תכנות ותכן מונחה עצמים 046271

תרגיל בית 1

מגישים :

עפר רוזנצווייג	מספר ת.ז: 066704727
ג'וני אשקר	מספר ת.ז: 205351026

תאריך הגשה: 13.12.2020

שאלה 1

א-)

שיקולי מימוש: על מנת להימנע מטעויות בחרנו להשתמש בכמה שפחות שדות, גם כאלה הניתנים לחישוב בזמן יצירת האובייקטים. כלומר, בהינתן שדות התלויים בשדות אחרים, בחרנו לא לממש את השדות התלויים, אלא לחשב אותם מחדש בעת הגישה אליהם. בצורה זו אנחנו ניגשים לשדות הללו רק בעת הגישה אליהם, ולא בפונקציות נוספות כגון הבנאי או ה-checkRep. מצאנו שהדבר מוריד את הסיכוי לטעויות ומעלה את הקריאות.

לעומת זאת החיסרון בגישה זו היא סיבוכיות זמן גבוהה יותר בחלק מן המקרים – וספציפית בפונקציה getGeoFeatures במחלקה Route.

בחירות בעקבות המימוש: במחלקות GeoFeature, Route בחרנו לא לממש את השדות start, end, startHeading, endHeading, name. הסיבה לכך היא שיותר קל לגשת ישירות ל segments ולחשב משם את השדה הדרוש בעת הצורך. הדבר יוצר פונקציית checkRep פשוטה ביותר וגורם לכך שכל פונקציה מטפלת אך ורק באיברים הנוגעים לה.

במחלקות GeoSegment מימשנו את השדות כמשתנים סופיים בשל הפשטות של המבנה.

במחלקה GeoFeature מימשנו את השדות בצורה עקיפה ולא החזקתי משתנים עבור רוב הנתונים. רק עבור שדה ה length החזקנו משתנה אמיתי בשביל להוריד מסיבוכיות הזמן.

במחלקה Route, בדומה ל GeoFeature, מימשנו את השדות בצורה עקיפה וחישבנו אותם בעת הצורך. בנוסף, כפי שנכתב קודם, בחרנו לתחזק רשימה של GeoSegment בלבד. הדבר מקל מאוד על פעולות הבנייה, ההשוואה והבדיקה.

בנוסף, את המחלקה route בחרנו לממש בעזרת רשימה של segments בלבד, ללא רשימה של geoFeatures. בעקבות זאת פונקציית ה checkRep נותרת פשוטה, והקוד היחידי כמעט בקוד שצריך בכלל להכיר בכלל במושג ה GeoFeature הוא הפונקציה getGeoFeatures.

חלופות למימוש: היינו יכול לבחור לממש את השדות כשדות אמיתיים במחלקות המורכבות. הדבר היה מביא לביצועים טובים יותר ומקל בדיבוג, אך קריאות הקוד הייתה פוחתת.

ב-)

1- בעצם התוספה פה האופציה שהפונקציה תוכל לקבל בנוסף לקלטים החוקיים שהיו לפני, את הקטעים שבהם הנקודה הסופית היא הנקודה הסופית בדרך הזו.
החלטנו שהטיפול במקרה זה יהיה טיפול במקרה שהוא חוקי להוספה, וצורת הטיפול (המימוש) תהיה כך: במידה ומתקבל קטע שבו הנקודה הסופית P2 שווה לנקודת סיום הסגמנט, נהפוך את הסגמנט בעזרת הפונקציה שמימושנו במחלקת GeoSegment ע"י יצירת סגמנט חדש שבו נקודות הקצה הפוכות. נקבל סגמנט שבו נקודת ההתחלה שווה לנקודת סיום הדרך הנוכחית.
את הסגמנט ההפוך שיצרנו נוסיף לדרך.
זה בעצם מאפשר לנו להוסיף סגמנטים לדרך עם רמת חופשיות מסוימת. כך שאם למשל קטע דרך מסוים מקובל/נוח/אינטואיטיבי יותר מבחינת המשתמשים בתוכנה לייצג אותו כקטע מ-A ל-B אז הם יוכלו לייצג אותו כך בלי להצטרך להפוך אותו בכדי שהמתודה תצליח לחבר אותו לדרך.
הערה: פסקת ה-return@ במפרט הולכת להשתנות בכך שהיא מסבירה שבמקרה החדש נוצרת דרך חדשה שסופה הוא תחילת המקטע והזווית סיום של הדרך תהיה הזווית הנמדדת כשהולכים בכיוון p2 ל- p1.

