Teknisk Dokumentation

Team06 EDAF45

27 februari 2017

1 Utvecklingskommandon

För att bygga projektet och generera .jar-filen:

- Befinn dig i **/pvgvt17-team06-production/dasUberEnduroApplicationen* i terminalen.
- Skriv på kommandoraden "./gradlew build".

För att köra acceptanstesterna:

- Befinn dig i **/pvgvt17-team06-production/* i terminalen.
- Skriv på kommandoraden "./runAcT.sh"

2 Designbeskrivning

Vårt program DasUberEnduroApplicationen består av fyra paket, util, registration, resultMerge och server_client. Hela programet använder sig av designmönstret Model-View-Controller. I Figur 1 visas en översikt av alla paket och alla klasser i src. Varje Racer är ett eget objekt som håller koll på information om sig själv som: namn, startnummer, tävlingsklass och övrig information. Olika sorters tävlingar beskrivs av interfacet RaceType, som implementeras av exempelvis OneLapRace, MultiLapRace och EtappRace. Vi använder oss av Strategy-mönstret som gör att varje Racer-objekt har ett RaceType-objekt i sig som bestämmer vilken typ av tävlingssort det Racer-objektet ska tävla i. RaceType-klasserna ansvarar för att generera sista delen av resultatsträngen för varje Racer.

Exempel, MultiLapRace: Om RaceType är ett MultiLapRace skriver först Racer-objektet ut de två första raderna i strängen som är ”StartNummer; Namn”. Därefter anropas RaceType.genResult() som genererar information från just den tävlingssorten. I detta fallet ser fortsättningen av strängen ut så här: ”#Varv; TotalTid; Varv1; Varv2; Varv3; Start; Varvning1; Varvning2; Mål”.

Utöver att generera strängen håller även RaceType reda på tider som hör till Racer-objektet. Databasen är sedan en samling av alla Racer-objekt, där dem sparas i en HashMap med dess startnummer som nyckel. I Klassen ResultWriter i util-paketet ska resultatet skrivas ut på en textfil. ResultWriter anropar då Database.getResult() som genererar en header beroende på vilken tävlingstyp det är och itererar sedan över databasen med Racers och kallar på alla Racers toString()-metod. Därefter returneras resultatet till ResultWriter som skriver ut det på en fil. IOReader-klassen i util-paketet står för inläsning av textfiler. Klassen läser dels namn filer för att fylla Database med Racers och dels start- och måltidsfiler för att lägga till start- och måltider för Racers i databasen. För att programmet ska fungera behöver filerna vara i textformat och formatterade precist efter den standard som är angivet i acceptanstester från story 1 och frammåt. Detta för att vår IOReader ska kunna läsa korrekt. För att sortera resultatet implementerar Racer-klassen interfacet Comparable, och jämför sig själv med andra Racers. Först efter antalet varv och sedan efter totaltid. Med denna lösning kan vi använda den i både varvlopp, maratonlopp och etapplopp. Jämförelsen och sorteringen görs först när utskriften sker.

2.1 Config JSON

För att stödja mer konfigurering av racet så använder vi oss av en JSON fil för att ge parametrar till programmet innan körning. Parametrar som stöds är följande:

race type - Tävlingstypen, kan vara ”maratonlopp”, ”varlopp” eller ”etapplopp”.

group start - Ange endast tid enligt ”hh:mm:ss” om man vill att alla ska ha samma starttid.

name file - Lokala sökvägen för namnfilen.

stipulated time - Stipulerade tiden för varvlopp, ges i "hh:mm:ss".

number of etapps - Ges som ett heltal för att tala om antalet etapper i etapplopp.

start files - Lokala sökvägen för filen med starttider och startnummer. Flera filer kan anges med komma mellan. För etapplopp ges startfiler som en vektor med objekt innehållandes parametrarna ”etapp” med numret på etappen och ”file” med lokala sökvägen för startfilen för den etappen.

finish files - Lokala sökvägen för filen med sluttider och startnummer. Flera filer kan anges med komma mellan. För etapplopp ges målfiler som en vektor med objekt innehållandes parametrarna ”etapp” med numret på etappen och ”file” med lokala sökvägen för startfilen för den etappen.

