Mogelijke theorievragen OGP

Basisconcepten Java

Waarom moet elke variabele vooraf gedeclareerd worden?
Zodat er geheugenplaats gereserveerd wordt voor deze variabele.

Wat zijn de voorwaarden voor de naam van een variabele?
Uitsluitend letters, cijfers en _. Daarbovenop moet de naam beginnen met een letter of _ en mag de naam niet tot de lijst van gereserveerde woorden behoren. De naam moet ook betekenisvol zijn.

Hoe kan je speciale tekens zoals ‘, “ en \ printen?
Door er een \ voor te zetten. Vb: ‘\’’, ‘\”’ en ‘\\’

Wanneer gebruiken we long?
Indien het bereik van int ontoereikend is, dan voegen we ook L toe achter het getal.

Wanneer gebruiken we byte of short?
Indien er op geheugen bespaard moet worden.

Wat is de prioriteit bij logische operatoren?
!, && en dan pas ||.

Hoe werkt lazy evaluation?
Bij &&: als de eerste voorwaarde false is, wordt niet verder getest en geeft men false terug.
Indien de eerste voorwaarde true is, zal de tweede voorwaarde het resultaat bepalen.
Bij ||: als de eerste voorwaarde true is, wordt niet verder getest en geeft men true terug.
Indien de eerste voorwaarde false is, zal de tweede voorwaarde het resultaat bepalen.

Hoe bekomen we een random geheel getal uit het interval [a,b] met de klasse Math?
int getal = a + (int)((b-a+1)*Math.random());

Basisbegrippen OGP

Welke karakteristieken heeft een object?
Identiteit: unieke code. Toestand: geheugeninhoud van het object (variabelen).
Gedrag: diensten die het object levert (methodes).

Als wat kan een klasse dienst doen?
Type van een object, fabriek: om objecten te maken, aanbieder van diensten (static meth.)

Wat is data hiding?
Toestand object is niet direct toegankelijk voor de gebruiker, we verbergen de implementatiedetails.

Bestaande klassen gebruiken

Indien we de string “Dit iS een ZiN” volledig in hoofdletters willen zetten,
moeten we dan s = s.toUpperCase() of s.toUpperCase() doen en waarom?
s = s.toUpperCase(), want een string is onveranderlijk, we moeten de inhoud overschrijven.

Indien we strings moeten aanpassen of ontzettend veel concateneren, hoe doen we dat?
Met de klasse StringBuilder.

Wat is boxing?
Boxing is de automatische conversie van primitieve waarde naar wrapper-object.

Wat is unboxing?
Unboxing is de automatische conversie van wrapper-object naar primitieve waarde.

Wat is het verschil tussen de next() en nextLine() methode bij een Scanner?
next() slaat alle whitespaces over en leest dan één woord in tot aan de volgende whitespace.
nextLine() leest (de rest van) één lijn in, met alle whitespaces.

Hoe zorgen we er voor dat de Scanner altijd afgesloten is?
Bij het openen van een scanner, altijd als laatste *variabele.*close() doen.
Of een try-block schrijven: try (Scanner sc = new Scanner(new File(*naam*))) { (…) }

Zelf klassen maken

Wat bevat een klasse net?
Data (instantievariabelen/attributen). Methoden (o.a. de constructoren).

Hoe kunnen we private attributen opvragen van buiten de klasse?
Door middel van een getter.

Wat is er net zo speciaal aan een constructor?
Heeft dezelfde naam als de klasse, geen returntype, wordt automatisch uitgevoerd bij *new* en zorgt voor de initialisatie van het object.

Wat zijn de verschillende onderdelen van een methode (bij declaratie) (signatuur)?
[modificatoren (private,public,static …)]
type teruggeefwaarde
naam van de methode
[lijst van parameters (aantal, volgorde, type, NIET naam)]
[excepties die de methode kan opwerpen]

Hoe zit het nu net met objectparameters en aanpassingen daaraan?
Primitieve types zullen een kopie meegeven en het origineel zal dus niet aangepast worden.
Bij referentietypes zal een referentie meegegeven worden waardoor het origineel aangepast zal kunnen worden.

Collections

Wat zijn de voor- en nadelen aan een array?
Voordelen: gemakkelijk en zeer efficiënt. Nadelen: grootte moet gekend zijn bij creatie.

Wat is er zo typerend aan een HashSet?
In een HashSet kan elk element slechts één keer voorkomen!