2- המפרט החדש **<u>חזק יותר</u>** מהמפרט שהיה קודם. ניתן להתייחס לשאלה בכמה אופנים:

- - פסקת ה- requires@ כעת היא פחות קפדנית דהיינו, מרחב הקלט התרחב והוא מכיל את מרחב הקלט שהיה קודם מה שאומר שהוא פחות מחמיר ולכן יותר חזק מה-requires@ שהיה קודם.
  - מרחב הפלט שהיה קודם עבור אותם קלטים חוקיים שהיו קודם נותר זהה. לכן ניתן להגיד שה return@ עבור אותו מרחב קלט ישן נותר זהה מבחינת חוזק.

בהסתכלות יותר כללית ניתן להגיד שזוהי פסקת return@ יותר ספיציפית לגבי הקלטים שהיא יוצרת עם שמירה על אותם פלטים בדיוק עבור אותם קלטים שהיו חוקיים קודם.

קיבלנו: requires@ יותר חזק ו- return@ לפחות עבור אותם קלטים זהה בחוזק או אפילו יותר ספציפי בהסתכלות כללית ולכן זהו בסה"כ מפרט יותר חזק מהמפרט הישן.

- דרך אחרת להסתכל על זה היא שאם מממשים את המתודה בצורה שתתמוך במפרט החדש אז היא תתמוך גם כן במפרט שהיה קודם.

- דרך שלישית היא לחשוב מהפרספקטיבי של משתמש בפונקציה. אם הוא השתמש זמן מה במפרט הישן ואז הוחלף המפרט והמימוש למימוש החדש לא נוצרים בעיות ולא יהיו פלטים "מפתיעים" מכיוון שפשוט המימוש החדש יתייחס לתת-קבוצה של הקלטים החוקיים הישנים והמימוש החדש יתייחס אליהם וימפה אותם לפלטים בדיוק באותו צורה שעשה קודם.

ג-)

התמיכה בשינוי נעשית ע"י קביעת הקונבנציה הבאה:
כיוונו של מקטע/דרך באורך אפס הוא זווית 0°

זה מה שיוחזר ע"י המתודה ()getHeading במידה והמקטע הנוכחי הוא מקטע מאורך 0 אבל מכיוון שהמימוש שלנו שומר את הכיוון כשדה פרטי, את הטיפול במקרה זה נעשה בבנאי של ה-GeoSegment. באחריות המשתמש לדעת שהאפס שמוחזר במקרה זה אינו ערך בעל משמעות הנדסית ע"י כך שיבדוק גם מהו אורכו של הסגמנט.

בנוסף נבצע את השינויים הבאים:

- הנ"ל מתייחס לשינויים ב Route וב- GeoFeature כך שעכשיו הם מחזירים פלטים יותר הגיוניים עבור הכיוונים בקצוות הדרך:
  * הכיוון ההתחלתי הוא הכיוון של הסגמנט הראשון שאורכו לא שווה לאפס
  * הכיוון הסופי הוא הכיוון של הסגמנט האחרון שאורכו לא שווה לאפס
  במידה וכל הסגמנטים בדרך או ב- GeoFeature אורכם אפס, כלומר, הדרך כולה אורכה אפס אז נתייחס לזווית ההתחלתית והסופית בצורה דומה להגדרת כיוונו של מקטע באורך אפס, קרי, במקרה הזה כיוון ההתחלה וכיוון הסיום הם 0°.

