe instantie van de klasse Random aan en koppelt een verwijzing er naar in de variabele random. Van de klasse Random worden later methodes aangeroepen om pseudo-willekeurige getallen te genereren.

int[] amounts = getRandomAray(NUMBER\_OF\_ARTICLES, MIN\_ARTICLES\_PER\_TYPE, MAX\_ARTICLES\_PER\_TYPE);

Dit maakt een nieuw integer array aan die de waarde heeft die gereturnd wordt door de methode getRandomAray. Deze methode maakt een array van de gespecificeerde lengte en vult deze met random getallen binnen de gegeven limieten.

cafetariaOffer = new CafetariaOffer(articleNames,articlePrices,amounts);

Dit maakt een nieuwe instantie van de klasse CafetariaOffer aan en koppelt een verwijzing er naar in de variabele cafetariaOffer. Hij geeft als parameters arrays door die vervolgens als aanbod worden opgenomen.

cafetaria.setCafetariaOffer(cafetariaOffer);

Dit zorgt ervoor dat het zojuist gecreëerde CafetariaOffer object in het veld hiervoor in het object van de klasse Cafetaria geset wordt.

Opdracht 3b:

Deze methode geeft een waarde terug die aan de volgende eisen voldoet: groter of gelijk aan min, kleiner of gelijk aan max. Dit wordt gedaan door een pseudo-random getal tussen 0 en max-min+1 te genereren, dit wordt zo gedaan omdat de nextInt methode begint bij nul (inclusief) en alle getallen kan kiezen tot de gegeven variabele (exclusief). Om er voor te zorgen dat je ook de max kan krijgen geef je +1.

**Week 4**

Opdracht 1C:

Er is geen constructor aanwezig, als er geen constructor aanwezig is definieert java automatisch een constructor zonder parameters. Daarom kan je alsnog new Administratie() aanroepen.

Opdracht 1D:

Deze methoden kunnen static zijn omdat er geen instantie van de klasse Administratie nodig is om deze methode te gebruiken. Er zijn namelijk geen veld variabelen waarvan deze methoden gebruik maakt. Hij verwerkt alleen input en return iets wat die met de methode gedaan heeft.

Opdracht 1E:

Een private constructor aanmaken gaat tegen dat je dit kan doen omdat er dan wel een constructor gedifinieerd is. Java gaat nu niet automatisch meer een ‘lege’ constructor maken. Hierdoor kan de constructor niet worden aangeroepen en dit voorkomt dat je een nieuwe instantie van deze klasse kan maken.

Opdracht 1G:

Zie bijgevoegde afbeelding.

Opdracht 1H:

Final zorgt ervoor dat de variabele na initialisatie niet meer een andere waarde kan krijgen. Hierdoor blijft de waarde altijd gelijk.

Opdracht 1I:

De compiler klaagt hierover omdat de methode die deze variabele gebruikt static is. Er is dus geen object van deze klasse wanneer deze methode aangeroepen wordt. Omdat de variabele niet static is kan deze pas gebruikt worden als er een object van de klasse gemaakt is. Bij deze methode is dat niet het geval. Daarom klaagt de compiler hierover.

Opdracht 1J:

Het probleem is dat de variabele aangepast kan worden. Hij kan wel gebruikt worden wanneer er geen object van de klasse is. Maar hij kan worden aangepast. Hierdoor kan het aantal dagen zomaar veranderd worden. Dit zou foute resultaten opleveren.

Opdracht 2B:

Je moet altijd beginnen met een aanroep naar super() omdat voor je iets anders gaat doen het object van de super klasse goed moet worden klaargezet. Anders kan je eventueel methodes van de superclass aanroepen die niet valide zijn als het object nog niet goed is geprepareerd. Daarom moet je in een subclass altijd beginnen met een call naar super.