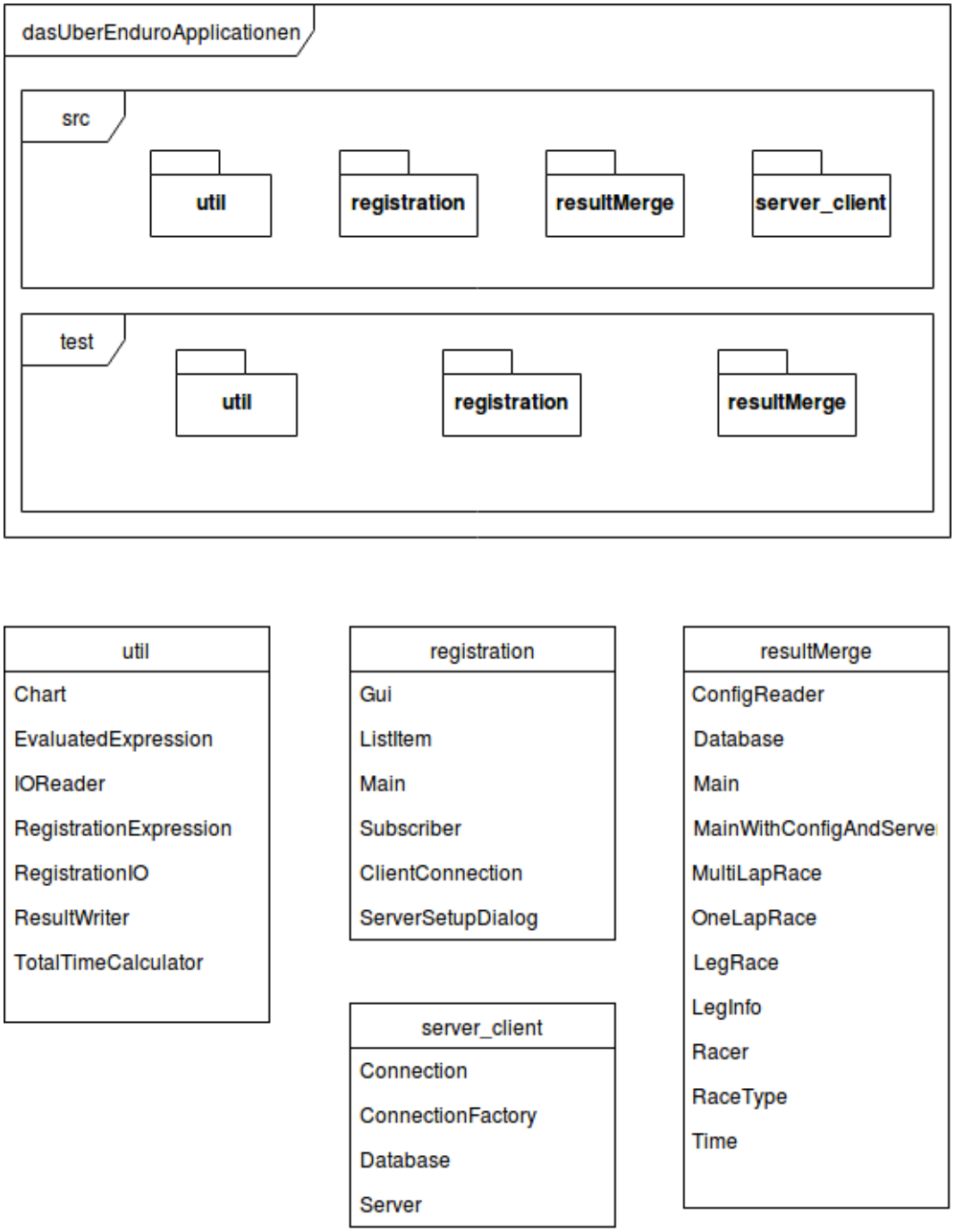
etapper - Används för att ge info om varje etapp, t.ex minimum tid. Ges som en vektor med objekt innehållandes parametern ”minimum time”.

