Teknisk Dokumentation

Team06 EDAF45

20 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:

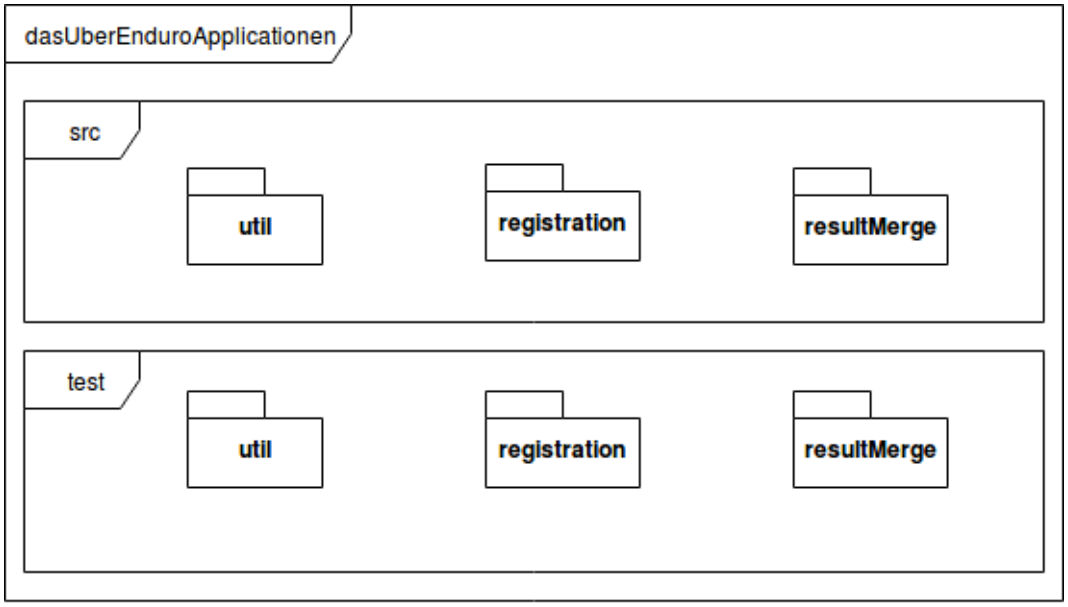
- Befinn dig i **/pvgvt17-team06-production/* i terminalen.
- Skriv på kommandoraden "./runAcT.sh"