Wat is er zo typerend aan een TreeSet?
In een TreeSet kan elk element slechts één keer voorkomen! Alle elementen zitten gesorteerd!

Overerving

Wat erft een afgeleide klasse NIET van de bovenklasse?
Deze erft de private methodes niet.

We zijn geen fan van protected attributen uit de bovenklasse, hoe lossen we dit op?
We voorzien voor elk attribuut dat aangepast/opgevraagd moet kunnen worden een getter/setter.

Wat is het verschil tussen method overriding en method overloading?
Bij method overriding gaan we een methode uit de basisklasse in de afgeleide klasse opnieuw definiëren en eigenlijk overschrijven. Bij method overloading zitten we met twee of meer methodes met dezelfde naam, maar verschillende signatuur.
Bij method overriding bevinden de methodes zich in verschillende klassen, bij method overloading in dezelfde klasse.

Wat zijn de regels rondom method overriding?
De methode moet in de afgeleide klasse opnieuw gedefinieerd worden. Correct overschrijven is van belang voor dynamic binding. We zitten niet met overloading, want de signaturen moeten gelijk zijn: gelijke naam en parameterlijst. De toegang moet gelijk of minder beperkend zijn. Private methodes kunnen niet overschreven worden.

Wat zijn shadowing variables?
Attributen in de afgeleide klasse hebben dezelfde naam als die in de bovenliggende klasse, hierdoor worden ze soort van overschaduwd. Indien we ze toch willen verkrijgen (en indien ze protected zijn): super.*variabelenaam*

Wat is dynamic binding?
Op het moment dat we de code runnen zal de compiler zelf bepalen van welke klasse hij de methode oproept, afhankelijk van welke klasse het object een instantie is.

Leg het principe van Reflection kort uit.
Bij Reflection gaan we at runtime klassen opvragen en er een object van maken met de klasse Class.
Voorbeeld: Basis obj = (Basis) Class.forName(type).newInstance();
Of (Basis) Class.forName(type).getConstructor(Class types).newInstance(argumenten);

Leg het principe van varargs uit.
We kunnen methodes zodanig declareren dat bij hun aanroep niet steeds hetzelfde aantal argumenten moet vermeld worden.
Voorbeeld: public int sum(int… numbers) { (…) }
Dit argument mag dus 0 of meerdere keren voorkomen, of een array zijn.
Varargs mogen slechts op één plaats voorkomen, en moeten altijd als LAATSTE staan.

Aan welke voorwaarden moet equals voldoen?
Niet-null ref.: x.equals(null) = false
Reflexitiviteit: x.equals(x) = true
Symmetrie: x.equals(y) = true ⇔ y.equals(x) = true
Transitiviteit: x.equals(y) = true en y.equals(z) = true x.equals(z) = true
Consistentie: als de objecten waar x en y naar refereren niet wijzigen, dan moet het resultaat van x.equals(y) steeds hetzelfde zijn.

Geef een goede structuur voor een equals()-methode.
Controleer of de objecten hetzelfde zijn.
Controleer of het object null is.
Controleer of het object van dezelfde klasse is?
Ja, cast het naar deze klasse en vergelijk de instantievariabelen.

Wat is iets wat we altijd moeten doen als we equals() overschrijven?
De hashcode overschijven.

Exception handling

Wat is het nut van een finally-blok?
Het stuk code dat in het finally-blok staat zal ongeacht de opgevangen errors toch uitgevoerd worden. Moest catch iets throwen of returnen, dan zal finally toch nog uitgevoerd worden.

Hoe zit het met excepties bij method overriding?
Indien de bovenliggende methode een exceptie opwerpt mag de overschrijvende methode de exceptie weglaten of een exceptie van een covariant type opwerpen.

Static, final en abstract interfaces

Maak objecten en geef ze een nummer, hoe zou je dit aanpakken?
Laat de klasse door middel van een static het aantal aangemaakte objecten bijhouden en ken telkens de waarde van deze static int toe aan een aangemaakt object.

Waarom is de methode main static gedeclareerd?
De main-methode is het startpunt van een Java-applicatie. Er zijn op dat moment nog geen objecten gecreëerd van de klasse waarin main gedefinieerd is. Door de static kan de main-methode geactiveerd worden zonder een object aan te hoeven maken.

Kan er van een final klasse overgeërfd worden?
Neen, alle methodes zijn final, dus waarom zouden we hiervan overerfen?

