שפת ג'אווה ומבוא לפיתוח ווב

My Amazing Git - קיץ 2019

התרגיל מנוסח בלשון זכר, אך מכוון ל 2 המגדרים בצורה שווה

מרצה: **אביעד כהן** [aviadco@mta.ac.il](mailto:aviadco@mta.ac.il)

תקציר

גיט מי אפ סקוטי..

תוכן העניינים

[דרישות הקורס 4](#_Toc19399636)

[כללי 4](#_Toc19399637)

[איך להגיש תרגילים באיחור, ולהישאר בחיים 5](#_Toc19399638)

[הנחיות כלליות לכתיבת התרגיל 6](#_Toc19399639)

[תרגיל reflection – תרגיל עצמאי ביחידים (5%). מועד הגשה: 27.7.19 8](#_Toc19399640)

[כללי 8](#_Toc19399641)

[מבנה התרגיל 8](#_Toc19399642)

[ניקוד 8](#_Toc19399643)

[איך בודקים ? 9](#_Toc19399644)

[מה מגישים ? 9](#_Toc19399645)

[מקרה בדיקה לדוגמא 9](#_Toc19399646)

["M.A.Git" – מע' ניהול קבצים חכמה במיוחד ! 11](#_Toc19399647)

[מטרת התרגיל(ים) בקורס 11](#_Toc19399648)

[הקדמה הסטורית ומוטיבציה 11](#_Toc19399649)

[הוראות תכנון למע' קבצים חכמה – Git.A.M 12](#_Toc19399650)

[כללי 12](#_Toc19399651)

[SHA-1 12](#_Toc19399652)

[אובייקטים 13](#_Toc19399653)

[הקשרים בין האובייקטים 14](#_Toc19399654)

[שינויים לאורך הזמן 15](#_Toc19399655)

[ענפים (branch) 17](#_Toc19399656)

[שיתוף פעולה מרוחק 18](#_Toc19399657)

[הנחיות ספציפיות למימוש מערכת Git.M.A 18](#_Toc19399658)

[תרגיל 1 – מימושM.A.Git" " כאפליקציית Console (25%) - הגשה: 15.8.19 19](#_Toc19399659)

[פרטים יבשים 19](#_Toc19399660)

[דרישות 19](#_Toc19399661)

[חלוקה למודולים 22](#_Toc19399662)

[איך מתחילים ? (המלצה...) 23](#_Toc19399663)

[בונוסים 23](#_Toc19399664)

[סבבה, סיימתי. מה ואיך להגיש ? 24](#_Toc19399665)

[שאלות ותשובות 24](#_Toc19399666)

[תרגיל 2 – מימוש "M.A.Git" כאפליקציית JavaFX (35%) – הגשה: 21.9.19 25](#_Toc19399667)

[פרטים יבשים 25](#_Toc19399668)

[דרישות 25](#_Toc19399669)

[חלוקה למודולים 31](#_Toc19399670)

[איך מתחילים ? (המלצה...) 31](#_Toc19399671)

[בונוסים 32](#_Toc19399672)

[סבבה, סיימתי. מה ואיך להגיש ? 33](#_Toc19399673)

[שאלות ותשובות 33](#_Toc19399674)

[תרגיל 3 – מימוש "M.A.Git" כאפליקציית WEB (35%) – הגשה: 26.10.19 34](#_Toc19399675)

[פרטים יבשים 34](#_Toc19399676)

[דרישות 34](#_Toc19399677)

[חלוקה למודולים 39](#_Toc19399678)

[איך מתחילים ? 39](#_Toc19399679)

[בונוסים 40](#_Toc19399680)

[סבבה, סיימתי. מה ואיך להגיש ? 41](#_Toc19399681)

[שאלות ותשובות 41](#_Toc19399682)

[נספח א' – תיאור מצב של repository עם מספר קומיטים בה. 42](#_Toc19399683)

[נספח ב' – תיאור מצב של repository לפני/אחרי מיזוג בין 2 branch'ים 43](#_Toc19399684)

[נספח ג' – תיאור שיתוף פעולה עם repository אחר (clone, fetch, pull, push) 45](#_Toc19399685)

[נספח ד' – תיאור מבנה המערכת באמצעות קובץ XML 49](#_Toc19399686)

[נספח ה' – קובץ סכמה לוידוא קובץ ה XML 53](#_Toc19399687)

[נספח ו' – מידע על ספריות צד שמושיות לצד ג' 54](#_Toc19399688)

דרישות הקורס

## כללי

1. בקורס אין בחינה אך חובה להגיש תרגילים (סה"כ 4).
2. את מרבית התרגילים (3) ניתן להגיש בזוגות, אך לא בשלישיות / רביעיות / חמישיות או יותר

(כן, גם אם מדובר בשלישיה / רביעיה / חמישיה הצועדת יחדיו לאורך שנים מאז גיל הגן והגישה עד עכשיו את כל הפרוייקטים ביחד).

את תרגיל ה - [reflection](#_כללי) חובה להגיש ביחידים.

1. בעבודה משותפת על תרגיל יש להקפיד על מעורבות אקטיבית של כלל המגישים בכל חלקי התרגיל.
2. במידה והוגדר בונוס לתרגיל מסוים, ציון הבונוס יתווסף לציון התרגיל בלבד (ולא לציון הסופי של הקורס כולו).
3. התרגילים יוגשו דרך מערכת Mama. מוגדר רכיב 'מטלה' נפרד לכל תרגיל.
4. לפני שליחת התרגיל יש לבדוק שהוא עובד ומכיל את הקבצים המעודכנים ביותר, על מערכת "נקייה".

בצעו את סט הפעולות שאתם מצפים מן הבודק לבצע וודאו כי הכל מתנהל כראוי וכסדרו.

1. ניתן להחליף את השותפ/ה בכל תרגיל, ללא צורך באישור או הודעה למרצה.

הניקוד על כל תרגיל נזקף לזכות הסטודנטים שבצעו אותו בלבד.

1. יש להעלות את התרגיל רק עבור אחד מבני הזוג ולהוסיף את שם בת/בן הזוג ומספר תעודת הזהות שלה/ו גם באתר וגם בקובץ ה readme המצורף (פרטים בהמשך).