3 In och utläsning

Inläsning behövs göras av en konfigurations fil, en namnfil, eventuella starttidsfiler och sluttidsfiler. I konfigurationsfilen anges sökvägen till namnfilen, starttidsfiler och sluttidsfiler. Ett exempel på hur konfigfilen formatteras och hur man skriver in parametrar finns i repot. Starttidsfilen och sluttidsfilen genereras och formateras korrekt av vårt registreringsprogram. Namnfilen ska vara formaterad efter det format som det är i acceptanstesterna.

Output från result merge är en sorterad resultatlista, en osorterad resultatlista samt en sorterad resultatlista i html format.

4 Klasser och filstruktur



Figur 1: Paketstruktur och listor av klasser i programmet dasUberEnduroApplicationen.

4.1 registration

Gui Huvudfönstret för registrering av Racers. Uppdelad i tre paneler: översta för att skriva in startnummer och registrera, vänstra för att se alla nuvarande registreringar, och högra för att se alla nuvarande registreringar

som ej gått igenom på grund av oläsbar nummerplåt eller inkorrekt input. Det går att ta bort och editera registreringar i höger panel.

ListItem Ett ListItem-objekt består av Racer med sin registerade tid, samt Edit och Remove button om den befinner sig i höger panel.

Main main för registreringsprogrammet.

Subscriber Enbart till för att notifiera Gui:t då en racer har blivit editerad eller borttagen. Implementeras av Gui.

ClientConnection En connection mellan clienten och servern. Används för att skicka data till Servern.

ServerSetupDialog En dialogruta för att skriva in IP-adress och portnummer för serverprogrammet. Skapar en ClientConnection och för söker ansluta till servern.

4.2 resultMerge

ConfigReader Läser config-filen som innehåller alla väsentliga filer. Beskrivs mer i vår manual. Här skapas en Database beroende på vilken Race-type som väljs.

Database Central klass för resultathantering. Här hanteras exempelvis racers, tider, racetyp, resultat, m.m.

RaceType Interface med metoder gemensamma för alla olika RaceTypes.

LegRace Funktionalitet för etapplopp. Implementerar RaceType.

MultiLapRace Funktionalitet för varvlopp, med eller utan masstart. Implemeterar RaceType.

OneLapRace Funktionalitet för ett varvs lopp, med eller utan masstart. Implementerar RaceType.

LegInfo Håller koll extra info om alla etapper, t.ex minimum tid för etapper

Main main för resultMerge utan stöd för konfiguration.

MainWithConfigAndServer main för resultMerge med stöd för konfiguration och server. Integration med databas är inte klart än.

Racer Skapar en racer med namn, klass, startnummer och optional data. Kan jämföra sig med andra racers för att kunna sorteras samt skriva ut sig själv som en sträng.

Time Representera tider på ett generellt sätt för hela programmet.

4.3 server_client

Connection Den delen av servern som tar emot data från klienten och lägger till i databasen.

ConnectionFactory Innehåller en ServerSocket som skapar en ny Socket varje gång en ny klient kopplar upp sig till servern. Det är denna Socket som sedan kommunicerar med klienten på en egen Thread.

Database Enkel dummy-klass som servern lagrar data i.

Server main för server-programmet.

4.4 util

Chart Stöds ej för tillfället. Är tänkt att göra regex enklare och smidigare att hantera vid hantering av strängar.

IOReader Klass som sköter inläsning av filer

RegistrationIO Läser från och skriver till en fil för registration Gui-klassen.

ResultWriter Generera tre olika resultatfiler; en sorterad .txt, en osorterad .txt samt en sorterad .html fil.

TotalTimeCalculator Kan användas för att utföra beräkningar hos tidsträngar.

EvaluatedExpression Wrapper för två länkade listor, en som ger korrekt angivna startnummer samt felaktigt angivna startnummer.

RegistrationExpression Parsar en sträng från registration's Gui och ger tillbaka en EvaluatedExpression.

5 Program och filer

5.1 Program

Vår lösning består av tre huvudprogram. Ett som är registreringsprogrammet. Där kan man registrera tider till de olika startnummerna. Samma program används för start och målgång. Det är upp till användaren att döpa outputfilerna så man vet vad som är vad.

Det andra programmet är resultatprogrammet. Det behöves en konfigurationsfil som input för att kunna exekveras. I konfigfilen anges sökvägen till en namnfilen och eventuella start- och sluttidsfiler som programmet behöver för att kunna generera resultatfiler. I konfigurationsfilen anges även andra parametrar för tävlingen. Som till exempel tävlingstyp, stipulerad tid för varvlopp eller antalet etapper för etapplopp.

