HoGent
BEDRIJF
EN
ORGANISATIE

Valentin Vaerwyckweg 1
BE-9000 Gent
T +32 9 243 22 00
fbo@hogent.be
www.hogent.be/fbo

Opleiding, afstudeerrichting en jaar: TIN 2 Naam van het opleidingsonderdeel: OOPIII (Eventueel) dOLOD / deelexamen: Campus: Aalst / Schoonmeersen Lector(en):	Examendatum: Aanvangsuur examen: 13u30
Naam en voornaam student:	
Geboortedatum student:	Studentennummer:
Lector bij wie de student de onderwijsactiviteit volgde:	Lesgroep waarin de student de onderwijsactiviteit volgde:
Behaald resultaat: _____ op _____	

[ ] Tijdens het examen mogen **GEEN** hulpmiddelen gebruikt worden:

[x] Tijdens het examen mogen onderstaande hulpmiddelen gebruikt worden:

- Handboek “Java, How to Program” van Deitel en Deitel
- API-documentatie:
  - C:/Program Files/Java/docs/api/index.html of docs.oracle.com/javase/8/docs/api
  - docs.oracle.com/javaee/6/api
  - docs.oracle.com/javafx/2/api
- Ter beschikking gestelde slides en NetBeans-project

Algemene richtlijnen:

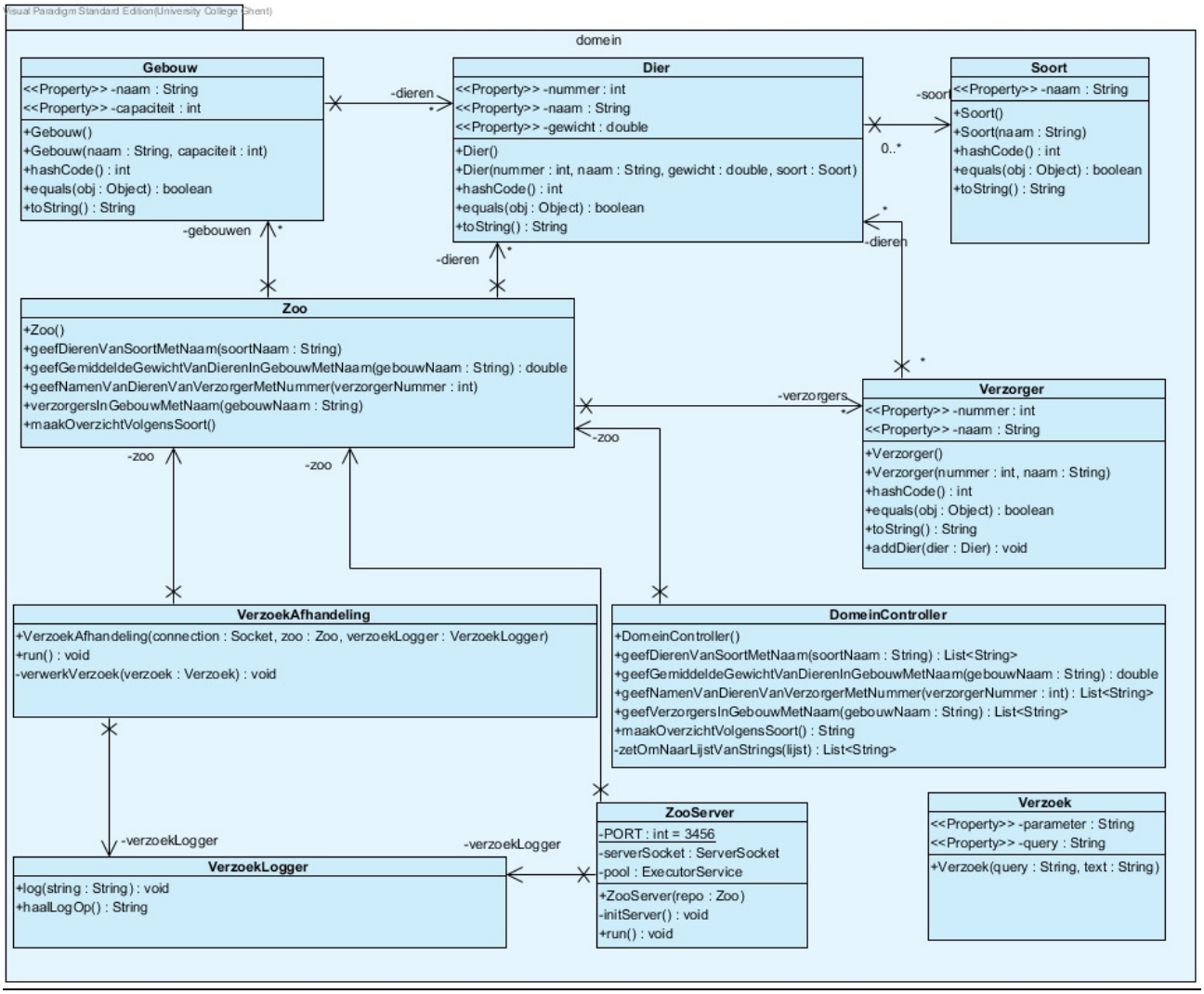
- Vul het bovenstaande kader aan.
- Vul op elke bladzijde je naam en voornaam in.
- Controleer of deze examenbundel alle pagina’s bevat, zo niet verwittig de docent of de toezichter zodat je een nieuw exemplaar krijgt.
- Schrijf duidelijk, onleesbaar wil zeggen geen punten.
- Elke vorm van mondelinge, schriftelijke of elektronische (bijv. mobiele telefoon, smartphones, enz.) communicatie tijdens examens, tenzij deze uitdrukkelijk is toegelaten, worden beschouwd als “onregelmatigheid” en vallen onder de toepassing van OER art. 60 Examentucht.