## איך להגיש תרגילים באיחור, ולהישאר בחיים

1. ניתן להגיש תרגיל עד שבוע איחור, כאשר עבור כל 24 שעות איחור – תורד נקודה אחת מציון התרגיל ; תרגיל שיוגש באיחור של יותר משבוע (ללא סיבה מוצדקת) – פשוט לא יבדק.
2. בתרגילים השונים ניתן לממש בונוסים (פרטים בהמשך).

המטרה של הבונוס היא לעזור לכם להעלות את הציון ולא להורידו !

רוצה לאמר: אל תגישו באיחור רק בשביל להספיק לפתח בונוס.

בונוס מפתחים **אם ורק אם** סיימתם את כל דרישות הבסיס להגשה, יש לכם עוד מספר ימים, וברצונכם לנסות ולהגדיל את הציון ע"י בונוס.

היות וכך, ולמען הסר כל ספק: **לא ייבדקו** הבונוסים עבור תרגילים שהוגשו באיחור (שאינו מוצדק).

בהתאם לכך, אני שומר לעצמי חירות רבה יותר בשינוי כזה או אחר של מי מסעיפי הבונוס, גם במהלך התרגיל עצמו.

1. עומס בלימודים, בעבודה, בחיים, בגלל הילדים או ההורים, שכנים וחברים (או בכל תחום אחר) אינו נחשב כסיבה לגיטימית להארכה.
2. במקרה של מחלה או מילואים יש להוסיף סריקה של אישור המחלה או המילואים להגשת התרגיל; בהגשת תרגיל באיחור בשל שירות מילואים יש לצרף להגשת התרגיל קובץ סרוק של טופס השחרור מהמילואים (ולא את צו הקריאה).
3. במקרה של בקשה להארכה (מכל סיבה שהיא, לרבות מילואים ומחלה) יש לפנות למרצה מראש על מנת לקבל אישור (במייל או פרונטלית בכיתה)
4. לאנשי הקבע – הישארות של שבת בבסיס אינה נחשבת כסיבה לגיטימית להארכה (מכיוון שזה חלק מהסדר העבודה בצה"ל); יציאה לאבט"ש כן נחשבת כמילואים ויש להגיש אישור ממפקד הבסיס.
5. סטודנט המגיש תרגיל באיחור של עד שבוע עם סיבה מוצדקת ישלח את הנימוקים לאיחור ביחד עם התרגיל.
6. ניתן לערער על ציון של תרגיל לכל היותר שבוע מיום פרסום המשוב והציון במע' המאמא.

כדי לערער יש לשלוח מייל למרצה בצירוף כל הסיבות והטענות שלכם.

## הנחיות כלליות לכתיבת התרגיל

* במהלך הקורס יוצגו דוגמאות והסברים מבוססים על כתיבה בסביבת הפיתוח (IDE) – Intelij IDEA.

אתם מוזמנים (ומעודדים בזאת) לפתח גם כן את התרגיל בסביבת העבודה intelij.

ניתן לקבל רישיון חינם לשימוש בגרסת ultimate, רק בשל היותכם סטודנטים במכללה (כבר שווה !)

יחד עם זאת, כל אחד רשאי לבחור לעבוד בסביבת העבודה הנוחה והמוכרת לו. כך או אחרת הגשת התרגיל אינה כוללת את סביבת הפיתוח אלא אך ורק הרצה ידנית מ cmd (כמו פעם...).

שימו לב: מבחינתכם, לבודק פשוט אין intelij (או כל ide אחר) ולכן זו אפילו לא אופציה.

חיסכו ממני (ומכם) את כתיבת המייל המבקש זאת.

* יש להגיש את התרגילים בתור קובץ zip/rar (לא 7Z !)

הקובץ יכיל:

1. כל הקבצים הרלבנטים להפעלת התרגיל (jar/war – פרטים בגוף התרגיל).
2. קובץ אצווה ( == batch) שיכיל את הפקודה שמריצה את התרגיל.
3. קובץ readme שיכיל את פרטי המגיש/ים, כמו גם הנחיות כלליות להרצה התרגיל וכל הנחות שלקחתם במהלך התרגיל ואתם סבורים שחשוב כי הבודק יכיר. דמיינו כי בכל שאלה/תקלה שיתקל בהן הבודק, יעמוד לרשותו רק קובץ ה readme שלכם. דאגו להבהיר ולהסביר את כל הדברים שיכולים להשתבש ו/או שבעטיים ייתכנו בעיות/שאלות/תהיות וכיוצב'.

כמו כן, כל הנחה שאתם מניחים בעצמכם לגבי אופן מימוש התרגיל (בין אם בלוגיקת התרגיל ובין אם בהנחה טכנולוגית) צריכה להיות רשומה בקובץ.

על קובץ הreadme להיות בפורמט word או pdf (**לא notepad !**). חי נפשי – אני אוריד על כך ניקוד.

* דווקא בגלל שאין זהו קורס שבו יכנסו לנבכי הקוד ויבדקו כל שורה ושורה, יש להקפיד ביתר שאת על קוד נקי, מסודר, קריא ויעיל. בפרט:
* הימנעו משכפול קוד
* פונקציות ארוכות מדי (יותר מגודל עמוד)
* בחירת שמות גרועים למחלקות, לפונקציות ולמשתנים
* הזחה (אינדנטציה) נכונה
* imports מיותרים
* יש להקפיד להשתמש ב-modifiers בצורה נבונה:
* מחלקה שלא אמורים לבנות אובייקטים שלה אמורה להיות מוגדרת כ-abstract
* קבועים יש לסמן כ- final
* משתנים של המחלקה רצוי (כמעט תמיד) להגדיר כ-private וכו'.
* יש להקפיד על מוסכמות בסגנון הכתיבה – שמות מחלקות יתחילו באות גדולה, שמות חבילות, משתנים ופונקציות באות קטנה, שמות קבועים יהיו מורכבים רק מאותיות גדולות וכו'.
* התמודדות עם קלט שאינו תקין (במקומות הרלבנטים) היא חלק בלתי נפרד מחווית המפתח (לטוב ולרע...).