Det tredje programmet är serverprogrammet som är under utveckling och som ännu inte är helt integrerat med de två andra programmen.

5.2 Filer

Följande filer existerar i workspacet eller behövs för att kunna köra programmet.

txt Det behövs några input txt filer för att generera outputfilerna. De som behövs av resultatprogrammet är tre olika slags filer. Startidsfiler (behövs ej vid masstart), sluttidsfiler och en namnfil. De två första genereras av registreringsprogrammet och namnfilen är given.

De txt filer som genereras är en osorterad resultatlista och en sorterad resultatlista.

HTML Av resultatprogrammet genereras även en HTML fil som reflekterar innehållet i den sorterade resultatlistan

JSON För konfigurering av resultatprogrammet används en JSON fil som inehåller olika parametrar. Denna filen behövs för att kunna köra programmet.

ShellScript Den filen som heter ”runAcT.sh” är ett av våra shellsript. Den används för att köra alla våra acceptanstester och visa vilka som går igenom/inte går igenom.

Vårt andra shellscript är ”gradlew” som genererar körbara JAR filer av vårt projekt. Det finns två versioner av denna fil. Filen med ändelsen .bat används på windows och den andra på Unix.

Gradle Vi har valt att använda verktyget Gradle för att bygga vårt projekt. Parametrar för hur projektet ska byggas ligger i filen build.gradle och settings.gradle

5.3 Relation mellan filer och program

Här följer ett exempel på ett maratonlopp och hur de olika filerna används i de olika programmen.
Under ett maratonlopp körs en instans av registreringsprogrammet där man genererar startider för åkarna. Denna fil döps förslagsvis till starttider.txt. En annan instans av registreringsprogrammet körs vid målgång där alla måltider registreras. Denna fil döps förslagsvis till maltider.txt. JSON filen behöver konfigureras på ett sådant sätt att resultatprogrammet ska kunna hitta startfilen, målfilen samt att man ska specifisera race typetill maratonlopp. Därefter kan resultatprogrammet köras som då genererar tre filer:

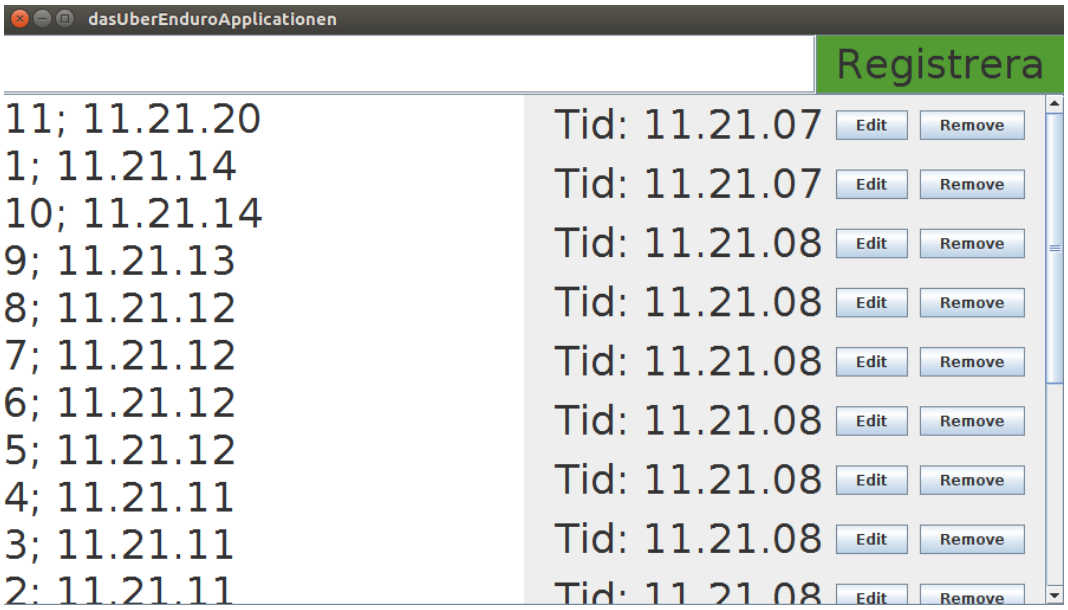
- output.txt - sorterad resultatfil
- output_unsorted.txt - osorterad resultatfil
- output.html - sorterad resultatfil på .html format