HoGent

Vul hieronder je naam en voornaam in:

Gegeven:

Alle vragen worden gesteld in functie van onderstaand klassendiagram!



Vul hieronder je naam en voornaam in:

Vraag1: Collections (35 punten).

Vervolledig de klassen **DomeinController** en **Zoo** met de onderstaande methodes zodat de uitvoer van Applicatie1 (in ui) de volgende uitvoer geeft:

```
Dieren van soort krokodil:
[Happy, Kroky]


Gemiddelde gewicht dieren in gebouw Reptielen:
99.5

Namen van dieren van verzorger 1:
[Kroky, Happy]

Dieren per soort:
Krokodil => [Kroky, Happy]
Vogel => [Beo, Koko]

BUILD SUCCESSFUL (total time: 1 second)
```

```
public class DomeinController {
  /**Deze methode maakOverzichtVolgensSoort geeft de informatie uit de map aangemaakt via
maakOverzichtVolgensSoort (p. 5) terug op volgende manier :
                Krokodil => [Kroky, Happy]
                Vogel => [Beo, Koko]
 */
   public String maakOverzichtVolgensSoort() {
```

Vul hieronder je naam en voornaam in:

public class Zoo {
/** Geeft alle dieren terug die behoren tot de diersoort met de opgegeven naam. De lijst van dieren moet gesorteerd zijn op gewicht (laag naar hoog). */

public List<Dier> geefDierenVanSoortMetNaam(String soortNaam) {

/** Geeft het gemiddelde gewicht terug van alle dieren die verblijven in het gebouw met de opgegeven naam. Geeft 0 terug indien er geen gebouw is met deze naam.*/

public double geefGemiddeldeGewichtVanDierenInGebouwMetNaam(String gebouwNaam) {

Vul hieronder je naam en voornaam in:

