Concepten OGP

Basisconcepten Java

Commentaar: // (enkele regel) of /* */ (meerdere regels)

Compileren via commandolijnvenster: cd *opslaglocatie_bestand* -> javac *bestand.java* -> java *bestand*

Primitieve datatypes: char, boolean, int, double …

Printen: System.out.print(inhoud); of indien newline toevoegen: System.out.println(inhoud);

long -> bereik int ontoereikend, short/byte -> op geheugen besparen

```
char c = '9';
// we willen 9 printen als int, omzetting?
int i = c - '0';
// als we de unicode waardes van elkaar aftrekken, zal het verschil 9
bedragen
```

Opletten met delingen: geheel / geheel -> geheel, geheel / reëel of reëel / geheel -> reëel

Lazy evaluation

```
uitdrukking1 && uitdrukking2
uitdrukking1 = false -> resultaat = false
uitdrukking1 = true -> resultaat = resultaat van uitdrukking2

uitdrukking1 || uitdrukking2
uitdrukking1 = true -> resultaat = true
uitdrukking1 = false -> resultaat = resultaat van uitdrukking2
```

Voorwaardelijke uitdrukking
voorwaarde ? uitdrukking1 : uitdrukking2

Basisbegrippen OGP

Object = al dan niet bestaande entiteit
- identiteit: unieke code
- toestand: geheugeninhoud object
- gedrag: diensten die het object levert

Klasse = object is een instantie van een klasse
- type van een object
- fabriek: constructor om object te maken
- aanbieder van diensten: klasse kan zelf ook diensten aanbieden

Bestaande klassen gebruiken

Klasse Random

import java.util.Random;
Constructor: Random()
Methodes: API

Klasse String

Geen noodzaak om constructor op te roepen.
Methodes: API
Immutable object

Klasse StringBuilder

Zeer handig indien we meermaals stukken moeten toevoegen achteraf -> geheugenbesparend
Methodes: API

Wrapper Classes

Elk primitief type heeft een wrapper class: boolean -> Boolean, char -> Character, byte -> Byte, short -> Short, int -> Integer, long -> Long, float-> Float, double -> Double
Elke wrapper class (behalve Character) heeft een statische methode om een string naar zijn type om te zetten. (kan NumberFormatException opwerpen)

Boxing en unboxing

Boxing = automatische conversie van primitieve waarde naar wrapper-object.
Unboxing = automatische conversie van wrapper-object naar primitieve waarde.

Klasse Scanner

Import java.util.Scanner;
Constructor: new Scanner(System.in/new File(bestandsnaam)/String)
Methodes: zie API
! altijd sluiten: naam.close() indien geen try methode gebruikt wordt !
next() leest tot whitespace, nextLine() leest tot newline

Zelf klassen maken

Klasse bevat: data (attributen) en methoden (die het gedrag bepalen)
Data hiding = implementatiedetails verbergen en data beschermen (consistentie bewaken)
Waarde opvragen van private attributen: getter, toekennen: setter.
Constructor overloading: aantal en type argumenten bepaalt welke constructor opgeroepen wordt.
Method overloading: twee of meer methodes uit dezelfde klasse hebben dezelfde naam, maar verschillende signatuur (en andere definitie)
Signatuur methode: [modificatoren], type teruggeefwaarde, naam methode, [lijst parameters], [excepties die opgeworpen kunnen worden]

Collections

Array

Methodes: API
type[] naam = {elementen} of new type[grootte];
Overlopen: kan met for en for-each loop

```
for (type var : collectie) {
      opdrachten;
}
```

args: command line arguments
args.lengt == 0 -> geen extra ingegeven dingen (in tegenstelling tot C/C++ waar het 1 zou zijn)

ArrayList

Dynamische array, geen vaste grootte.

Hash- en TreeSet

```
[Hash]Set<E> naam = new HashSet<>();
[Tree]Set<E> naam = new TreeSet<>();
Set<String> set = new HashSet<>();
Set<Character> ts = new TreeSet<>();
```

Hash- en TreeMap

```
[Hash/Tree]Map<K,V> naam = new Hash/TreeMap<>();
Map<String,Integer> lft = new HashMap<>();
```

Exception Handling

RuntimeException mag zonder try-catch.

Basissyntax: try-catch-finally

```
try {
      opdrachten die excepties kunnen opwerpen;
} catch (exceptieklasse e) {
      opdrachten met de exceptie e;
} catch (andere_exceptieklasse e) {
      opdrachten met de exceptie e;
}
finally {
      optioneel: deze opdrachten voeren we sws uit;
}
```

Finally wordt ten alle tijde uitgevoerd, als finally en catch iets returnen, zal de return van finally de return van catch overschrijven.

```
class NulDelingException extends Exception {}
// deze exceptie moet opgevangen worden
class NulDelingException extends RuntimeException {}
// deze exceptie moet niet opgevangen worden
```

Excepties bij method overriding

```
class A {
	public void meth() throws Exception { ... }
}
// één van de twee onderstaande methodes
class B extends A {
	public void meth() throws [Runtime]Exception { ... }
	public void meth() { ... // exceptie opvangen }
}
```

Alle mogelijke excepties opvangen

```
catch (Exception e) { ... }
// altijd laatste in de catch-lijst!
```

Static, final en abstract interfaces

Static zorgt er voor dat iets per klasse bestaat, niet per object.
Opgelet: static methode heeft alleen toegang tot static methodes / attributen.

```
public class Voorbeeld {
	static int aantal;
	public Voorbeeld() {
		aantal++;
	}
	static int aantalGemaakteObjecten() { return aantal; }
}
```

Final zorgt er voor dat iets niet te wijzigen is. Men kan van een final klasse niet overerven en een final methode kan men niet overschrijven, ze wordt in-line gecompileerd -> besparing tijd.

Abstract geeft aan dat iets onvolledig is en nog gespecialiseerd moet worden.
Het verplicht tot het overriden van methodes

```
public abstract class Klasse {
	public abstract void print();
}
```

Gebruik van referentie-objecten

Copy-constructor is een constructor met één parameter die van hetzelfde type is als de klasse waarvan hij deel uitmaakt.

```
public Datum(Datum d) {
	dag = d.dag; maand = d.maand; jaar = d.jaar;
}
```

Privacy leak: indien we als teruggeefwaarde een referentie geven, dan bestaat het gevaar op een privacy leak. De programmeur zo via een omweg het private attribuut kunnen aanpassen.

Enumerations en i18n

VROEGER

```
public class Season {
	public static final int SPRING = 0;
	public static final int SUMMER = 1;
	public static final int FALL = 2;
	public static final int WINTER = 3;
}
```

NU

```
public enum Season { SPRING, SUMMER, FALL, WINTER}
```

```
//properties bestand
begroeting = Hoi.
afscheid = Vaarwel.
//resourcebundle
Locale currentLocale = new Locale("nl", "BE");
ResourceBundle rb = RecourseBundle.getBundle("msg",currentLocale);
System.out.println(rb.getString("begroeting"));
```