5.4 Distribution av programvaran

Det finns två gradle filer för att bygga programmet på antingen en UNIX-dator eller en Windows-dator, men rätt fil väljs automatiskt. Allt en användare behöver göra är att öppna ett terminalfönster, positionera sig i dasUberEnduro-Applicationen mappen samt skriva: ./gradlew build
Efter detta genereras ett antal mappar samt två .jar filer. De två .jar filerna är de två olika programmen. Registreringsprogrammet hittas i mappen /registration/build/libs och resultatprogrammet hittas i /resultMerge/build/libs.

6 Gui

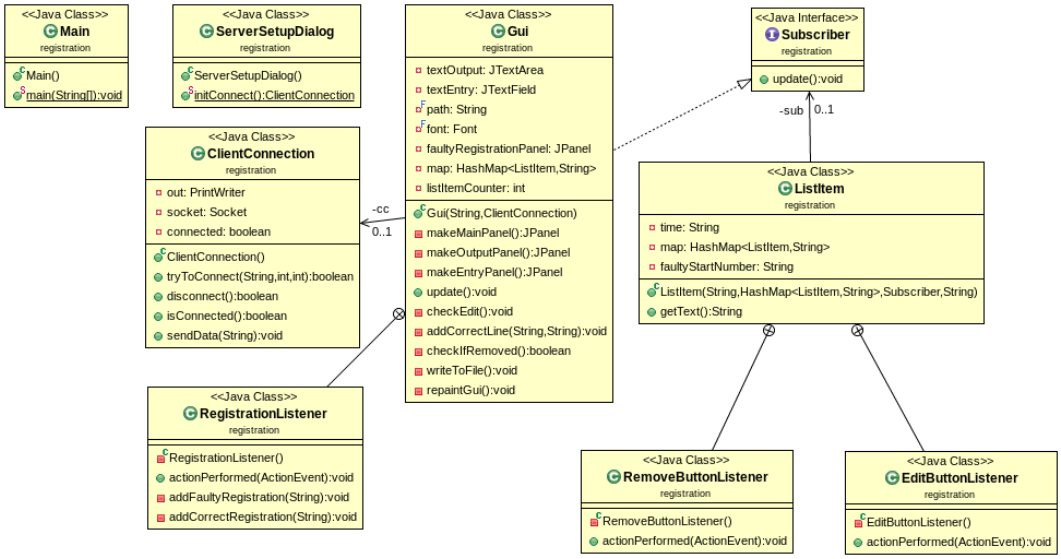
Figur 2 visar hur användargränssnittet ser ut för vårt program. I fältet högst upp skriver man in startnummer för varje Racer. När man klickar på Registrera hamnar den racern i kolumnen till vänster och tilldelas en starttid. Om man inte kan se startnumret klickar man bara på Registrera utan att ange startnummer, då hamnar starttiden i den högra kolumnen tillsammans med två knappar: Edit och Remove. Edit-knappen används om man vill skriva in ett korrekt startnummer, och isåfall hamnar den sedan i den vänstra kolumnen på samma sätt som vid initial registrering. Om man istället klickar på Remove-knappen försvinner starttiden från registeringsprogrammet.



Figur 2: Gui av dasUberEnduroApplicationen.