/** Geeft de namen van de dieren terug die verzorgd worden door de verzorger met het opgegeven nummer. Geeft een lege lijst terug indien er geen verzorger is met dit nummer.*/

```
public List<String> geefNamenVanDierenVanVerzorgerMetNummer(int verzorgerNummer){
```

/**De methode maakOverzichtVolgensSoort geeft een overzicht (Map) terug van alle dieren per Soort.*/

```
}//einde Zoo
```

Vul hieronder je naam en voornaam in:

Vraag 2: JPA (15 punten)
Opgave 2.1: Mapping

- Vervolledig onderstaande klassen met de juiste en nodige JPA-annotaties zodat de objecten ervan kunnen opgeslagen worden in een database herexamen.
- Vervolledig ook de klassen met eventueel nodige extra methodes.
- Zie klassendiagram voor de associaties!
- De namen van de tabellen/velden in de database zijn dezelfde als namen van de klassen/attributen.
- We opteren ervoor om voor elke klasse een aparte tabel te voorzien.
- Overige klassen: alle attributen worden weggeschreven naar de database.

```
public class Dier implements Serializable
{


    private  int nummer;
    private String naam;
    private double gewicht;


    private  Soort soort;


    public Dier(int nummer, String naam, double gewicht, Soort soort)
    {
        this.nummer = nummer;
        this.naam = naam;
        this.gewicht = gewicht;
        this.soort = soort;
    }
   /*overige methodes*/
}//einde Dier


public class Soort implements Serializable
{

    private  String naam;


    public Soort(String naam)
    {
        this.naam = naam;
    }

   /*overige methodes*/
}//einde Soort
```

```
public class Verzorger implements Serializable
{

    private  int nummer;
    private String naam;


    private final List<Dier> dieren = new ArrayList<>();




    public Verzorger(int nummer, String naam)
    {
        this.nummer = nummer;
        this.naam = naam;
    }

  /*overige methodes*/
}//einde Verzorger



public class Gebouw implements Serializable
{


    private  String naam;
    private int capaciteit;



    private final List<Dier> dieren = new ArrayList<>();




    public Gebouw(String naam, int capaciteit)
    {
        this.naam = naam;
        this.capaciteit = capaciteit;
    }

//...
}
```

