KANDIDAT

10011

PRØVE

IDATT1001 1 Programmering 1

Emnekode	IDATT1001
Vurderingsform	Hjemmeeksamen
Starttid	16.12.2021 08:00
Sluttid	16.12.2021 12:00
Sensurfrist	16.01.2022 22:59
PDF opprettet	20.04.2022 11:07

Forside

Oppgave	Tittel	Oppgavetype
i	Forside Hjemmeeksamen	Informasjon eller ressurser

Javadoc API

Oppgave	Tittel	Oppgavetype
1(a)	Oppgave 1 a)	Langsvar
1(b)	Oppgave 1 b)	Langsvar

Oppgave 2

Oppgave	Tittel	Oppgavetype
2(a)	Oppgave 2 a)	Programmering
2(b)	Oppgave 2 b)	Programmering
2(c)	Oppgave 2 c)	Programmering

Oppgave 3 - Modellering

Oppgave	Tittel	Oppgavetype
3(a)	Oppgave 3a)	Filopplasting
3(b)	Oppgave 3b)	Langsvar

Oppgave 4 - Refleksjon

Oppgave	Tittel	Oppgavetype
4	Oppgave 4 - Refleksjon	Langsvar

Opplasting av Prosjektfil

Oppgave	Tittel	Oppgavetype
5	Opplasting av Prosjekt	Filopplasting

1

Oppgave 1

En viktig ferdighet som programvareutvikler er å finne frem i dokumentasjonen av biblioteker og rammeverk som kan være nyttig å bruke i din løsning.

I denne oppgaven skal du slå opp i dokumentasjonen for **Java Development Kit (JDK API)** og se nærmere på klassen **LocalTime**.

JDK-Dokumentasjonen er lastet ned på eksamens-PC'en du sitter på, og er tilgjengelig fra skrivebordet.

(a) Oppgave 1 a)

Gitt følgende kodelinje:

```
LocalTime time = LocalTime.parse("10:23:15");
```

Hvilken metode i klassen LocalTime kan du bruke for å hente ut minuttene av tidspunktet over? Gi eksempel på bruk av denne metoden (f.eks. ved å skrive kodelinjen som skriver ut minuttene til konsollet).

Skriv ditt svar her

The method you could use to retrieve the minutes, or minute-of-hour field, of the LocalTime object time is getMinute(). In practice, this can be seen as follows:

System.out.println(time.getMinute());

This line of code prints out the object time's minute-of-hour field to the console.

Maks poeng: 5

(b) Oppgave 1 b)

Dersom vi skal sammenligne to tider for å finne ut hvilket tidspunkt som er tidligere eller seinere enn et annet, hvilke metode(r) i LocalTime-klassen kan jeg bruke ?

Skriv hele **signaturen** til metoden(e), med en kort forklaring i egne ord hva metoden(e) eventuelt returnerer.

Skriv ditt svar her

The first method you could use in order to determine the chronological order of times, which one occurs earlier or later, is by using the LocalTime method compareTo(LocalTime other). This method's method signature is:

public int compareTo(LocalTime other)

The compareTo() method takes two LocalTime objects and checks to see if the LocalTime discussed is earlier, same time as, or later than the LocalTime parameter, other. Based on the answer, the method returns an int with the respective number -1, 0, 1.

Another method you could use to check if one LocalTime object is earlier or later than another LocalTime object is by using the method isAfter(LocalTime other). This method's method signature is:

public boolean isAfter(LocalTime other)

Similar to compareTo(), the isAfter() method takes two LocalTime objects and looks at their time variables relative to each other. However, instead of returning an int for whether the non-parameter LocalTime object occurs before, equal to, or after the parameter LocalTime, the isAfter() method returns a boolean. isAfter() checks if the non-parameter object occurs after the parameter object. If so, then the method returns the boolean true. If the non-parameter object occurs at the same time as or before, then the method returns false.

Finally, the method isBefore(LocalTime other) can also be used to check two LocalTime object's time field relative to each other. The method's method signature is:

public boolean isBefore(LocalTime other)

Again, similar to the isAfter(), the method isBefore() returns a boolean value. However, instead of returning true if the non-parameter object occurs after the parameter object, the isBefore() returns true if it occurs before. Furthermore, if the object's times are the same or if the non-parameter object occurs after, then false is returned.

Maks poeng: 5

2

Oppgave 2

Programmering

I resten av oppgavene skal du utvikle en applikasjon som skal benyttes under idrettsarrangementer.

Systemet som skal utvikles skal benyttes til å registrere resultater fra **lengedehopp med tilløp** (engelsk: long jump).

Nedenfor følger kravspesifikasjonen til den endelige løsningen du skal lage gjennom de neste oppgavene i oppgavesettet. NB! Les over **alle** oppgavene videre i oppgavesettet **før** du begynner på løsningen.