- מתודת ה-computeLine של מחלקות ה- formatters יתעלמו מקיומם של GeoFeatures בעלי אורך כולל אפס ולא תודפסנה בעת ההגעה אליהם הודעות ניווט למניעת יצירת הודעות חסרות תוכן.

ד-)

- האם Route הוא true subtype של GeoFeature?

לא! מכיוון שלכל אובייקט מטיפוס GeoFeature יש תכונה : שם המקטע הגיאוגרפי

למשתנים מטיפוס Route אין תכונה כזאת ולכן לא מתקיים התנאי של true subtype שהוא שמירה על המשמעויות של המתודות והשדות של מחלקת האב, אצל תת המחלקה.

- האם GeoFeature הוא true subtype של Route?

לא! מכיוון שמשמעות המתודה של הוספת מקטע (addSegment) משתנה. בפרט, מתודת הוספת מקטע עבור Route תוסיף מקטע כל עוד הקצה ההתחלתי שלו הוא סוף הדרך הנוכחית.
אותה מתודה ב- GeoFeature מוסיפה מקטע חדש אך ורק כאשר בנוסף לתנאי עבור הקצה ההתחלתי, גם שם המקטע זהה לשם ה GeoFeature כך שכל מימוש יצטרך או להחליש את המפרט ע"י הוספה לפסקת ה- @requires או טיפול במקרה נוסף בפונקציה מה שיצריך שהמשתמש תמיד יהיה חייב לדעת עם איזה actual type הוא עובד כדי שידע מה לצפות מפעולת המתודה.
ז"א התנאי עבור true subtype לא מתקיים גם בכיוון הזה.

ה-)

- **מחלקה שיכולה להיות צאצא של RouteFormatter:**

נלך על הדרך הנאיבית למציאת מחלקה כזו. לפי אותה תבנית שלפיה נבנתה המחלקה WalkingRouteFormatter למשל נגדיר מחלקה CyclingRouteFormatter
מחלקה זו תתן הנחיות לרוכב אופניים באותו אופן בדיוק ואותו מבנה הודעה כמו WalkingRouteFormatter רק שבמקום להציג בהודעה "and walk for" המחלקה הזו בעת הצגת ההודעות תציג "and cycle for".

מחרוזת מוחזרת מ- computeLine של מחלקה זו תהיה למשל:

Turn sharp left onto HaYovel and cycle for 14 mintues

לא תתבצע דריסה של אף אחת משתי הפונקציות שכבר מומשו במחלקה האבסטרקטית RouteFormatter ורק computeLine תמומש בדומה ל-WalkingRouteFormatter עם השינוי המינורי של "cycle" בהודעה.
הצגנו דרך ברורה למימוש ההיררכיה שבה תהיה CyclingRouteFormatter צאצא של RouteFormatter ולכן דוגמא זו היא אכן דוגמא טובה.

- **מחלקה <u>שלא</u> יכולה להיות צאצא של RouteFormatter:**

נגדיר את המחלקה PersonRouteFormatter
מחלקה זו מציגה עבור איש מסלול בתוך העיר. הצורה בה יוצג המסלול תלויה באם האיש הולך ברגל או שיש לו הובר בורד.

אם הוא הולך ברגל ההודעות שתודפסנה יהיו מהצורה:

Turn sharp right and walk for 24 minutes

אם יש לו הובר בורד אז ההודעות שתודפסנה הן מהצורה:

Turn left and go straight for 7 minutes

מחלקה זו:

- לא יהיה מבחינתה משמעות עבור מתודת חישוב מסלול שמקבלת רק את Route וכיוון התחלתי
- תצטרך שיהיה לה פונקציה נוספת שתקבל בנוסף ל- Route ולכיוון ההתחלתי עוד פרמטר שהוא הדרך שבה האיש ילך ברחוב. כל פעם שנרצה להציג מסלול נצטרך להשתמש בפונקציה הזו.