יש לוודא קלט תקין מהמשתמש בכל שלב ולהחזיר הודאות שגיאה קריאות, אינפורמטיביות במידה והקלט אינו תקין. (למשל: לא להגיד שהקובץ לא תקין – אלא מה לא תקין בקובץ בצורה מפורטת...)

* יש לדאוג שבאף מקרה לא ייזרק Exception שלא טופל ושיגרום לתעופה של התוכנית; יש לטפל ב-Exceptions בנקודה שבה ניתן לעשות כן.
* הוראות שגויות שייגרמו לאפליקציה שלא לרוץ יורידו נקודות, ולכן רצוי מאוד שתנסו להתקין את האפליקציה בעצמכם לפי ההוראות שתכתבו.
* **זהו תרגיל מתגלגל. המטרה היא לבנות בסיס ראשוני בתרגיל הראשון, ולהמשיך ולהשתמש בו, ככל האפשר (ואפשר !) במהלך התרגילים הבאים. השקיעו חשיבה ותכנון בעיצוב הפתרון תוך מחשבה על איך מה שתעשו היום ישרת אתכם מחר. (זה כלל נכון לחיים, לא רק לתרגיל זה).**
* חלק מהעבודה בתרגילים היא קבלת החלטות בנושאים שאינם מפורטים במדויק. המטרה היא לתרגל את הנושאים המרכזיים הנלמדים בקורס, ולא לתפור מוצר לפי דרישות של לקוח. על כן, בכל מקום שלא מופיעה דרישה מדויקת – מוטל עליכם לבחור בדרך ההגיונית ביותר שנראית לכם ולציין את בחירתכם בקובץ ה Readme אשר מוגש עם התרגיל. אם יש ספק לגבי אופן פעולתכם אתם מעודדים לשאול האם הפתרון שאתם חושבים לתת לסוגיה מסוימת הוא קביל ולגיטימי (שאלות בפורום, מייל למרצה וכו)
* **ווידוא הגשת התרגיל טרם הגשתו:**
* **יש לוודא כי ההגשה שלכם רצה היטיב על מע' נקייה, באופן שבו גם הבודק יריץ אותה,** על מערכת נקייה וללא תוצרי לוואי אחרים של הפעלות קודמות שלכם.
  + **הבודק יבצע את הבדיקה על מע' windows 10. כל מי שמפתח על גבי mac/linux – זכותכם – אבל גם חובתכם לוודא כי אתם רצים היטיב על windows 10. למען הסר ספק, לא תתבצע בדיקה על מע' הפעלה אחרת. כמו כן לא תהיה התחשבות בתקלות שמקורם רק בשל עבודה על מע' הפעלה שונות (ולא שאמורות להיות תקלות כאלה..)**
  + **יש לוודא כי כל קבצי הבדיקה השונים שהועלו ל mama נטענים בהצלחה ע"י המע' שלכם טרם ההגשה. בדיקת הבודק תתחיל מבדיקה בסיסית המבוססת בצורה גסה על קבצים אלה. חבל ליפול Level 0 על שטות שיכולתם לעלות עליה בשנייה עוד בשלב הפיתוח.**
  + **הגשה חוזרת בשל תקלות level 0 תתחיל מראש מציון של 90. בלי שום יוצא מן הכלל. הקדימו תרופה למכה.**
* **בחלק מהתרגילים ניתנת אפשרות למימוש דרישות בונוס.**
  + **ישנם 2 סוגי בונוסים:**
    1. **בונוס בתוך טווח התרגיל - יכול להביא אתכם לכל היותר לציון 100, ולחפות במקרה והורדו לכם נקודות בשל תקלות.**
    2. **בונוס מחוץ לטווח התרגיל – יכול להעלות את ציונכם אף מעבר ל 100 (וכן, יש כפל מבצעים לטובת הלקוח).**
  + **בכל מקרה יש לבצע את הבונוס אם ורק אם סיימתם את כל דרישות הבסיס ההכרחיות לתרגיל.**
  + **חלק מהבונוסים בתרגילים השונים הם כאלה שנועדו "להקדים תרופה למכה" – מימוש דרישה בתרגיל n אשר בכל מקרה תגיע כדרישה חובה בתרגיל n+1.**

**הדבר נועד לעודד אתכם להוריד את העומס הצפוי בתרגיל n+1, מתוך הנחת יסוד שתרגיל n הוא קל יותר ומרווח יותר.**

**תכננו את עבודתכם בהתאם ושאפו "להקדים תרופה למכה", במידת האפשר. (וגם זה כלל חשוב לחיים, בלי קשר לתרגיל ולקורס).**

* + **פירוט הבונוסים, משקלם ונקודותיהם מפורט בגוף התרגיל הספציפי.**
  + **אם כבר מממשים בונוס, יש לממש את כולו, עפ"י דרישתו כדי לזכות במלוא הניקוד שהוא מקנה. בכל מקרה ההחלטה על ניקוד הבונוס היא בידי הבודק/מרצה בלבד (אל דאגה, המגמה היא להיות נדיבים ככל האפשר...)**
  + **כאמור, ולמען הסר כל ספק – ניקוד הבונוס מתווסף לניקוד התרגיל הספציפי שבו הוא מומש ולא לניקוד הסופי של הקורס. לא ניתן לקבל ציון סופי בקורס שהוא מעל ל 100 (גם אם בזכות הבונוסים הגעתם לציון כזה).**

תרגיל reflection – תרגיל עצמאי ביחידים (5%). מועד הגשה: 27.7.19

## כללי

**מועד הגשה: 27.7.19 צורת הגשה: ביחידים בלבד**

בתרגיל זה תזכו להכיר ולתרגל את מנגנון ה Reflection בג'אווה.

המצגת מכילה ידע בסיסי המהווה נק' פתיחה בלבד לנושא זה, ולכן כחלק מהתרגיל תדרשו גם ללמוד לבד נושאים נוספים הקשורים לעולם ה reflection שייתכן ואינם מכוסים (או אינם מכוסים כהלכה) במצגת.