Hoe werkt normale compilatie?
De code van methode zit slechts 1x in geheugen, bij aanroep springen we naar het stuk code en doen we aan stackmanipulatie. Dit neemt veel tijd in beslag door de administratie.

Hoe werkt in-line compilatie?
Code wordt expliciet volledig ingevoegd op de plaatsen van aanroep: opdrachten die de methode gebruikt, worden ter plekke gegeneerd. De aanroep is hier veel sneller.

Wat zouden redenen zijn om voor final te kiezen?
Final zegt aan de compiler dat dingen afgewerkt zijn, dus geen dynamic binding nodig. De compiler kent de code volledig. Het compilen zal veel sneller gaan.

Leg het diamond problem uit.
A bevat methode m(), B en C erven m() van A en overschrijven deze. Klasse D implementeert zowel klasse B als C (interface), maar overschrijft m() niet. Als we deze code zouden uitvoeren weet de compiler niet welke m() er moet uitgevoerd worden.
De oplossing hiervoor is dat de implementerende klasse D de methode m() MOET overschrijven.

Waarom mogen methodes uit de klasse Object niet geïmplementeerd worden in interfaces?
Methodes uit klassen hebben voorrang op methodes uit interfaces.

Geef de syntax van een lambda expression.
(parameterlijst) -> { (opdrachten) }

Generische types

Wat is de aanduiding van een generisch type in de API?
<E>

Wat zijn de voor- en nadelen hiervan?
Voordelen: er is controle bij het compileren en we moeten geen type casting meer doen.
Nadelen: we moeten telkens het type meegeven als we iets willen maken. Vb: List<Punt>.

Wat is de voorwaarde om objecten van generische types aan elkaar gelijk te stellen?
Het type tussen de <> moet gelijk zijn.

Hoe zit het met hiërarchie bij generische types, geef ook een voorbeeld.
De hiërarchie is gebaseerd op basistype, niet op type van paramter.
Een ArrayList<Integer> is een Collection<Integer> en een List<Integer>, maar geen ArrayList<Object>.
Stel dat de volgende code gecompileerd kan worden:
List<Integer> li = new ArrayList<>(); li.add(123); List<Object> lo = li;
lo.add("hallo wereld"); Integer i = li.get(1);
Dit zou tot een ClassCastException leiden, want Object en Integer zijn verschillende types.

Hoe noteren we een wildcard en wat is er speciaal aan?
<?>, dit is welk enkel read-only.

Hoe kunnen we wildcards gebruiken voor objecten van een bepaalde klasse en afgeleide klassen?
<? extends Klasse> bij een bovengrens en <? super Klassenaam> bij een ondergrens.

Hoe werkt een generic template?
We voegen na de toegankelijkheid een <T> toe en overal waar we dat type gaan gebruiken noteren we T. T is een referentietype, geen primitief type.

Geef een voorbeeld van een bounded generisch type.

```
public <T extends Comparable> T max(T[] tab) {
	T max = tab[0];
	for (T element: tab) {
		if (element.compareTo(max) > 0) {
			max = element;
		}
	}
	return max;
}
```

Gebruik van referentie-objecten

Een copy-constructor moet altijd een diepe kopie (deep copy) maken van objectattributen, op één uitzondering na, dewelke.
Indien het objectattribuut een immutable object is (vb. String).

Wat is het best om te klonen, .clone() of de interface Cloneable?
Indien het enkel primitieve types zijn, dan kunnen we .clone() gebruiken. Zitten we met referentieobjecten, dan moeten we Cloneable implementeren en onze eigen clone functie schrijven.

Klassen: Organisatie en toegang

Wat is een package?
Een bundeling van klassen met volgende kenmerken: de klassen horen bij elkaar, staan in dezelfde subdirectory en de naam van deze directory is de naam van de package.

Welke toegangsmodificatoren zijn het best te gebruiken?
Bij attributen: private. Bij methoden: meestal public, behalve hulpmethoden: private. Bij klassen: public indien we ze buiten de package willen gebruiken, geen toegangsmodificator indien we ze enkel binnen de package willen gebruiken.

Enumerations & I18N

Wat is enumerations?
Een verzameling constanten die verschillende waarden voorstellen.

Wat zijn de stappen voor i18n?
Maak de properties-bestanden. Definieer de Locale. Creëer een ResourceBundle. Haal de tekst uit de ResourceBundle.