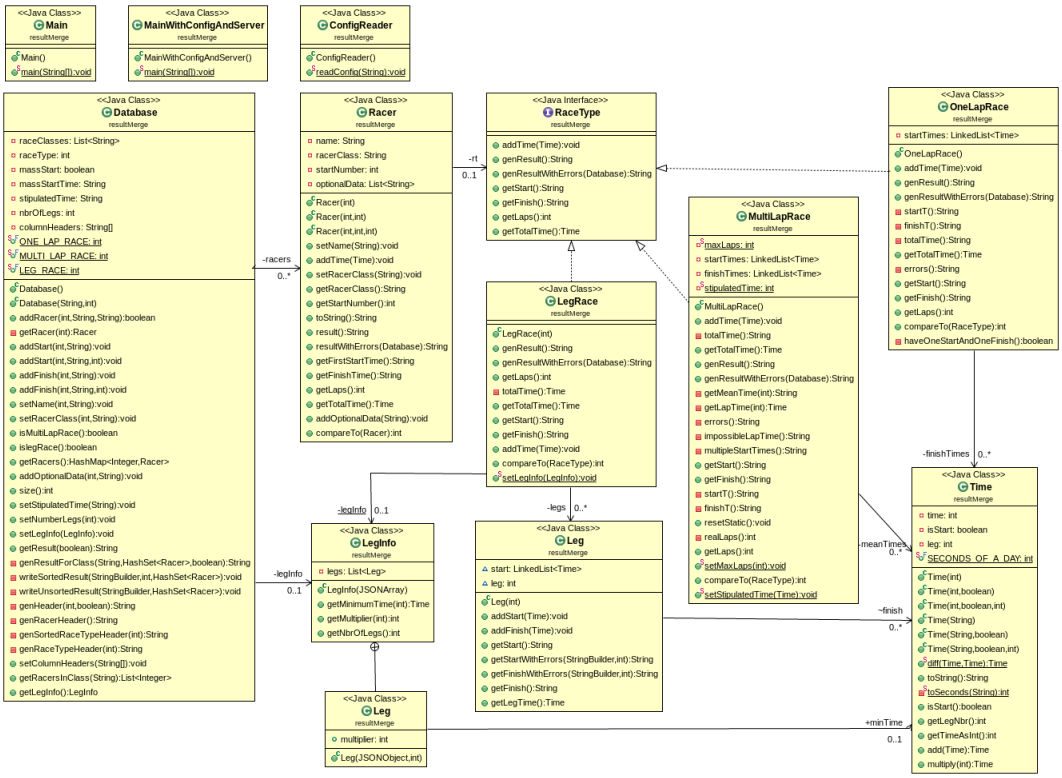
7 UML-diagram

7.1 registration



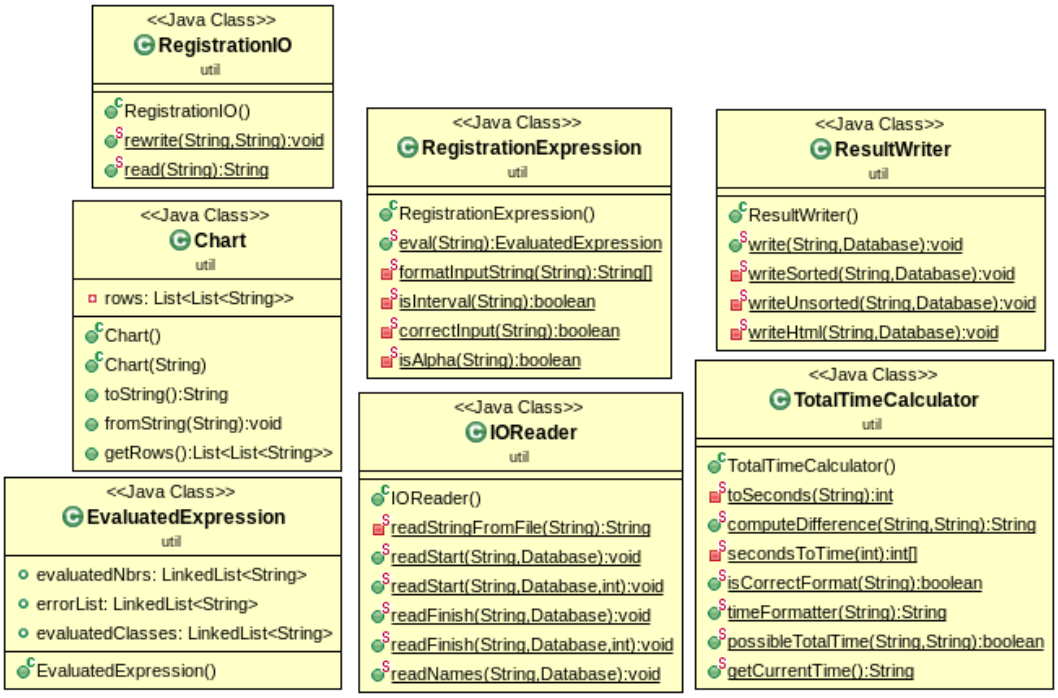
Figur 3: UML av Registration-paketet.