במקרה זה אי אפשר שרפרנס למחלקה RouteFormatter יחזיק PersonRouteFormatter ושנוכל להשתמש בו ללא קשר ל- actual type שלו כי בלי לדעת איזה טיפוס אמיתי מוחזק לא ידע באיזה פונקציה נשתמש. מה עוד שכל שימוש בפונקציית חישוב המסלול של המחלקה האבסטרקטית עלוללגרום להתנהגות לא מוגדרת.

שאלה 2

ב-)

הבעיה העיקרית היא שהחלון הראשי אחראי להצגת המידע עבור Route אחד בלבד.

נניח שיש לנו תוכנה שצריכה להציג למשתמש ממשק שבו הוא יוכל לערוך כמה מסלולים שונים אז הדרך היחידה לפי מימוש זה היא להשתמש ב RouteFormatterGUI בתור מחלקת קופסא שחורה, ולהוסיף מופע שלה עבור כל Route שהמשתמש ייצור. ז"א, ליצור מופע גרפי של חלון עבור כל Route שהמשתמש הוסיף למערכת דבר שיגרום ליצירת מספר חלונות כמספר הRoute-ים במערכת וזה לא יעיל בכלל במיוחד עבור מספר Route-ים גדול כך שהדרך היחידה לעשות ממשק זה אפשרי בכלל לשימוש חיצוני היא להסתיר ולהציג חלונות באופן דינמי כל הזמן.

חיסרון נוסף הוא שבמידה ויהיו בתוכנה תכונות שמאפשרות למשתמש למשל להשוות דרכים אז נצטרך או לעבוד עם הצגת קומבינציות של חלונות (ממש ממש ממש גרוע) או ליצור עוד שכבה גרפית להצגת קומבינציות של דרכים. (למה לא לעשות זאת מלכתחילה?)

הדרך העדיפה לניהול תוכנה כזו היא לבנות שכבה אחת (מחלקה) שתוכל להציג מידע באופן גרפי בחלון אחד על Route ספציפי כולל יכולת הוספת מקטעים אליו למשל אבל שגם תציג ScrollList של כל ה Routes שכבר הוספו למערכת. מחלקה זו תנהל במקביל יותר מRoute ותחזיק אותם ב-container/containers כלשהם. כאשר נתבקש להציג route מסוים אז נציג את המידע הספציפי שלו ב scrolLists שהתוכן שלהם יהיה המידע עבור Selected Route וניתן למשל להחזיק משתנה שישמור על איזה route עובדים.

לאותו ממשק ניתן יהיה להוסיף רשימה שנקרא לה למשל routes to compare וכפתור שיוסיף את selected Route לרשימה זו ובאפון דינמי לעדכן label שיציג למשל רק את הRoute ה"עדיף" (מה שלא תהיה הגדרת עדיף) בין ה Routes המוצגים ב- routes to compare.

בצורה זו מספר החלונות לא יגדל לינארית עם מספר הדרכים שיתווספו למערכת.

ג-)

אנחנו מעבירים:

1- Frame
2- this של המופע הנוכחי של ה- ב RouteFormatterGUI.

1: בסביבת מרובת מסכים הגדרת frame ה- owner מאפשר לתיבת הדיאלוג לדעת באיזה מסך היא צריכה להופיע. מקור:

https://docs.oracle.com/javase/7/docs/api/java/awt/Frame.html

2: ניהול הכפתורים (התעוררות באירוע) וה- scrollList של חלון (מידע על ה currently selected element) ב- GeoSegmentDialog נעשה בבנאי של GeoSegmentDialog כולל טעינת פונקציה ניהול אירוע הלחיצה וזה חייב להיות שם כי צריך לדעת מה היא הפעולה שתתבצע (על איזה כפתור נלחץ בדיאלוג) ועל איזה סגמנט מדובר. לכן אנו חייבים לתת לפונקציות אלה גישה למחלקת האב כדי שיוכלו לקרוא לפונקציית addSegment של מופע המחלקה GeoSegmentGUI הספציפי ש"הפעיל" את הדיאלוג.