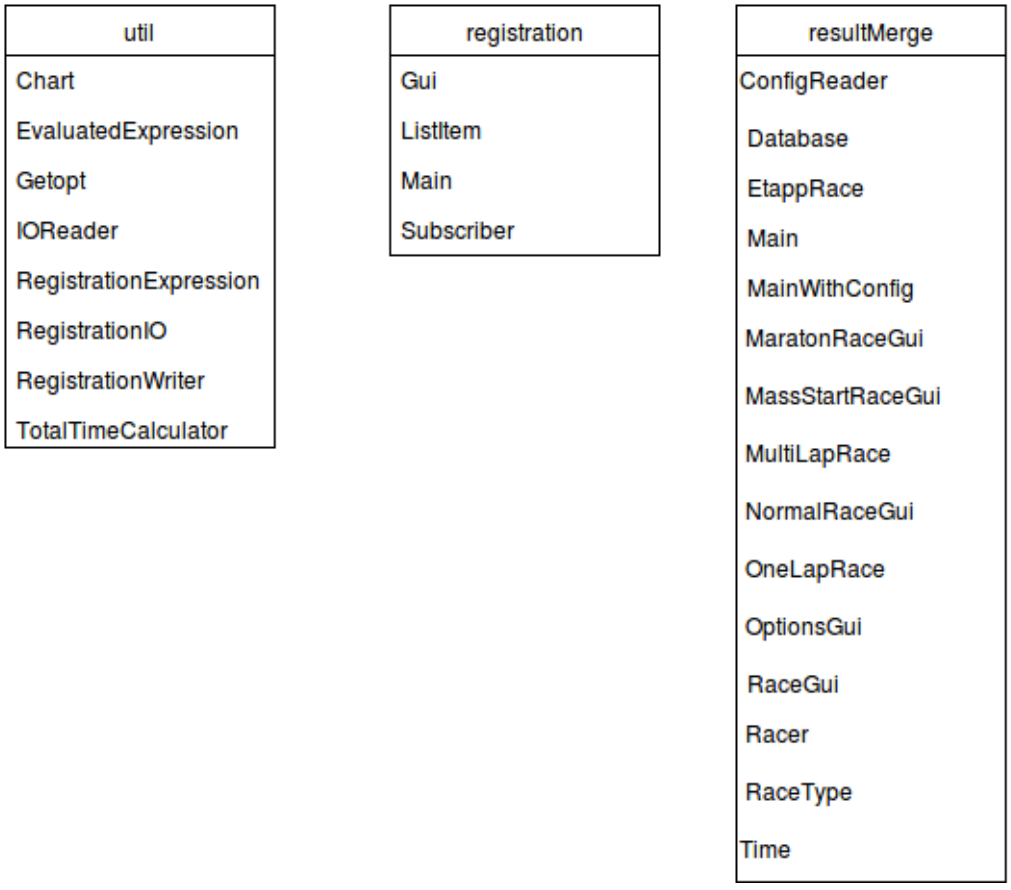
2 Designbeskrivning

Vårt program DasUberEnduroApplicationen består av 3 paket, util, registration och resultMerge. Hela programet använder sig av designmönstret Model-view-controllerFigur 1 visar alla paket samt listor av 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 och MultiLapRace. 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 har ansvar 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 denna tävlingssorten.
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. Den genererar en header beroende på vilken tävlingssort det är och itererar sedan över databasen med Racers och kallar sedan på varje Racers toString()-metod. IOReader-klassen i util-paketet står för inläsning av textfiler. För att programmet ska fungera behövs en namnfil, en starttidsfil och en sluttidsfil. Dessa måste vara i textformat och formatterade precist efter en viss standard. Detta för att vår IOReader ska kunna läsa korrekt.
Racer-klassen implementerar 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 och maratonlopp.

3 Klasser och filstruktur





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

3.1 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. 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.

3.2 resultMerge

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

EtappRace Funktionalitet för etapprace. Implementerar RaceType. Stöds ej för tillfället (se Teknisk doumentation).

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

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

RaceGui Gui för resultMerge-programmet.

MaratonRaceGui Gui för maratonlopp.

MassStartRaceGui Gui för masstart.

NormalRaceGui Gui för ett varvs lopp.

Main main för resultMerge utan stöd för konfiguration.

MainWithConfig main för resultMerge med stöd för konfiguration.

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 Stöds ej för tillfället. Är tänkt att representera tider generellt sätt för hela programmet och implementeras i samband med etapplopp.

3.3 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.

Getopt Parsar argument från kommandoraden.

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.

4 Program och filer

4.1 Program

Vår lösning består av två 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 behöver ett antal input filer för att kunna generera ett resultat. Input filerna är starttidermåltiderssom genereras av föregående program. Det behövs även en namnlista som kan länka ett startnummer med ett namn och klass. Det behövs också en config fil som bestämmer olika parametrar till tävlingen. Som tillexempel tävlingstyp, minimum tid för ett varv, namnen på de andra inputfilerna.