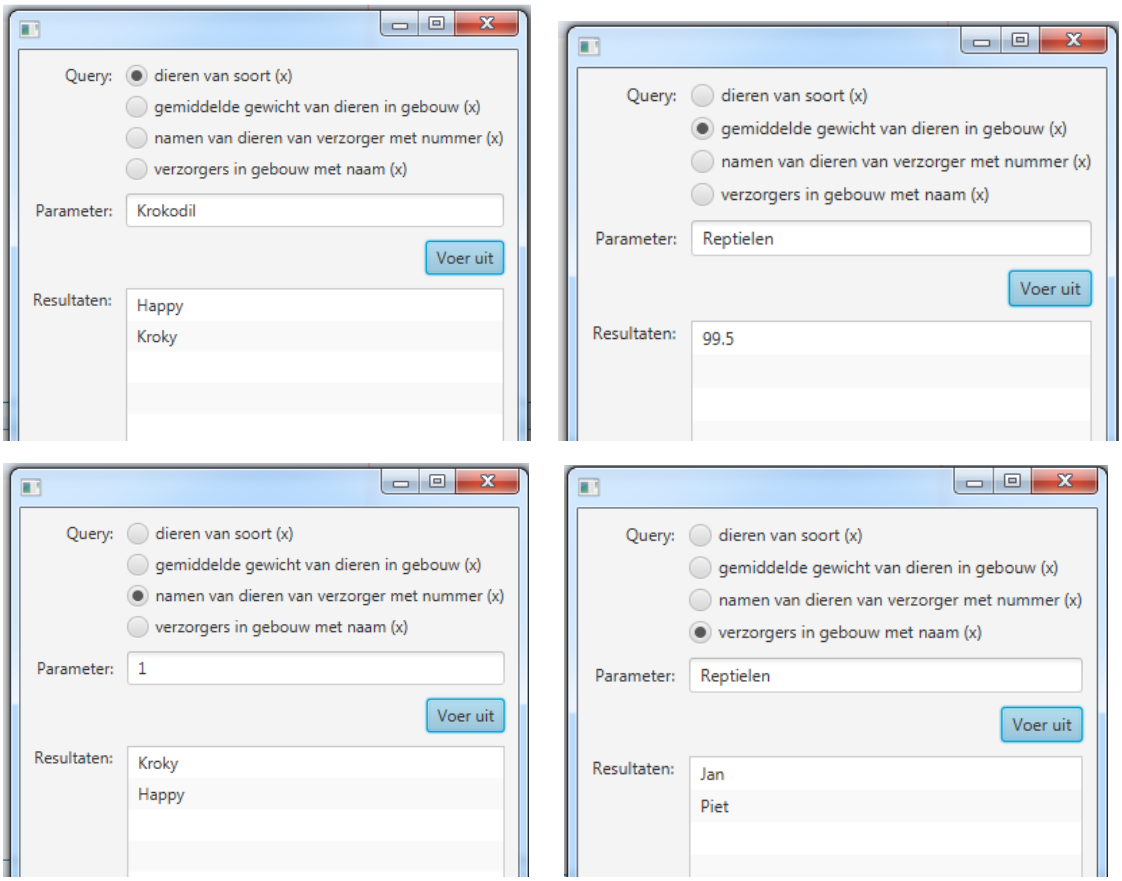
Opgave 2.2: JPQL

Voeg in de klasse Dier een JPQL NamedQuery toe die alle dieren van een opgegeven soort kan opvragen. De resultaten dienen opnieuw gesorteerd te zijn volgens gewicht (laag naar hoog).

Opmerking: het is niet nodig om configuratiebestanden of mapper-klassen aan te maken. Deze worden niet gevraagd!

Vul hieronder je naam en voornaam in:

Vraag 3: ListView – Generics (20 punten) (ook generics deel in vraag4).



In bovenstaande gui wordt een query gekozen en de bijbehorende parameter ingegeven. Na het klikken op de Voer uit-knop wordt het resultaat getoond in een ListView (zie QueryPanelController-klasse).

De uiteindelijke uitwerking van de gui gebeurt verder in de volgende vragen.

Klasse QueryPanelController.java

Schrijf de generieke methode showValue (wordt aangeroepen vanuit de methode doeQuery) om een lijst van bepaalde willekeurige objecten aan de ListView resultatenListView te koppelen.

```
private void showValue {
    spinner.setVisible(false);
    resultatenListView.
}
```

Vul hieronder je naam en voornaam in:

Om de uiteindelijke gui te laten werken hebben we een generieke klasse **Verzoek** van doen. Van deze klasse worden objecten geïnstantieerd die als attribuutwaarden de gekozen query (=query1, query2, query3 of query4) en de ingevulde parameter bevatten.