Kravspesifikasjon

Følgende informasjon skal registreres for hvert hopp i konkurransen:

- Startnummer til deltageren. Kan være etternavnet til deltakeren (som vist i bildet over), eller et nummer. (engelsk: start number)
- Navnet til deltager, på formen "fornavn etternavn" (engelsk: name of the athlete)
- Resultat (antall meter hoppet), som f.eks. 8,69 m. (engelsk: result)
- Gyldig hopp. Dersom deltakeren ikke treffer "planken" innenfor kravene, dømmes hoppet som ugyldig. (engelsk: faul).
- Tidspunktet (klokkeslett) for hoppet. (engelsk: time)

Eksempel på registrerte resultater fra lengdehopp kvinner under sommer-OL 2020:

Startnummer	Navn	Resultat (meter)	Gyldig hopp	Tid (hh:mm)
210	Malaika Mihambo	6,98	Ja	10:15
211	Tara Davis	6,85	Ja	10:17
204	Brittney Reese	6,52	Ja	10:19

Startnummer	Navn	Resultat (meter)	Gyldig hopp	Tid (hh:mm)
224	Khaddi Sagnia	6,76	Ja	10:21
211	Tara Davis	6,42	Nei	10:24
210	Malaika Mihambo	6,56	Ja	10:30
204	Brittney Reese	6,86	Ja	10:34
224	Khaddi Sagnia	6,65	Nei	10:37
210	Malaika Mihambo	6,12	Ja	10:40

Den endelige løsningen skal ha følgende funksjonalitet:

Brukeren/funksjonæren må kunne:

Vurderingskriterier

Din besvarelse vil bli vurdert basert på følgende kriterier:

- Om du har fulgt (og holdt deg til) kravspesifikasjonen
- Om du viser god forståelse for grunnleggende prinsipper i programmering som variabler, datatyper, metoder, betingelser, løkker osv.
- Om du har fulgt prinsippene for god design (coupling, cohesion, responsibility driven design osv)
- Om du har valgt gode, selvforklarende navn på klasser, metoder og variabler/felt/parametre
- Om du har dokumentert koden din godt (Javadoc)
- Om du har god og fornuftig samhandling med bruker (et godt brukergrensesnitt)

(a) Oppgave 2 a)

Bruk ditt foretrukne IDE (BlueJ, Visual Studio Code, IntelliJ osv). Opprett et nytt tomt prosjekt under din bruker på eksamens-PC'en.

1. Lag en klasse for å representere et registrert lengdehopp. (engelsk: long jump result)
2. Definer fornuftige felt av relevante datatyper (for tid kan du f.eks. bruke klassen **LocalTime** fra oppgave 1)
3. Opprett nødvendige metoder (konstruktør(er), aksessor- og eventuelt mutator-metoder) og dokumenter både klassen og metodene dine iht JavaDoc-standarden.

Når klassen din er kodet i ditt IDE, kopierer du koden og limer inn i svarfeltet under.

Lim inn koden din her..

```java
import java.time.LocalTime;

/**
 * This class represents a long jump result entry. It, therefore, handles all the
 * as the athlete's start number, their name, the jump length, whether the jump w
 * No mutator methods were used in this class since once the entry has been place
 *
 * @author 10011
 */
public class LongJumpResult {
    private final String START_NUMBER; //Can be the athletes last name or a numbe
    private final String ATHLETE_NAME; //Should be full name with first and last
    private final double JUMP_RESULT;
    private final boolean FOUL;
    private final LocalTime TIME_OF_JUMP;

    /**
     * This is a constructor representing the object LongJumpResult. It, therefor
     * to describe a long jump result. Additionally, the constructor checks for i
     * number or athlete name is left blank or if the jump result is a negative n
     * exception is thrown.
     * @param START_NUMBER A string representing an athletes start number.
     * @param ATHLETE_NAME A string for the athlete's name.
     * @param JUMP_RESULT A double for the length jumped by the athlete.
     * @param FOUL A boolean stating whether the jump was a foul or not. If true,
     * @param TIME_OF_JUMP The time of the jump in hours:minutes:seconds.
     */
    public LongJumpResult(String START_NUMBER, String ATHLETE_NAME, double JUMP_R
        TIME_OF_JUMP) {
        if(START_NUMBER.isBlank()) throw new IllegalArgumentException("The start
        if(ATHLETE_NAME.isBlank()) throw new IllegalArgumentException("The athlet
        if(ATHLETE_NAME.split(" ").length < 2) throw new IllegalArgumentException
                "and last name for the athlete");
        if(JUMP_RESULT < 0) throw new IllegalArgumentException("The jump result c
        this.START_NUMBER = START_NUMBER.toUpperCase();
        this.ATHLETE_NAME = ATHLETE_NAME.toUpperCase();
        this.JUMP_RESULT = JUMP_RESULT;
        this.FOUL = FOUL;
        this.TIME_OF_JUMP = TIME_OF_JUMP;
    }

    /**
     * This is a constructor which copies all of the primitive variables/values o
     * using those variables. It takes in a LongJumpResult object and uses the co
     * this(), to create a new object.
     * @param longJumpResult A LongJumpResult object, which will have its variabl
     */
    public LongJumpResult(LongJumpResult longJumpResult) {
        this(longJumpResult.getSTART_NUMBER(), longJumpResult.getATHLETE_NAME(),
            longJumpResult.isFOUL(), longJumpResult.getTIME_OF_JUMP());
    }

    /**
     * This method retrieves the start number of the athlete.
     * @return A string containing the start number of the athlete.
     */
    public String getSTART_NUMBER() {
        return START_NUMBER;
    }

    /**
     * This method retrieves the athletes full name.
     * @return A string containing the athlete's name.
     */
    public String getATHLETE_NAME() {
        return ATHLETE_NAME;
    }

    /**
     * This method retrieves the jump result of a given athelete.
     * @return A double representing the length jumped.
     */
    public double getJUMP_RESULT() {
        return JUMP_RESULT;
    }

    /**
     * This method retrieves the status of whether the jump of an athlete was a f
     * @return A boolean representing the validity of the jump. If valid, then fa
     */
    public boolean isFOUL() {
        return FOUL;
    }

    /**
     * This method retrieves the time of the jump.
     * @return A LocalTime object containing the time of the athlete's jump.
     */
    public LocalTime getTIME_OF_JUMP() {
        return TIME_OF_JUMP;
    }

    /**
     * This is a toString, where the given object's object variables are returned
     * here in order to minimize the amount of garbage collection needed, since i
     * @return A string with LongJumpResult's information.
     */
    @Override
    public String toString() {
        String jumpStatus = "Valid jump";
        if(FOUL) jumpStatus = "Invalid jump";
        StringBuilder sb = new StringBuilder();
        sb.append("Athlete:  ").append(ATHLETE_NAME).append("\nJump: ").append(JU
        sb.append(jumpStatus).append("\nTime of ").append("jump:  ").append(TIME_
        return  sb.toString();
    }
}