שימו לב כי במצגת ישנם 3 נושאים. התרגיל הוא על הנושא הראשון בלבד של Reflection.

מהות התרגיל היא לכתוב investigator שיודע לקבל מופע (instance) של איזה שהוא אובייקט, ואז יודע "לחקור" אותו ולענות על מספר שאלות בהקשרו.

במהלך התרגיל תצטרכו להשתמש אך ורק ביכולת ה Reflection של ג'אווה כפי שמוסברות במצגת.

**אין להשתמש (ואין שום צורך) בשום ספריית צד שלישי כדי לבצע את המטלות השונות !**

צפי העבודה על התרגיל, בהינתן שקראתם והבנתם את המצגת, הוא 3-4 שעות.

צפי אורך הקוד שעליכם לכתוב הוא לא יותר מ 250 שורות (אני עשיתי זאת ב 160 שורות מרווחות היטיב..)

הבדיקה לתרגיל תבוצע בצורה אוטומטית, ע"י קוד שיטען את הקובץ שלכם ויריץ את סט הבדיקות שנגזרות ממנו, כלומר יקרא לכל השיטות המוגדרות ב interface שבחרתם, תוך השוואת הערך המוחזר מהקריאה אל ערך מצופה.

## מבנה התרגיל

התרגיל מכיל ממשק בשם Investigator, אותו עליכם לממש. מהות השיטות בממשק הוא לחקור instance של class אחר.

כל שיטה בממשק מתועדת היטיב מבחינת מה היא צריכה לעשות, מה הפרמטרים שהיא מקבלת ומה היא צריכה להחזיר.

חלק מהתרגיל כולל גם התמודדות עם התיעוד והבנה בעזרתו בלבד מה עליכם לעשות בכל שיטה ושיטה.

**הערות חשובות**:

1. שימו לב כי ה class שאתם מממשים חייב להכיל default public constructor !
2. שימו לב כי עליכם למקם את הממשק שקיבלתם (Investigator.java) בדיוק תחת ה package: **reflection.api** (ובהתאם לכך גם היררכיית הספריות כמובן).
3. במידה ויש מטודות הדורשות תפיסת exception, יש לתפוס אותו אולם אין להדפיסו ! (זה יוצר אי-סדר בהדפסות הפלט). במידה ומתרחשת תקלה או שתזרקו הלאה את ה exception ותוכנית הבדיקה תתמודד איתו בדרכה (הורדת ניקוד על הסעיף המדובר) או שתחזירו ערך כלשהוא כתוצאת המטודה (גם אם הוא לא נכון).
4. התוכנית מתחילה מקריאה לפונקיה load (אחת הפונקציות המתוארות בממשק). בפונקציה זו תקבלו את ה instance אותו עליכם לחקור. אפשר להניח כי זו הפונקציה הראשונה שתקרא, וכי היא תיקרא בדיוק פעם אחת.
5. כל מטודה בתרגיל עובדת בפני עצמה ופועלת על ה instance שקיבלתם במטודה load. אין להניח או להסתמך על סדר בקריאת המטודות (למען האמת בכל בדיקה הסדר הוא רנדומלי)

## ניקוד

התרגיל שווה עד 5 נקודות **מהציון הסופי** (!!)

(לא רע ל 3 - 4 שעות עבודה ו 200 שורות קוד...)

## 

## איך בודקים ?

קיבלתם את תוכנית הבדיקה עצמה, אותה אני הולך להפעיל גם בעצמי לטובת בדיקת התרגיל.

התוכנית מקבלת כפרמטר חיצוני את שם הקובץ המקומפל שלכם (.class) ובודקת אותו על מספר מקרי בדיקה.

בעותק התוכנית שקיבלתם כרגע, יש דוגמא למקרה בדיקה פשוט עליו תוכלו לנסות ולהתנסות ולוודא אם אתם בכיוון הנכון או אם לאו.

כדי להפעיל את תוכנית הבדיקה עליכם לנווט לספרייה בה נמצאת תוכנית הבדיקה להקליד ב CMD:

…\> RunTester <your .class file name>

אם הכל עובד כשורה, תקבלו את הפלט הבא:

Test name: Rectangle Class

Testing Rectangle Class basics...

Testing [ getTotalNumberOfMethods ]: expecting answer [6] and got [6]

…

Test Score: 100

## מה מגישים ?

עליכם להגיש קובץ zip, הכולל **בדיוק** 2 קבצים (ו 2 קבצים בלבד !):

1. קובץ התוכנית שלכם בלבד, בגרסתו המקומפלת (.class).
2. קובץ קוד המקור (למקרה של בעיות חמורות בלבד)

שם קובץ הזיפ צריך להכיל את שמכם ואת הת.ז. (אין להגיש קובץ readme בתרגיל זה...)

**מה לא מגישים ?**

1. את קובץ הממשק שקיבלתם
2. תיקיית פרויקט...
3. כל זבל אחר שבמקרה יושב לכם ליד הקוד...

## 

## מקרה בדיקה לדוגמא

במקרה זה מתואר האובייקט Rectangle היורש מאובייקט שנקרא Polygon.

מימוש זה כבר מוטמע בתוכנית הבדיקה שקיבלתם ומופע שלו יינתן כקלט לתוכנית שלכם.

המופע יאותחל בצורה הבאה:

**rectangle** = **new** Rectangle(4,6);

(המימוש הוא חלקי ולא תמיד הגיוני – אז אל תתפסו לקטנות...)