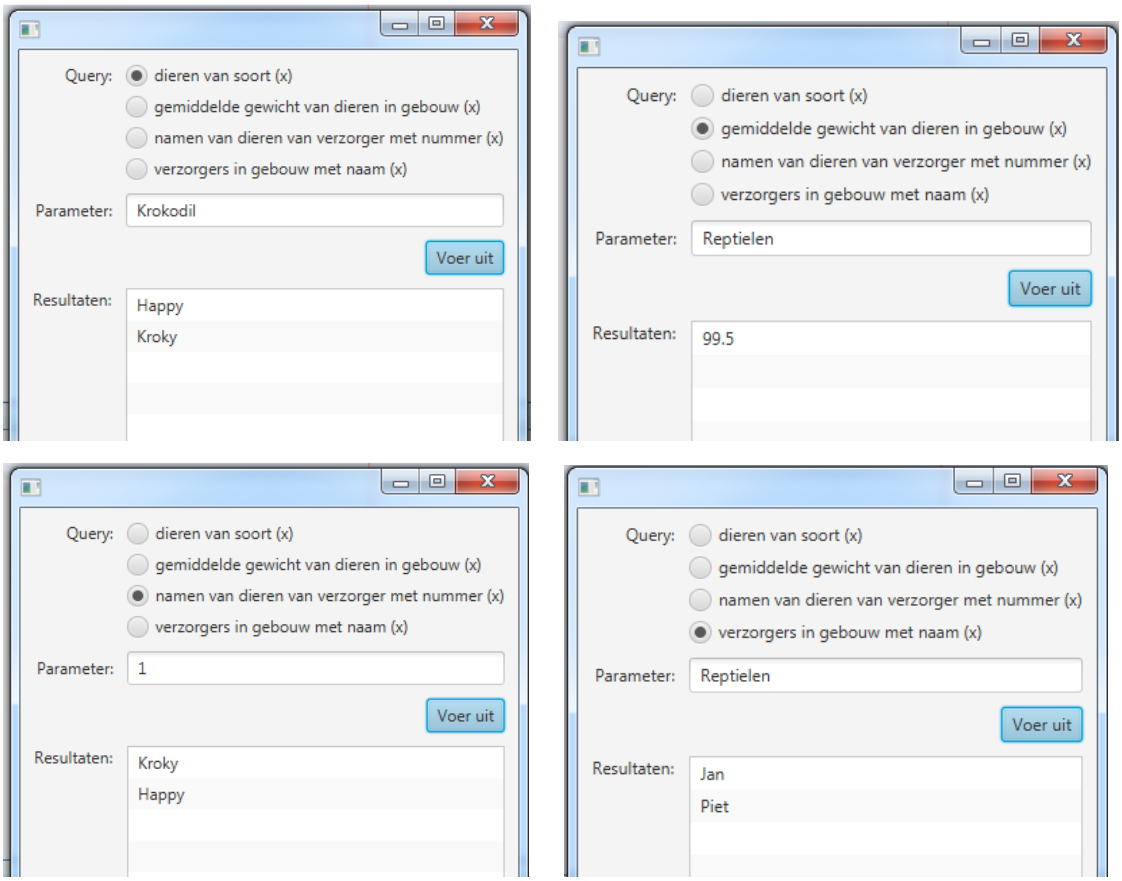
Vul de generieke klasse Verzoek verder aan zodat naast de attributen query en parameter, het resultaat van de query kan worden gestockeerd. Het resultaat is een lijst van willekeurige objecten. In ons voorbeeld kunnen dat objecten van Dier, String, Verzorger of Double zijn.
Let erop dat objecten van **Verzoek** over een objectstream worden verzonden en dus Serializable moeten zijn.

Maak de generieke klasse verzoek, vul verder aan:

```
public class Verzoek

   private final String query;

   private final String parameter;

   //nog attribuut…




   public Verzoek(String query, String parameter) {
      this.query=query;
      this.parameter=parameter;
   }
   public String getParameter() {
      return parameter;
   }
   public String getQuery() {
      return query;
   }

  //nog getter/setter voor attribuut..

}
```

Vul hieronder je naam en voornaam in:

Vraag 4: Netwerken/Multithreading/Generics (30 punten).



Bovenstaande gui is een afzonderlijke clientapplicatie die de getoonde resultaten verkrijgt door een verzoek aan de ZooServer applicatie te versturen via een TCP netwerkverbinding. Voor elke nieuwe keuze van query wordt opnieuw een verbinding met de ZooServer gemaakt en daarna beëindigd. Er kunnen meerdere clients tegelijk actief zijn, die elk hun verzoeken naar de ZooServer versturen. De ZooServer zal zo'n verzoek afhandelen in een afzonderlijke thread (klasse **VerzoekAfhandeling**), zodat meerdere verzoeken niet op elkaar hoeven te wachten tot ze volledig afgehandeld zijn.