```

Maks poeng: 10

(b) Oppgave 2 b)

Løsningen din trenger også å inneholde et **register** for å holde på alle resultatene.
Registeret må som minimum inneholde følgende metoder:

- finne det beste gyldige hoppet (det lengste)
- søke etter alle hopp utført av en gitt deltager

Utover dette står du helt fritt til å velge hvordan du implementerer dette registeret, og hvilke øvrige metoder du tenker registeret bør ha for at den endelige løsningen din skal fylle kravene til funksjonalitet i kravspesifikasjonen.

Bruk Javadoc for klassen til å beskrive (og begrunne) hvilke metoder du mener registeret bør ha. Begrunn også her ditt valg av klasse fra "*Java Collection Framework"* (ArrayList, HashMap, HashSet osv).

Løs oppgaven i ditt IDE (BlueJ, Netbeans, IntelliJ osv), og kopier deretter koden for hele register-klassen inn i svarfeltet under.

Skriv ditt svar her

```java
import java.time.LocalTime;
import java.util.*;

/**
 * This is a class representing the register for a long jump event. It, therefore
 * different long jump entries/results of the event. An arrayList is used since i
 * can therefore be made longer if necessary, or removed if needed. In that sense
 * flexibility when handling the entries.
 *
 * @author 10011
 */
public class JumpResultRegister{
    private final List<LongJumpResult> JUMP_RESULT_REGISTER;
    private final String NAME_OF_EVENT;

    /**
     * This is the constructor representing a register. Therefore, an arrayList i
       JumpResultRegister
     * object is instantiated.
     * @param NAME_OF_EVENT The name of the event, f.eks Olympic Games 2021.
     */
    public JumpResultRegister(String NAME_OF_EVENT) {
        this.JUMP_RESULT_REGISTER = new ArrayList<>();
        this.NAME_OF_EVENT = NAME_OF_EVENT;
    }

    /**
     * This method registers a new LongJumpResult object to the JUMP_RESULT_REGIS
     * duplicates of the same jump, the input provided is check to see if there a
     * checkSameNameAndNum method {@link #checkSameNameAndNum(String, String)} is
     * has only one start number.
     * @param START_NUMBER A string with start number of athlete.
     * @param ATHLETE_NAME A string with the athlete's name.
     * @param JUMP_RESULT A double with the jump result.
     * @param FOUL A boolean of whether the jump was valid or not.
     * @param TIME_OF_JUMP A LocalTime object, showing the when the jump took pla
     * @return A boolean stating whether the jump result was successfully added t
     */
    public boolean registerNewJump(String START_NUMBER, String ATHLETE_NAME, doub
        TIME_OF_JUMP){
        LongJumpResult longJumpResult = new LongJumpResult(START_NUMBER, ATHLETE_
        if(this.JUMP_RESULT_REGISTER.contains(longJumpResult)){
            return false;
        }
        if(!checkSameNameAndNum(ATHLETE_NAME, START_NUMBER)){
            return false;
        }
        this.JUMP_RESULT_REGISTER.add(longJumpResult);
        chronologicalResults();
        return true;
    }

    /**
     * This method returns a deep copied version of the register.
     * @return The main register as a list containing LongJumpResult.
     */
    public List<LongJumpResult> listAllJumps(){
        return deepCopyList(this.JUMP_RESULT_REGISTER);
    }

    /**
     * This method returns all the jump attempts/results of a given athlete. The
     * entry in the registry and checking if the input athlete has the same name
     * @param athleteName A string for athlete's name.
     * @return A deep copied list of all the entries that the athlete has on the
     *          LongJumpResult objects.
     */
    public List<LongJumpResult> resultByAthlete(String athleteName){
        if(athleteName.isBlank()) throw new IllegalArgumentException("Please ente
        List<LongJumpResult> athletesResult = new ArrayList<>();
        for(LongJumpResult longJumpResult : JUMP_RESULT_REGISTER){
            if(longJumpResult.getATHLETE_NAME().equals(athleteName.toUpperCase())
                athletesResult.add(longJumpResult);
            }
        }
        return deepCopyList(athletesResult);
    }

    /**
     * This method find the best three jump results in all the entries within the
     * @param numberOfPlaces The number of best results
     * @return A list of the three best jumps, the list contains LongJumpResult o
     */
    public List<LongJumpResult> bestResults(int numberOfPlaces){
        List<LongJumpResult> sortByResults = this.JUMP_RESULT_REGISTER;
        Collections.sort(sortByResults, Comparator.comparing(LongJumpResult :: ge
        List<LongJumpResult> topThreeResults = new ArrayList<>();

        int index = 0;
        while(index < this.JUMP_RESULT_REGISTER.size() && index < numberOfPlaces)