**public class** Polygon {  
  
 **private** Set<Point> **points**;  
  
 **public** Polygon() {  
 **points** = **new** HashSet<>();  
 }  
  
 **public int** getTotalPoints() {  
 **return points**.size();  
 }  
  
 **protected void** addPoint(**int** x, **int** y) {  
 **points**.add(**new** Point(x, y));  
 }  
}

**public class** Rectangle **extends** Polygon **implements** Comparable, Serializable {  
  
 **private int x**;  
 **private int y**;  
 **private final int SCALE** = 2;  
  
 **public static void** PRINT\_SOMETHING() {  
 System.***out***.println(**"this is a static method"**);  
 }  
  
 **public** Rectangle() {  
 **x** = -1;  
 **y** = -1;  
 }  
  
 **public** Rectangle(**int** x, **int** y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 updateParent();  
 }  
  
 **private void** updateParent() {  
 addPoint(0, 0);  
 addPoint(**x**, 0);  
 addPoint(0, **y**);  
 addPoint(**x**, **y**);  
 }  
  
 **public int** calcArea() {  
 **return x** \* **y**;  
 }  
  
 **public int** calcPerimeter() {  
 **return** twice(**x**) + twice(**y**);  
 }  
  
 **private int** twice(**int** num) {  
 **return** 2 \* num;  
 }  
  
 @Override  
 **public int** compareTo(Object o) {  
  
 **return this**.calcArea() - ((Rectangle)o).calcArea();  
 }  
}

"M.A.Git" – מע' ניהול קבצים חכמה במיוחד !

## מטרת התרגיל(ים) בקורס

חלק א' – מימוש מנוע מע' הקבצים המאפשר לתפעל את המע' כאפליקציית console עם פונקציונליות בסיסית.

חלק ב' – הרחבת מע' הקבצים במספר פיצ'rים ופיתוח קליינט גרפי במימוש כאפליקציית Java FX (Desktop).

חלק ג' – מימוש אפליקציית web המציעה אפשרות לשתף פעולה עם מע' קבצים חכמות של משתמשים אחרים

## הקדמה הסטורית ומוטיבציה

מע' ההפעלה מעניקה למשתמש מספר שירותים, כאשר אחד מהם הוא מע' הקבצים. מע' הקבצים היא מע' שבה מתנהלים כל קבצי המשתמש, המחולקים (בדר"כ) לספריות המאפשרות גישה מהירה לפריט המידע המבוקש (== הקובץ).

מע' הקבצים המסופקת ע"י מע' ההפעלה מאפשרת תמיד לקבל אך ורק את המצב הנוכחי, הנתון של כל הקבצים במע' ;

היא **אינה** מסוגלת לשמור את המצב שהיה נתון בו כל קובץ וקובץ לאורך הזמן ; היא אינה יכולה לשמור את המצב של כל מע' הקבצים לפני השינויים על פני ציר הזמן: ברגע שבוצע שינוי בקובץ והוא נשמר – מבחינת מע' הקבצים – זהו העותק הנוכחי של הקובץ ; ברגע ששם ספרייה שונה – לא ניתן לדעת מה היה שמה הקודם ; ברגע שקובץ נמחק/הועבר מספרייה אחת לשנייה – לא ניתן לדעת מה היה מצב הדברים טרם השינוי (שאולי נתגלה בנתיים כשינוי הרסני שגרם לדברים במע' להפסיק לעבוד...)

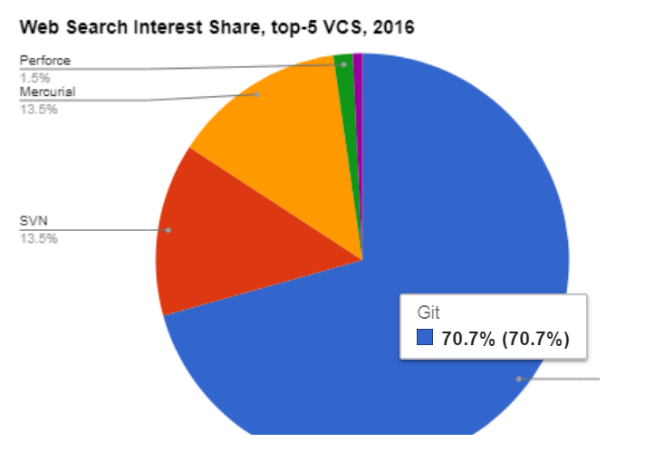
בעולם פיתוח התוכנה, מפתחים רבים משנים את הקוד של המע' אותה הם מפתחים. הקוד מפוזר על פני קבצים רבים ומאורגן, בדר"כ, בהיררכיה ענפה של ספריות ותתי ספריות. עבור חברה המפתחת מוצר – הצורך להיות מסוגל לחזור לפיסת הקוד טרם ביצוע שינוי כזה או אחר הוא לא פחות מקריטי, זאת כדי להיות מסוגלת לאתר באגים ושגיאות, לשחזר גרסאות ללקוח וכן הלאה.

על הרקע הזה הלכו ופותחו ברבות השנים מע' לניהול גרסאות קוד (**V**ersion **C**ontrol **S**ystem - **VCS**) – פיסות תוכנה המסוגלות "לסמלץ" מע' קבצים חכמה שבה יש יכולת לשמור לא רק את הפוזיציה הנוכחית של הקבצים השונים אם כי גם את ההסטוריה המלאה שלהם מתחילת הדרך. בשנים האחרונות, מע' אחת חולשת על כל שוק התוכנה ומהווה את הבסיס המרכזי לפיתוח תוכנות באשר הן. המערכת הזו נקראת **GIT**.

[GIT](https://en.wikipedia.org/wiki/Git) הומצאה ע"י [Linus Torvalds](https://en.wikipedia.org/wiki/Linus_Torvalds), אשר היה (ועודנו) מי שיצר את גרעין מע' ההפעלה הפופולרית linux.

באפריל 2005 ליינוס חיפש מע' ניהול גרסאות קוד שתהיה טובה יותר מן הנמצא בשוק באותו הזמן. משלא מצא את מבוקשו, הגה את גיט (יחד עם מפתחים נוספים) במשך כ... שבוע אחד בלבד ! (ובכן, את החלק מהותי לפחות).

החל משנת 2009 (בערך...) GIT הלכה ותפסה תאוצה והפכה להיות מע' ה VCS הנפוצה ביותר בקרב מפתחים, חברות וארגוני תוכנה, גדולים כקטנים !



## הוראות תכנון למע' קבצים חכמה – Git.A.M

בתרגיל זה נממש מע' ניהול קבצים חכמה משלנו, העובדת בצורה דומה מאוד (מאוד !) לאופן שבו GIT עובדת.