De klasse **ZooServer** wacht op de komende client verzoek en delegeert dan het effectieve werk aan een nieuwe instantie van **VerzoekAfhandeling**.

Opstart van een client : **startUp.StartUpGuiClientApp.java**
Opstart van de ZooServer : **ui.ApplicatieZooServer.java**
Deze beide klassen zijn reeds volledig uitgewerkt.

Als de **VerzoekAfhandeling** een verzoek uitvoert zal die bijkomend een logging doen. Dit gebeurt bij een instantie van **VerzoekLogger**, met een aanroep van de methode **log**. Via het stringargument van de **log** methode wordt de hostnaam van de client gevolgd door de querynaam (vb "query1") meegegeven. Als het verzoek afgehandeld is zal de **VerzoekAfhandeling** nogmaals een logaanroep doen met hetzelfde argument van de eerste aanroep maar uitgebreid met " afgehandeld".

De **VerzoekLogger** houdt elke log in een lijst van String bij (tot max 1000 logs bijhouden). De **VerzoekLogger** heeft buiten de **log**methode ook een een methode **haalLogOp**, die telkens de oudste log van de lijst verwijdert en teruggeeft (FIFO organisatie). We schrijven deze methode wel uit, maar

gaan ze voorlopig nog niet verder gebruiken in onze toepassing.
Let wel op dat meerdere verzoeken tegelijkertijd kunnen doorgaan.

De client: Klasse QueryPanelController.java

//de generieke hulp methode **contactServer** die **doeQuery** gebruikt. **doeQuery** is afgewerkt.

```
private void contactServer ( verzoek ){
    verzendVerzoek(verzoek);
    verzoek = ontvangResultaat();
    showValue(verzoek.getResultaat());
}
```

```
//nog uit te werken methoden: de generieke methode
//
// de generieke methode verzendVerzoek stuurt over het netwerk een Verzoek object van een client
// naar de server.
private void verzendVerzoek ( verzoek ) {
```

```
// ontvangResultaat: de client ontvangt via het netwerk het door de server verder aangevulde Verzoek
// object.
private Verzoek ontvangResultaat () {
```

Vul hieronder je naam en voornaam in:

//de initConnection method zal een verbinding tussen client en server aanmaken, en zorgen dat er via
//een stream kan gecommuniceerd worden

```
private void initConnection () {
    final int PORT = 3456;
    final String HOST = "localhost";
```

De klasse **ZooServer**: // enkel nog de run methode uitschrijven!
//De **ZooServer** wacht clientverzoeken af en delegeert de afhandeling ervan aan een instantie van
//**VerzoekAfhandeling**.

```
  public void run() {
```

De klasse **VerzoekAfhandeling**:
//enkel nog **log** aanroepen in de methode **verwerkVerzoek** en de **run** methode uitschrijven!

```
 private void verwerkVerzoek(Verzoek verzoek ) {
        String logBericht =



        verzoekLogger.log( logBericht );
        switch (verzoek.getQuery()) {

//enz….
         verzoekLogger.log(String.format("%s afgehandeld", logBericht) );
}
```

Vul hieronder je naam en voornaam in:

//de server ontvangt een clientverzoek (een instantie van **Verzoek**), verwerkt het verzoek (methode //**verwerkVerzoek**) en stuurt het aangepaste **verzoek** terug naar de client .

```
public void run() {
```

De klasse **VerzoekLogger**:
//Methode log plaats de string berichten in een wachtrij (FIFO organisatie)
//Methode haalLogOp haalt een bericht uit die wachtrij.
//Let op: er kunnen meerdere **log** en **haalLogOp** acties tegelijkertijd plaats vinden

```
public class VerzoekLogger {


  public  void log(String string) {









   }
  public String haalLogOp(){









  }
}
```