            topThreeResults.add(sortByResults.get(index));
            index++;
        }

        return deepCopyList(topThreeResults);
    }

    /**
     * This method makes sure the main register is always in chronological order
     */
    private void chronologicalResults(){
        Collections.sort(this.JUMP_RESULT_REGISTER, Comparator.comparing(LongJump
    }

    /**
     * This method makes sure that an athlete does not have a different start num
       name
     * of the athlete is the same as an earlier entry, but the start number is di
     * @param nameOfAthlete Name of the athlete.
     * @param numberOfAthlete Start number of the athlete.
     * @return A boolean, stating false if the athlete has an entry from before w
       otherwise, true.
     */
    private boolean checkSameNameAndNum(String nameOfAthlete, String numberOfAthl
        for(LongJumpResult longJumpResult : this.JUMP_RESULT_REGISTER){
            if(nameOfAthlete.equals(longJumpResult.getATHLETE_NAME())){
                if(!numberOfAthlete.equals(longJumpResult.getSTART_NUMBER())){
                    return false;
                }
            }
        }
        return true;
    }

    /**
     * This method calculates the average jump length of all the entries in the r
     * by adding up all the jumps and then dividing that total by the number of e
     * @return A double showing the average jump length of the registered jumps.
     */
    public double calcAverageJump(){
        double totalJumpLength = 0;
        for(LongJumpResult longJumpResult : this.JUMP_RESULT_REGISTER){
            totalJumpLength += longJumpResult.getJUMP_RESULT();
        }
```

```java
        double averageJumpLength = totalJumpLength / this.JUMP_RESULT_REGISTER.si
        return averageJumpLength;
    }

    /**
     * This method creates a list of all the events that occur before the given t
     * @param time The time for which the user wants to check the events before.
     * @return A list of all the events before the given time.
     */
    public List<LongJumpResult> eventsBefore(LocalTime time){
        List<LongJumpResult> eventsBeforeList = new ArrayList<>();
        for(LongJumpResult longJumpResult : JUMP_RESULT_REGISTER){
            if(longJumpResult.getTIME_OF_JUMP().isBefore(time)){
                eventsBeforeList.add(longJumpResult);
            }
        }
        return deepCopyList(eventsBeforeList);
    }

    /**
     * This method creates a list of all the events that occur after the given ti
     * @param time The time for which the user wants to check the events after.
     * @return A list of all the events after the given time.
     */
    public List<LongJumpResult> eventsAfter(LocalTime time){
        List<LongJumpResult> eventsAfterList = new ArrayList<>();
        for(LongJumpResult longJumpResult : JUMP_RESULT_REGISTER){
            if(longJumpResult.getTIME_OF_JUMP().isAfter(time)){
                eventsAfterList.add(longJumpResult);
            }
        }
        return deepCopyList(eventsAfterList);
    }

    /**
     * This is a method that deepCopies a given list with object type LongJumpRes
     * to be made in the different methods in this class, this method reduces rep
     * @param longJumpList A list of LongJumpResult objects.
     * @return A deep copied version of the list sent in.
     */
    private List<LongJumpResult> deepCopyList(List<LongJumpResult> longJumpList){
        List<LongJumpResult> copiedList = new ArrayList<>();
        for(LongJumpResult longJumpResult : longJumpList){
            copiedList.add(new LongJumpResult(longJumpResult));
        }
        return copiedList;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("\t\t").append(this.NAME_OF_EVENT).append("\n");
        for(LongJumpResult longJumpResult : this.JUMP_RESULT_REGISTER){
            sb.append(longJumpResult).append("\n");
        }
        return sb.toString();
    }
}
```

Maks poeng: 25

(c) Oppgave 2 c)

I denne oppgaven skal du ferdigstille applikasjonen din med et **brukergrensesnitt** i henhold til kravspesifikasjonen.

Vi har laget ferdig *rammeverket* for et menybasert brukergrensesnitt som du kan ta utgangspunkt i. I Ålesund og på Gjøvik har vi brukt et tekstbasert brukergrensesnitt i konsollet. I Trondheim har dere benyttet *JOptionPane*-klassen for å lage et forenklet grafisk brukergrensesnitt:

- For dere i Ålesund og Gjøvik: LongJumpUI.java
- For dere i Trondheim: LongJumpGUI.java

NB! Når du laster ned filen, vil filnavnet være tilgriset med Inspera-generert støy og ikke lett å gjenkjenne som en Java-fil. Endre filnavnet til et av navnene over, så er filen klar til bruk.