7.2 ResultMerge



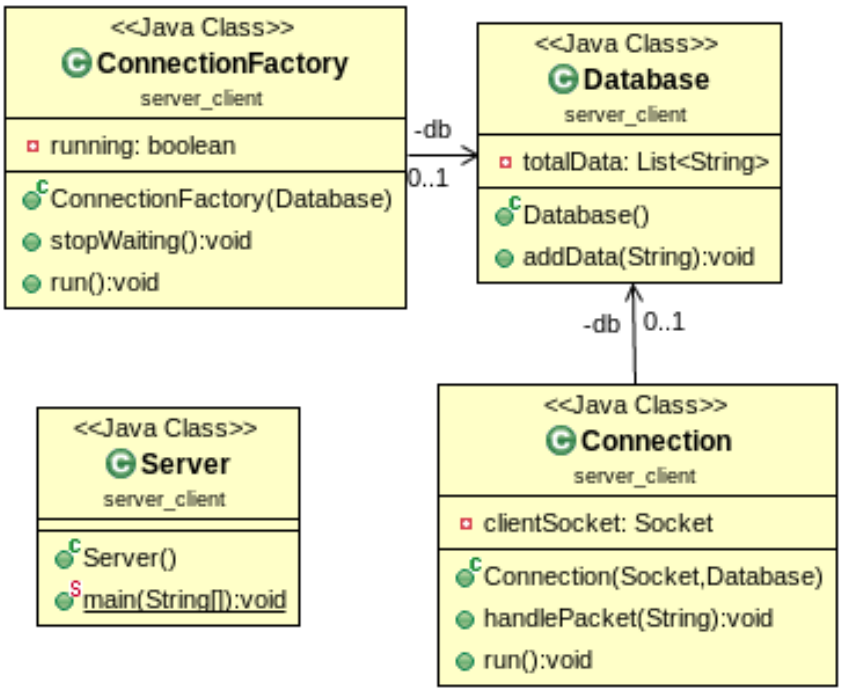
Figur 4: UML av ResultMerge-paketet.

7.3 Util



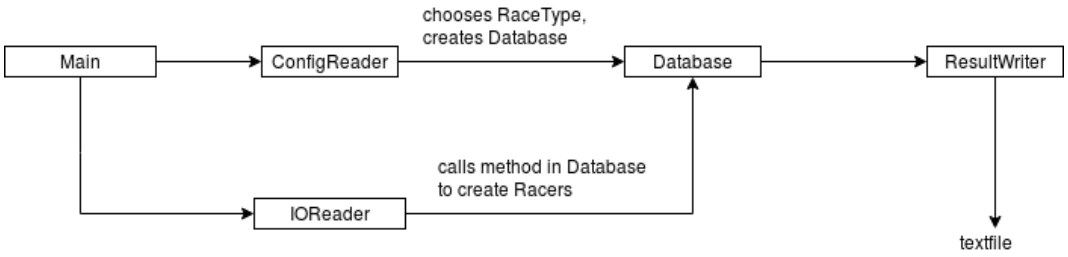
Figur 5: UML av Util-paketet.

7.4 ServerClient



Figur 6: UML av serverClient-paketet.

7.5 Flow



Figur 7: Programflöde från inläsning till utskrift.

8 Server/Client Guide

Starta server:

- Gå in i **/serverTest/* mappen i terminalen.
- Kör scriptet "./mkServerTest".
- Nu är servern igång, för att avsluta skriv i terminalen "exit".

Starta klient:

- Starta registration Gui.
- Skriv i IP-adress-fältet "127.0.0.1öch Port-fältet "10008".

9 Framtida projektplaner (Work in progress)

Vi arbetar på ett antal parallella områden som påverkar UML-diagramen mycket. Nedan är de viktigaste.

- **Util** - Vi håller på att utökar vårt Util-paket för att formatteringen av strängar ska ske på samma ställe. Detta kommer ändra en hel del beroenden och duplicerad kod i stort sätt hela projektet.
- **Client - Server** - En client-server lösning har påbörjats men har ännu inte integrerats helt.