(אל דאגה ! לא נייצר את GIT במלואה, אם כי רק את החלקים העיקריים שלה ומכאן יוכל כל אחד ואחת לקחת את זה לאיזה כיוון שיחפץ, ובעיקר לכתוב בקו"ח שהוא מימש את GIT...)

למע' שלנו נקרא M.A.Git – **M**y **A**mazing **GIT** (שזה קרוב מאוד למילה magic, ולא בכדי...)

## כללי

M.A.Git מנהלת repositories (מחסנים) של מידע. בתוך ה repository יישמרו כל המידעים האפשריים ש M.A.Git צריכה כדי לאפשר את היכולת לחזור אחורה בזמן ולהציג את מצב הקבצים המנוהלים על ידה לאורך ההסטוריה.

כל פרוייקט ב M.A.Git מנוהל בספרייה משלו על גבי מע' הקבצים הרגילה. בראשית הספרייה הזו תשב ספריית הניהול של M.A.Git והיא תיקרא .magit (שימו לב כי הנקודה '.' היא חלק משם הספרייה !) וכל הקבצים והספריות המנוהלים מספרייה ראשית זו ואלך (פנימה) הם הם הקבצים המנוהלים ע"י M.A.Git, ועבורם (ורק עבורם) היא תוכל לספק את השירותים העודפים שלה.

דוגמא:

אם המשתמש פתח repository חדש בספרייה c:\test, אזי מע' הקבצים תראה כך:

C:\test

.magit

<all files, folders and subfolders to maintain by this repository>

המע' תאפשר כמובן לנהל כמה M.A.Git repositories במקביל, המנוהלים במקומות שונים על גבי מע' הקבצים.

M.A.Git תאפשר לטעון ולאתחל repository מקובץ XML אשר מתאר את כל פרטי המידע הנמצאים בה לאורך ההסטוריה כולה. פרטי קובץ ה XML המלא מובאים בנספח [זה](#xmlAppendix)

## SHA-1

SHA-1 הוא שם של אלגוריתם מעולם הקריפטוגרפיה האחראי על הצפנה של תוכן.

SHA-1 פועל כך שעבור כל סוג של Input (מחרוזת תווים סופית) הוא מפיק בדיוק 40 תווים אקסה-דצימלים (Hex) המייצגים בצורה חד חד ערכית את התוכן שהוכנס לו. מספיק שינוי בודד, קטן, איזוטרי (אפילו להוסיף "רווח") במקום כלשהוא בקלט, כדי ש SHA-1 יפיק עבורו מחרוזת אקסה שונה לחלוטין (ועדיין, בדיוק 40 תווים).

SHA-1 הוא אלגוריתם חד כיווני: לא ניתן לחזור ממחרוזת ההקסה (40 תווים) ולהפיק חזרה את הקלט שהוכנס לאלגוריתם (!)

M.A.Git תשתמש ב SHA-1 כאמצעי לאיתור והכוונה אל פיסות האובייקטים שהיא מנהלת.

**אל דאגה**, אינכם צריכים לממש בעצמכם את האלגוריתם הנ"ל.

תקבלו ספרייה צד שלישי שעושה עבורכם את העבודה. היא תצפה לקבל קלט ותחזיר לכם את מחרוזת ה SHA-1 המצופה - בדיוק 40 תווי אקסה...

## אובייקטים

M.A.Git מנהלת בתוכה 3 סוגי "אובייקטים":

1. Blob (file) – זוהי יחידת המידע הבסיסית ביותר המכילה **אך ורק** את התוכן של כל קובץ וקובץ.

במע' ינוהלו הרבה יחידות של blob'ים.

עבור כל כל קובץ במע', בכל שלב, התוכן שלו **בלבד** יוכנס כ input ל SHA-1 ויישמר עבורו מחרוזת הפלט (40 תווי אקסה).

1. Folder (tree) – מייצגת ספרייה במע'.

ספרייה יכולה להכיל אסופה של קבצים ו/או אסופה של ספריות נוספות (תתי ספריות).

הספרייה מכילה רק את המידע על הקבצים ולא את תוכנם הממשי (המוחזק כבר ע"י blob).

המידע המוחזק בספרייה עבור כל קובץ הוא:

* 1. שם
  2. SHA-1
  3. סוג (ספרייה / קובץ)
  4. מי שינה אותו אחרון
  5. תאריך שינוי אחרון

האופן שבו אפשר לנהל "ספרייה" הוא פשוט להתייחס אליה כאל 'קובץ' המכיל את המידע של הדברים הנמצאים בתוכה.

כך למשל אם יש לנו ספרייה שנקראת foo המכילה שני קבצים goo.java ו moo.txt כמו גם תת ספרייה בשם sub, אפשר יהיה לממש את הספרייה כקובץ טקסטואלי שבו יהיה כתובים המידעים הנ"ל:

(כאן, התו ',' נבחר להיות התו המפריד בין פרטי המידע השונים של כל קובץ – אתם יכולים לבחור כל תו אחר)

*goo.java, ABCD…, file, Ruven, 1.1.2012-13:23:11:444*

*moo.txt, 345F…, file, Shauly, 3.5.2015-14:24:12:334*

*sub, 2DF23S…, folder, Irena, 7.4.2004-11:11:11:111*

שימו לב כי גם הספרייה לא מכילה מידע על עצמה ! (שם, מיקום וכו') ונחווית כ"קובץ" ש M.A.Git מנהלת.

כמו כן לא ניתן לתחזק במע' ספרייה ריקה. אם בספרייה אין שום תוכן (קובץ או ספרייה אחרת) – היא פשוט לא קיימת.

עבור כל "קובץ" המתאר ספרייה יווצר גם כן SHA-1 המתאר את תוכנו ותישמר עבורה מחרוזת הפלט (40 תווי אקסה)

1. Commit – מייצג פוזיציה מסויימת בזמן T של מצב ה repository כפי שהייתה ברגע יצירת ה commit.

כל קומיט מכיל את המידע הבא:

* 1. SHA-1 של הספריה הראשית של הפרוייקט
  2. SHA-1 של הקומיט שקדם לקומיט זה (אב ישיר)
  3. SHA-1 של קומיט נוסף שקדם לקומיט זה (אב ישיר נוסף, רלבנטי רק למקרה של Merge, ראו בהמשך)
  4. מסר/הודעה על מהות הקומיט
  5. מתי נוצר הקומיט
  6. מי יצר את הקומיט

גם כאן המע' מתייחסת לקומיט כאל "קובץ" הכתובים בו המידעים הנ"ל ושומרת אותו בקרבה ;

גם כאן תוכן הקובץ של הקומיט יעבור SHA-1 ומחרוזת הפלט תישמר עבורו.