4.2 Filer

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

txt Det behövs några input txt filer och det genereras ett antal output txt filer. De som behövs av reultatprogrammet är tre filer. Dessa är starttider,sluttider och namn. 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 gradlewsom genererar körbara JAR filer av vårt projekt. Det finns två versoiner av denna filen. 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

4.3 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 behövs 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:

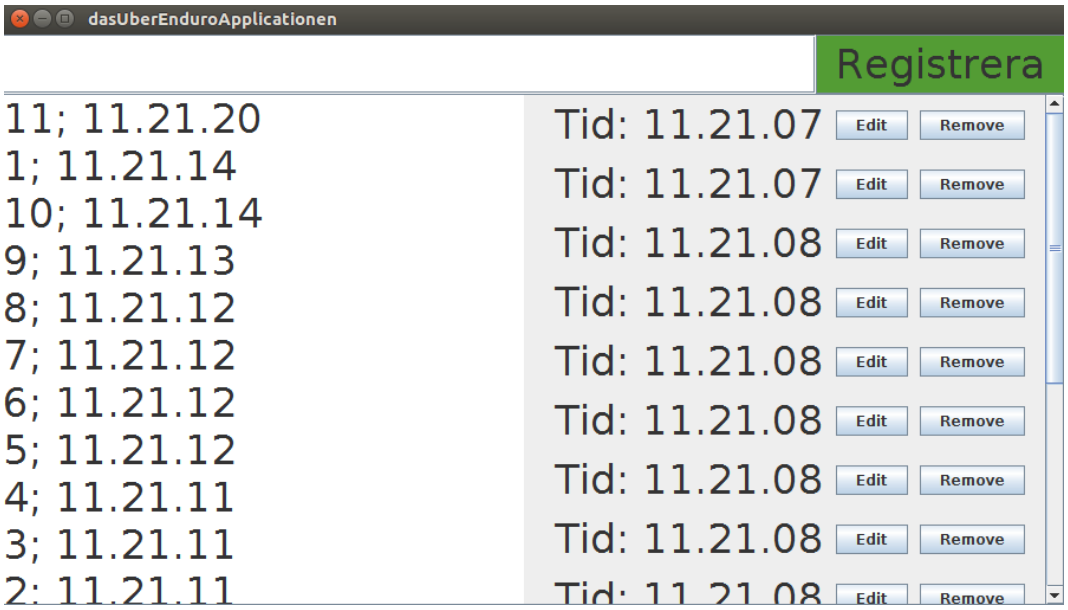
- output.txt - sorterad resultatfil
- output_unsorted.txt - osorterad resultatfil
- output.html - sorterad resultatfil på .html format

4.4 Distribution av programvaran

Det finns två gradle filer för att bygga programmet på antingen en UNIXdator eller en Windowsdator, men rätt fil väljs automatiskt. Allt en användare behöver göra är att öppna ett terminal fönster, positionera sig i dasUberApplikationen 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.

5 Gui

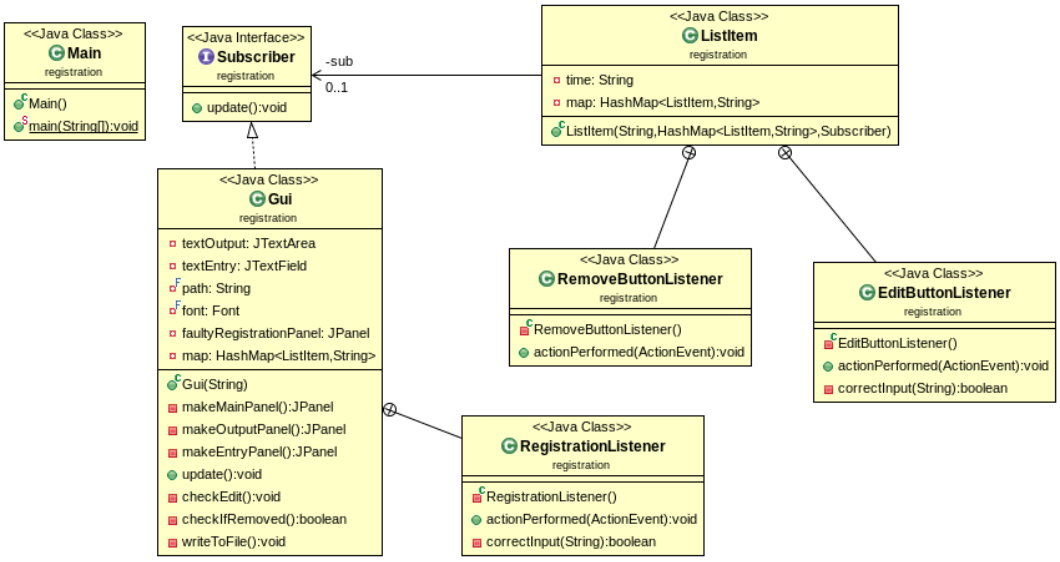
Figur 2 visar hur användargränssnittet ser ut för vårt program. I fältet högst upp skriver man in startnummer för varje Racer. När man klickar på Registrera hamnar den racern i kolumnen till vänster och tilldelas en starttid. Om man inte kan se startnumret klickar man bara på Registrera utan att ange 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.