ד-)

נחלק את השינויים לשינויים במחלקות שאחראיות על ההצגה הגרפית והשינויים במחלקת Route

- שינויים במחלקות ההצגה הגרפית:
צריכים להוסיף ברמת ה GUI אלמנטים שיאפשרו לממשק לתמוך בפעולה זו לכן את לחצן ה- Add GeoSegment נחליף ב- Manage Route למשל ובתוך הדיאלוג יהיה ניתן לבחור דרך ואז ללחוץ על כפתור חדש (סה"כ 3 כפתורים) שנקרא לו למשל Remove אשר לחיצה עליו תפעיל מתודה חדשה שנוסיף ל parent (RouteFormatterGUI) שאחראית על מחיקת מקטע במידה וזה מתאפשר** נקרא למתודה החדשה למשל RemoveSegment
אם יימחק המקטע האחרון ב- Route המשתנה Route יחזיק null

** מחיקת מקטע דרך היא חוקית אך ורק אם המקטע הוא האחרון או הראשון ב Route.
(אחרת Route מתפצל לשניים)

- למחלקה Route נוסיף מתודה חדשה בשם RemoveSegment שתיצור arrayList של geoSegments חדשה שתכיל את כל המקטעים שהיו חוץ מהמקטע שוסר.
מחלקה זו תשתמש אחר כך בבנאי שהוספנו כבר למלקת Route שיודע ליצור מופע של Route בהינתו רשימה של סיגמנטים.

שאלה 3

א-)

Abstraction function:

Every PolyTerm object in the terms list represents an added term in the polynomial.
for every term in the list:

- The coefficient of the added term = coeff field of the respective Polyterm
- The degree (p | x^p) of the added term = power field of the respective Polyterm

בעצם זהו מיפוי חד-חד ערכי (עד כדי סדר המחוברים) משדות המחלקה לפולינום.

ב-)

Representation Invariant:

* For every PolyTerm t in the terms List: t.power >= 0

* The powers of added terms are unique

* The polyTerms in the terms list are in an increased order of power such that the added term with the largest degree is at the end of the terms list.

זה מבטיח שני דברים:

- חזקות המחוברים מתאימות להגדרה של פולינום חוקי.

- המחובר בעל החזקה הגבוהה ביותר הוא המחובר שמיוצג בעזרת ה-term בסוף רשימת ה- terms, דבר שמבטיח שהמימוש של המתודה ()degree באמצעות גישה ישירה לאיבר האחרון ברשימה יניב תוצאה נכונה.

ג-)

מה שגורם למתודה coeff() להיות לא יעילה הוא שאין לה מידע על מיקומו של ה term שמייצג את המחובר בעל הדרגה p ולכן הוא יצטרך לחפש אותו בכל הרשימה. נוכל לפתור בעיה זו ע"י שמירת ה- terms במקומות ברשימה שהאינדקס שלהם יהיה החזקה של ה- term. ואז תחת ההנחה שפונקציית get של הממשק List אמורה להיות ממומשת בעזרת מערך דינמי מה שמאפשר גישה לאיבר מסוים ב- O(1) נוכל להשתמש בה כדי לגשת לterm המתאים ברשימה ע"י "שליפת" האיבר במקום ה- p כלומר מה שהמתודה coeff() תעשה הוא : return term.get(p).coeff שמתבצע ב- O(1)

Rep. Invariant מתאים :

Representation Invariant:

* For every PolyTerm t in the terms List: t.power >= 0

* the index of a PolyTerm in the terms list is equal to the power field of that same PolyTerm

(for each 0 <= i < terms.size: terms.get(i).power = I)

לשים לב שמימוש כזה מחייב שפונקציה שתוסיף מחוברים תהיה אחראית על תחזוקת הרשימה לשמירה על ה rep. Invariant. פתרון אפשרי הוא שתמלא את כל האינדקסים ברשימה בין שני מחוברים בterms עם חזקה מתאימה לאינדקס ומקדם אפס. (ולקבץ מחוברים אם מתווספים יותר מאחד בעלי אותה חזקה, מקרה שלא קשור לשאלה)