Når du har programmert ferdig brukergrensesnittet i din IDE, kopierer du **hele klassen** og limer inn i svarfeltet under:

Lim inn klassen din her...

```java
import java.time.LocalTime;
import java.util.Scanner;

/**
 * This is the basic user interface for the Long Jump Register application.
 *
 * @author 10011
 */
public class LongJumpGUI {

    private static Scanner input = new Scanner(System.in);
    private static final int ADD_RESULT = 1;
    private static final int LIST_ALL_RESULTS = 2;
    private static final int SHOW_RESULT_BY_ATHLETE = 3;
    private static final int SHOW_SINGLE_BEST = 4;
    private static final int SHOW_BEST_RESULTS = 5;
    private static final int CALCULATE_AVERAGE_RESULT = 6;
    private static final int FIND_BEFORE_TIME = 7;
    private static final int FIND_AFTER_TIME = 8;
    private static final int EXIT = 9;
    private final JumpResultRegister RESULT_REGISTER;

    public LongJumpGUI() {
        this.RESULT_REGISTER = new JumpResultRegister("Olympic Games 2021");
    }

    /**
     * This method shows the menu to the user and then waits for the user to ente
     * That input is then used as the number of the case wanted and is returned b
     * a correct input type.
     *
     * @return The menu choice input by user
     */
    private int getMenuChoice() {
        int menuChoice = 0;
        System.out.println("\n***** Long Jump Register Application v0.1 *****\n")
        System.out.println("1. Add a new result");
        System.out.println("2. Show all the current entries");
        System.out.println("3. Show all the entries of a given athlete");
        System.out.println("4. Show the best entries");
        System.out.println("5. Show the top 3 best entries");
        System.out.println("6. Show the average length jumped");
        System.out.println("7. Show events before a given time");
        System.out.println("8. Show events after a given time");
        System.out.println("9. Quit");
        System.out.println("\nPlease enter a number between 1 and 9.\n");
        Scanner sc = new Scanner(System.in);
        if (sc.hasNextInt()) {
            menuChoice = sc.nextInt();
        } else {
            System.out.println("You must enter a number, not text");
        }
        return menuChoice;
    }

    /**
     * This method starts the program by starting a while loop, which goes throug
     * retrieves the menu choice by calling getMenuChoice {@link #getMenuChoice()
     *
     * Starts the application. This is the main loop of the application,
     * presenting the menu, retrieving the selected menu choice from the user,
     * and executing the selected functionality.
     */
    public void start() {
        boolean finished = false;
        while (!finished) {
            int menuChoice = this.getMenuChoice();
            try {
                switch (menuChoice) {
                    case ADD_RESULT:
                        addNewResult();
                        break;

                    case LIST_ALL_RESULTS:
                        listAllResults();
                        break;

                    case SHOW_RESULT_BY_ATHLETE:
                        showResultByAthlete();
                        break;
                    case SHOW_SINGLE_BEST:
                        showSingleBestResult();
                        break;

                    case SHOW_BEST_RESULTS:
                        showBestResults();
                        break;

                    case CALCULATE_AVERAGE_RESULT:
                        calculateAverageResult();
                        break;

                    case FIND_BEFORE_TIME:
                        beforeTime();
                        break;

                    case FIND_AFTER_TIME:
                        afterTime();
                        break;

                    case EXIT:
                        System.out.println("Thank you for using this application!
                        finished = true;
                        break;

                    default:
                        System.out.println("Unrecognized menu selected..");
                        break;
                }
            }
            catch(Exception e){
                e.printStackTrace();
            }
        }
    }

    /**
     * This method obtains all the input necessary to add a new result into the r
     * to attempt to register the entry via the registerNewJump method
     * {@link JumpResultRegister#registerNewJump(String, String, double, boolean,
     */
    private void addNewResult(){
        System.out.println("First, to add a new entry into the register, you need
            +
                "This may either be a number or the last name of the athlete.");
        String startNum = input.nextLine();
        System.out.println("Now, enter the athlete's full name.");
        String athleteName = input.nextLine();
        System.out.println("Please, enter the jump distance in meters: f.eks 5.87
        double jumpLength = input.nextDouble();
        input.nextLine();
        System.out.println("If the entry was valid, type (Y)es. If it was invalid
        String foulString = input.nextLine().toUpperCase().trim();
        boolean foul = false;
        if(foulString.charAt(0) == 'N'){
            foul = true;
        }
        System.out.println("Finally, you need to enter the time of the jump.");
        LocalTime longJumpTime = jumpTime();

        if(this.RESULT_REGISTER.registerNewJump(startNum, athleteName, jumpLength
            System.out.println("You have successfully added a new jump to the reg
        }
        else{
            System.out.println("The logging of the jump was unsuccessfully. This
                .");
        }
    }

    /**
     * This method prints out all the information to all the entries in the regis
     * implicitly using the toString in the JumpResultRegister {@link JumpResultR
     * and the listAllJumps method {@link JumpResultRegister#listAllJumps()}.
     */
    private void listAllResults(){
        System.out.println("The following are all the entries registered: ");
        for(LongJumpResult longJumpResult : this.RESULT_REGISTER.listAllJumps()){
            System.out.println(longJumpResult + "\n");
        }
    }

    /**
     * This method prints out all the entries from a given athlete {@link JumpRes
     */
    private void showResultByAthlete(){
        System.out.println("What is the name of the athlete that you want to chec
        String athleteName = input.nextLine();
        System.out.println(this.RESULT_REGISTER.resultByAthlete(athleteName));
    }

    /**
     * This is the method used to find the single best result out of all the entr
         JumpResultRegister#bestResults(int)}.
     */
    private void showSingleBestResult(){
        System.out.println("The best result so far is: \n");
        System.out.println(this.RESULT_REGISTER.bestResults(1));

    }

    /**
     * This method prints out the best results out of all the entries in the regi
     * {@link JumpResultRegister#bestResults(int)}.
     */
    private void showBestResults(){
        System.out.println("The best results are: \n");
        for(int i = 0; i < this.RESULT_REGISTER.bestResults(3).size(); i++){
            System.out.println((i+1) + ". " + this.RESULT_REGISTER.bestResults(3)
        }
    }

    /**
     * This method prints out the average length of all the jumps in the registry
     * {@link JumpResultRegister#calcAverageJump()}.
     */
    private void calculateAverageResult(){
        System.out.println("The average length jumped was " + this.RESULT_REGISTE
    }

    /**
     * This method finds all the events before the time given by the user, throug
         ()}.
     */
    private void beforeTime(){
        System.out.println("To find the time before, first answer these questions
        System.out.println(this.RESULT_REGISTER.eventsBefore(jumpTime()));
    }

    /**
     * This method finds all the events after the time given by the user, through
        ()}.
     */
    private void afterTime(){
        System.out.println("To find the time after, first answer these questions:
        System.out.println(this.RESULT_REGISTER.eventsAfter(jumpTime()));
    }

    /**
     * This method asks the user for a time in hours and minutes. This time is th
     * object. If the hours or minutes are wrong, then an IllegalArgumentExceptio
     * @return A LocalTime object representing the time given by the user.
     */
    private LocalTime jumpTime(){
        System.out.println("What was the hour when the jump occurred? Type in the
        int hoursInt = input.nextInt();
        String hours = String.valueOf(hoursInt);

        System.out.println("What was the minutes when the jump occurred? Type in
        int minutesInt = input.nextInt();
        String minutes = String.valueOf(minutesInt);
        input.nextLine();

        if(hoursInt < 0 || hoursInt > 23) throw new IllegalArgumentException("You
        else if(hoursInt > 0 && hoursInt < 10){
            hours = "0" + hours;
        }

        if(minutesInt < 0 || minutesInt > 59) throw new IllegalArgumentException(
        else if(minutesInt > 0 && minutesInt < 10){
            minutes = "0" + minutes;
        }

        return LocalTime.parse(hours + ":" + minutes);
    }

}