## הקשרים בין האובייקטים

M.A.Git מנהלת מאחורי הקלעים מעין טבלת מיפויים גדולה בין SHA-1 (40 תווי אקסה) ל"אובייקטים" השונים המוצבעים על ידיהם.

היות והתוכן של כל אובייקט (המתואר למעשה כ"קובץ") יכול להיות גדול – כל קובץ כזה יעבור כיווץ באמצעות zip ויישמר בתוך ספריית ה .magit, תחת ספרייה ייעודית שנקראת objects . השם של כל קובץ יהיה שם ה SHA-1 שלו. (להלן "המיפוי")

היות וקומיט מכיל הפנייה ל SHA-1 המייצג את הספרייה הראשית של המע' ;

היות וכל ספריה שכזו מכילה הפניות SHA-1'ים לתתי הספריות ו/או הקבצים המופיעים בתוכה ;

היות ובסוף נגיע לספריות קצה המכילות הפניות SHA-1 לתוכנם של הקבצים עצמם ;

נוכל לטעון כי:

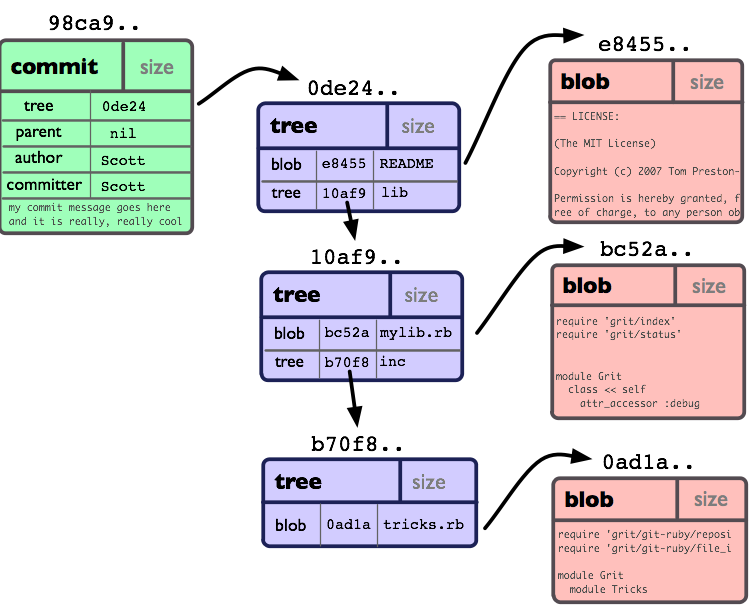
**קומיט מאפשר לנו להציג את תוכנה של מע' הקבצים המנוהלת ע"י M.A.Git בכל רגע נתון !**

הסבר:

בהינתן הכוונה לקומיט מסוים, אפשר באמצעות המיפויים שאנו שומרים לייצר את כל הקבצים של המע' כפי שאלו התקיימו ברגע יצירת הקומיט:

נתחיל מהקומיט המכווין ל SHA-1 של הספרייה הראשית. ניגש לטבלת המיפויים ונגיע באמצעות ה SHA-1 למיקום של האובייקט המכווץ. נפתח אותו ונגלה את תוכנה של הספרייה הראשית.

נעבור על כל אחת מהשורות שרשומות בה ובאמצעות טבלת המיפויים וה SHA-1 נגיע אל האובייקט המכווץ של כל פריט ופריט בתוכה. נפתח גם אותם ונמשיך להתנהל כך בצורה רקורסיבית עד שנסיים את הסריקה של כל העץ הנפרס מהקומיט.



**(הערה**: התמונה מהווה המחשה בלבד למבני הנתונים והקשרים בינהם ואינה כוללת את כל המידע שאתם נדרשים לאכסן בכל אחד מהרכיבים מטעמי מקום.**)**

## שינויים לאורך הזמן

נניח כי ברגע נתון לקחנו את אחד הקבצים (foo.txt) ושינינו את תוכנו. בעקבות כך ישתנה גם ה SHA-1 של אותו הקובץ. נחשב SHA-1 זה ונוסיף את התוכן המכווץ של הקובץ לספריה objects. היות וה SHA-1 של הקובץ ששינינו השתנה, צריך לעדכן אותו ב"קובץ הספרייה" המכילה את הקובץ ששינינו. היות וכך גם התוכן של "קובץ הספרייה" ישתנה ולכן צריך לחשב עבורו SHA-1 חדש. גם כאן נשמור את תוכנו (המכווץ) של קובץ הספרייה החדש ונוסיף למיפוי עם ה SHA-1 החדש. שימו לב כי קבצים אחרים הנמצאים באותה הספרייה אשר אינם השתנו – היו ונותרו כשם שהיו קודם: "קובץ הספרייה" החדש ממשיך להצביע אליהם (באמצעות ה SHA-1) בדיוק כמו קודם לכן.

נמשיך בתהליך הזה "מעלה" לספרייה שהכילה את הספרייה שהכילה את הקובץ וכן הלאה עד שנגיע לספרייה הראשית.

עכשיו אנו מוכנים לייצר קומיט חדש במע', אשר יפנה ל SHA-1 החדש שנוצר עבור הספרייה הראשית. ניתן לקומיט זה הודעה המתארת את מהותו (תיקון באג בקובץ foo.txt) ונוסיף פרטי מידע נוספים.

נרשום ב"קובץ הקומיט" את ה SHA-1 של הקומיט הנוכחי (שהיווה בסיס אשר יחסית אליו בוצע השינוי הנ"ל) ונייצר את הקומיט החדש במע'. באופן זה הקומיט החדש במע' בעצם מייצג את מצב מע' הקבצים כולה (המנוהלת ע"י M.A.Git) כפי שהייתה ברגע יצירת קומיט זה (כלומר השינוי בקובץ foo.txt וכל שאר הקבצים שלא השתנו מהפעם הקודמת).