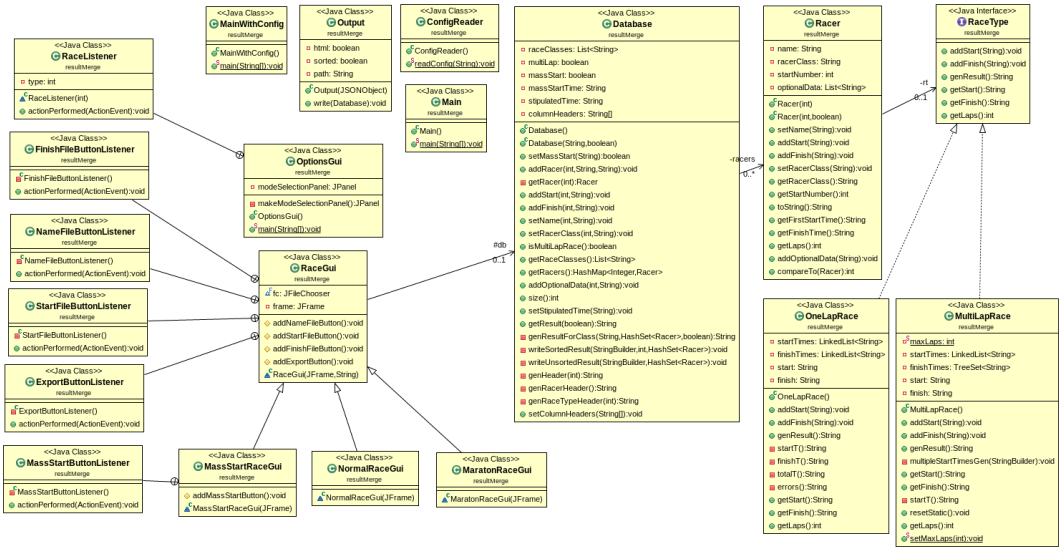
6 UML-diagram

6.1 Gui



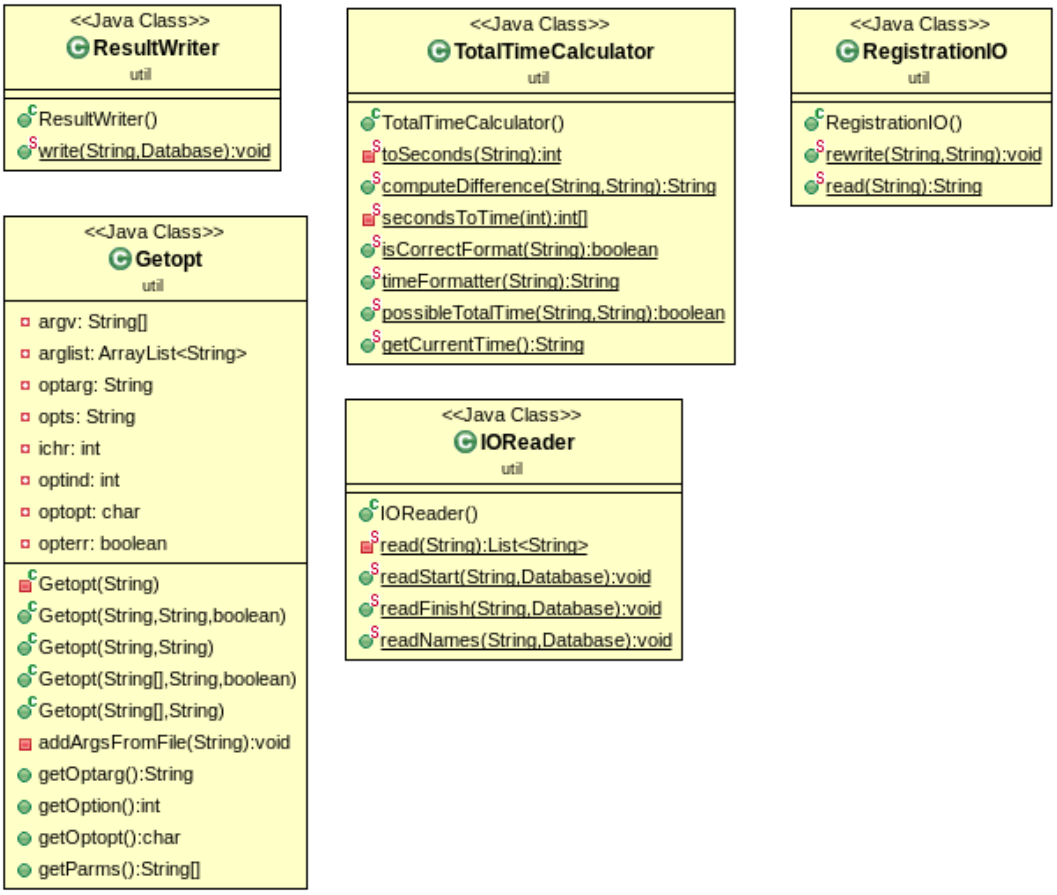
Figur 3: UML av GUI-paketet.

6.2 ResultMerge



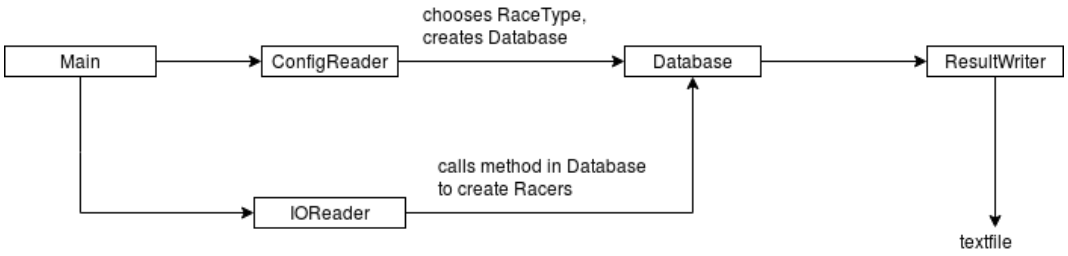
Figur 4: UML av ResultMerge-paketet.

6.3 Util



Figur 5: UML av Util-paketet.

6.4 Flow



Figur 6: Programflöde från inläsning till utskrift.

7 Framtida projektplaner (Work in progress)

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

- **Etapplopp** - Vi har börjat implementera logiken för etapplopp, men det saknas helt konfigurationsstöd. Planen är att detta ska följa med i Release 2.

- **Main** - För tillfället håller vi på att byta hur programmet startas, från parametrar till en konfigurationsfil. Detta är i princip klart men inte stabilt ännu.
- **Mer generella tider** - För att etapplopp ska kunna fungera måste vi kunna skicka mer information än bara tiden. Vi har därför börjat implementera denna struktur, men snart krävs en stor refaktorisering eftersom båda sätten att representera tider används för tillfället i de respektive RaceType-klasserna.
- **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.