```

Maks poeng: 25

3(a) Oppgave 3a)

Tegn et **klassediagram** i henhold til **UML-standarden** over løsningen du har implementert.

Det er tilstrekkelig at du tegner klassene kun med navn (ikke felt og metoder). Få med hvilke avhengigheter (avhengigheter, assosiasjoner osv) som finnes mellom klassene.

Du kan tegne diagrammet på papir som du "scanner" med mobiltelefonen din og laster opp her. Eller du kan benytte digitale verktøy for å tegne klasse-diagrammet. Eksporter diagrammet som bildefil (.jpg, .png el.l.) og last opp filen.

✓

Din fil ble lastet opp og lagret i besvarelsen din.

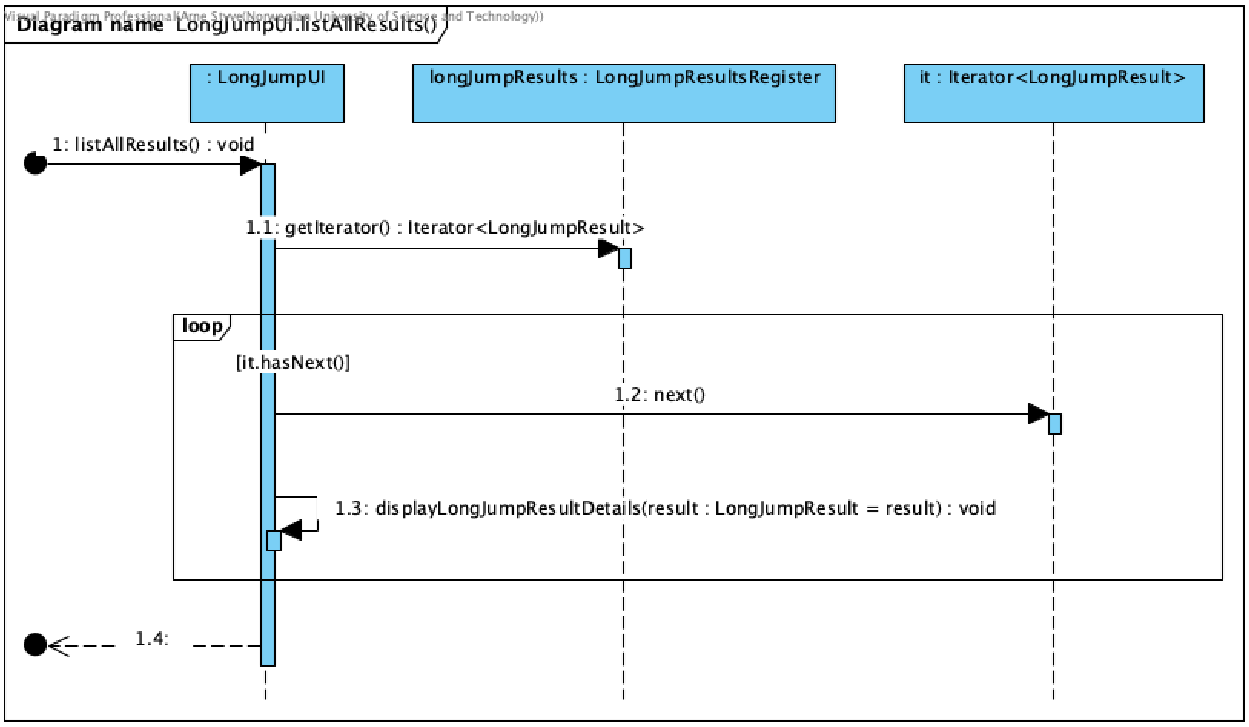
Last ned Fjern Erstatt

Filnavn:	Class Diagram.pdf
Filtype:	application/pdf
Filstørrelse:	280.24 KB
Opplastingstidspunkt:	16.12.2021 10:59
Status:	**Lagret**

Maks poeng: 5

3(b) Oppgave 3b)

Følgende UML-diagram er oppgitt:


Visual Paradigm Professional(Arne Styve(Norwegian University of Science and Technology))
Diagram name LongJumpUI.listAllResults()
: LongJumpUI
longJumpResults : LongJumpResultsRegister
it : Iterator<LongJumpResult>
1: listAllResults() : void
1.1: getIterator() : Iterator<LongJumpResult>
loop
[it.hasNext()]
1.2: next()
1.3: displayLongJumpResultDetails(result : LongJumpResult = result) : void
1.4:

Hva heter denne typen diagram, og hva viser diagrammet?

Skriv ditt svar her

This diagram is called an sequence diagram (or sekvensdiagram in Norwegian). It represents a potential way the program may run and the different interactions involved. In essence, it shows the course a program may take when communicating with other classes when different choices are taken by the user. It provides the reader with a more detailed depiction of how the program operates with the different creations of objects and the path the program takes when various methods are called.

Maks poeng: 5

4 Oppgave 4 - Refleksjon

Reflekter over hvordan du har valgt å løse prosjektet. Hva var den største utfordringen? Hva vil du fremheve som du synes du har løst på en god måte (hva har du gjort for å sikre høy kvalitet og en robust løsning) ?

Basert på klassediagrammet du har laget i oppgave 3a, kommenter på objekt interaksjon i henhold til design du har valgt å bruke.

Basert på hvordan du har valgt å løse prosjektet i denne oppgaven, hvordan vil du vurdere løsningen din i henhold til design-prinsippene **coupling**, **cohesion** og **responsibility driven design**? Er det noen av klassene du f.eks. tenker kunne vært bedre designet iht prinsippet om cohesion?

Dersom du hadde mer tid for å løse oppgave, hvilken deler av programmet vil du ha endret (refactoring), og hvorfor ?

Skriv ditt svar her

My biggest challenge while tackling this project was figuring out how to reduce repetitive functions and maintain a easily readable code. This can be clearly seen through my deep copy method. Instead of using a similar technique for deep copying every list that needed to be sent back to the GUI, I decided to create a separate function which tackles this problem. By doing this, I leave each method even more general, which (as I will discuss later) adds to the cohesion.

Furthermore, I have ensured high quality code through succinct names for variables and methods. This adds even more to the readability of the code. I have also tried to optimize the code's performance, which can be seen through the use of StringBuilder to append strings together instead of String and +.

I would like to bring focus to the additional attention that was put towards securing a composition relationship. Throughout learning about composition and aggregation, composition was presented as the goal due to its impact on coupling. However, I felt I never really managed to get true composition. Composition is a relationship where one class interacts with an object in a way that the object will "live" and "die" in that class only. What that means is that the object itself and its place in memory cannot be accessed from anywhere other than through the class. In this project, I focus on not sending any references to the objects in the register to the GUI or other classes. Since those objects can only be accessed through the register class, there exists a composition relationship. This relationship can be seen in the class diagram I drew. The composition relationship was achieved through deep copying every object that was returned, where a deep copy is a new object being created with the same primitive variables as another object.

I exercise good coupling throughout my program in many different ways. One of the most important ways I reduce coupling, or rather maintain loose coupling, is through using no direct coupling. If the user wants to access, LongJumpResult via the LongJumpGUI, then instead of making a direct connection to LongJumpResult the GUI must first go through the register. This may be seen with registerNewJump() where instead of adding the object from the GUI, I have created a method which adds it from the JumpResultRegister class. Furthermore, instead of using data-structure/stamp coupling, the program focuses on making data coupling. Data coupling makes for looser coupling and can be seen through methods taking in primitive parameters instead of composite (object) parameters. The best example for this is also registerNewJump() where the method takes in the parameters (String START_NUMBER, String ATHLETE_NAME, double JUMP_RESULT, boolean FOUL, LocalTime TIME_OF_JUMP) instead of (LongJumpResult ...). Finally, to make sure the coupling is even looser, I have limited the accessibility of different variables and methods between classes. By this, I mean that I have reduced content coupling by making variables and methods private and having getters for those private variables. This can be seen with all the private variables in LongJumpRegister. Overall, I have kept focus on a responsibility driven design for the code, which means that the different classes can operate with limited knowledge of each other's methods and results in loose coupling.

In regards to cohesion, the program focuses on maintaining high cohesion. First and foremost, on a class level, the program maintains high cohesion through having each class serve one purpose/represent one idea. An example of this is resultByAthlete(String athlete). This method does not concern itself about any of the other methods. It serves the one purpose of finding all the results of one athlete. On a deeper level, cohesion is promoted through reusing methods within a class, instead of having redundancies. As discussed earlier, this can be seen with my deep copy method. Distinct, well-named variables and methods also contribute to a higher cohesion.

If I had more time, I would focus on organizing the data in a certain way. For example, I do not have any exception handling for when a user does not enter a valid start number. Since the start number has to either be a number or their last name, a user could potentially just type a (which isn't a real last name). Other than error-proofing the code, I believe I could make an individual class for the athletes which would hold some information about them. That would make the register a little more detailed and could add another element to the program. It would also potentially eliminate some necessary error-handling. I would also like to have made the eventsBefore() and eventsAfter() methods into one more generalized method. This would further increase cohesion.

Maks poeng: 20

5 Opplasting av Prosjekt

Når du er ferdig med din besvarelse så lager du en **ZIP-fil** (IKKE RAR eller annet format!!) av ***hele prosjektmappen din*** med alle undermapper og IDE-spesifikke filer.

Du lager ZIP-fil av en mappe på følgende måte:

- I windows: høyreklikk på mappen i filutforskeren din, og velg "Send til->Komprimert (zippet) mappe"
- På MacOSX: høyreklikk mappen i Finder, og velg "Komprimer..."

Sjekk at størrelsen på ZIP-filen viser mer enn noen få bytes. Er du i tvil om du har fått ZIP'et alt? Pakk ut ZIP-filen til annet sted på din datamaskin og gå over innholdet. Ser alt greit ut, kan du levere inn ZIP-filen her.

✓

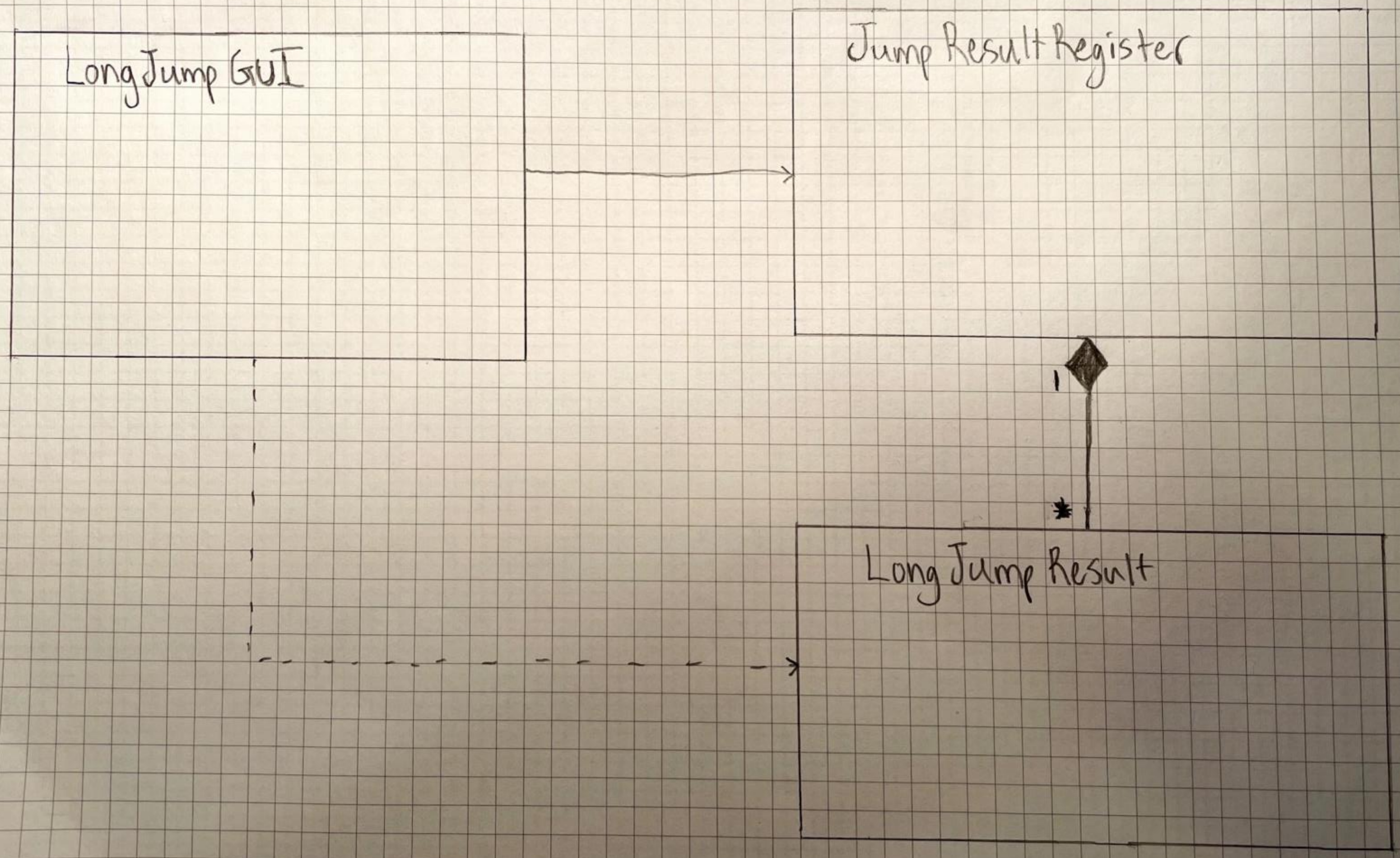
Din fil ble lastet opp og lagret i besvarelsen din.

Last ned Fjern Erstatt

Filnavn:	Programming 1 Exam 2021.zip
Filtype:	application/zip
Filstørrelse:	290.84 KB
Opplastingstidspunkt:	16.12.2021 12:41
Status:	**Lagret**

Maks poeng: 0

Class Diagram


LongJump GUI
Jump Result Register
Long Jump Result
1
*