היות וקומיט זה מצביע לקומיט שקדם לו ;

היות וראינו כי מכל קומיט ניתן בצורה רקורסיבית "לפרוס" את מע' הקבצים ברגע יצירת הקומיט -

אזי נוכל למעשה לאמר כי **שרשרת הקומיטים,** בצירוף המיפוי בין SHA-1 למיקומם של אובייקטים **מהווה את היכולת לחזור לכל נקודה בזמן ולהציג כל קובץ/ספרייה עם התוכן שלה בכל רגע נתון (!!!)**

שינוי ב M.A.Git נחשב כ:

* קובץ קיים התעדכן (יחסית לתוכנו הקודם).
* קובץ חדש נוצר
* קובץ קיים נמחק

**שימו לב**: **אין צורך** לתמוך במצב של שינוי שם קובץ (מבלי שתכולתו תשתנה) או במצב של העתקת קבצים מספרייה לספרייה (כך שתוכנם ושמם יישאר זהה)

שימו לב כי כל עדכון של מידע במע' (קובץ ו/או ספרייה) רק **מתווסף** לספריית ה objects. לעולם לא נמחוק מידע המופיע בספרייה זו (המיכלה את המיפוי הנדרש בין sha1 לפריט המידע המבוקש) וכך למעשה תישמר כל ההסטוריה של כל הקבצים/ספריות האפשריים במע'.

M.A.Git מגדירה מספר "אזורי עבודה"

* working copy (WC) – זהו האזור על פני מע' הקבצים עצמה (בתחומי repository כלשהוא) שבו נמצאים בכל רגע נתון כל הקבצים המוצבעים ע"י הקומיט הנוכחי בתוספת השינויים שבוצעו במי מהם ע"י המשתמש.
* commited – זהו אזור "לוגי" ומהווה בעצם את ה repository עצמו ובו מאוחסנים כל הקומיטים במע' (כל מה שמצוי בנבכי הספרייה .magit)

.magit

objects

branches

M.A.Git management (hidden) folder

C:\test

M.A.Git

Repsitory

**WC**

**(Root Folder)**

Folders & files

**דוגמא להמחשה:**

בחרנו קומיט מסויים ופרסנו אותו ב WC. כרגע WC מציג את כל הקבצים כפי שהיו בעת קומיט זה. למצב זה נקרא מצב "נקי".

נניח כי שינינו את תוכנו של קובץ f. כרגע ב WC יש את כל קבצי הקומיט במצב המקורי, כאשר קובץ f למעשה שונה ממצבו המקורי ומצוי עם השינוי שהרגע חוללנו בו. למצב זה נקרא "שינויים פתוחים". במצב זה (חלק מ-) הקבצים שב WC שונים מן הקבצים המוצבעים ע"י הקומיט הנוכחי.

תהליך יצירת הקומיט החדש כולל מספר שלבים:

(שלבים אלה יופרדו לכמה פקודות עצמאיות שונות כפי שיפורט בהמשך בגופו של תרגיל 1):

1. זיהוי כל הקבצים שהשתנו ב WCיחסית לקומיט הנוכחי (חישוב SHA-1 לכל אחד מהם והשוואתו ל SHA-1 של הקובץ הנוכחי, במידה וקיים). כל השינויים הללו יוצגו למשתמש בדמות 3 רשימות:
   1. קבצים שהתעדכנו (נתיב מלא לכל קובץ)
   2. קבצים חדשים שנוצרו (נתיב מלא לכל קובץ)
   3. קבצים שנמחקו (נתיב מלא לכל קובץ)
2. המשתמש יבצע קומיט אשר ייכלול את כל השינויים שנצברו ב WC.

מייד לאחר ביצוע ה commit, המע' חוזרת להיות במצב "נקי".

**שימו לב**:

* שינוי של שם המשתמש שבוחן את המע' אינו מהווה "שינוי" בשום אופן. אך ורק שינוי בתוכנם או קיומם של קבצים.
* בהתאם למימוש שלכם, יש להימנע ממצב שבו תוכן של ספרייה לא השתנה כלל, אולם מה שהשתנה הוא רק הסדר של האברים בתוך הקובץ המתאר את הספרייה. כדי להימנע מכך יש לשמור את תוכן קובץ הספרייה ממוין בצורה לקסיקוגרפית וכך כל תוכן של ספרייה שמכיל את אותם הפרטים בעלי אותם השמות ייראה וייכתב תמיד אותו הדבר.

## ענפים (branch)

M.A.Git תאפשר למשתמשיה לפצל את עבודתם ולעבוד "במקביל" על כמה נושאים שונים.

יכולת זו תתאפשר באמצעות רכיב הנקרא Branch.

Branch (ענף) הוא לא יותר ממצביע ל SHA-1 של קומיט מסויים ; לכל branch יש שם ייחודי משלו (אולם אין להשתמש ב'רווח' בשם ה branch).

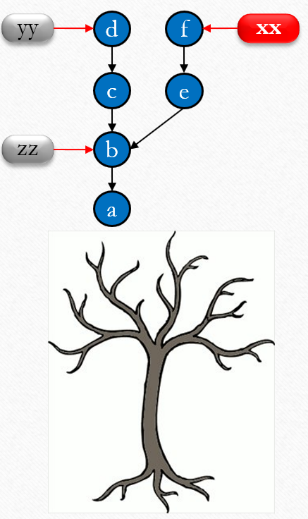
המע' יכולה לנהל כמה branch'ים כאלה (המצביעים לקומיטים שונים במע'), אולם בכל רגע נתון יש בדיוק branch אחד המסומן אקטיבי. Branch זה יישמר בכל רגע נתון בקובץ מיוחד שנקרא HEAD. התוכן של הקובץ הוא בסה"כ שמו של ה branch האקטיבי.

הקומיט המוצבע ע"י ה branch האקטיבי הוא הקומיט שקבציו "ייפרסו" במע'.

כאשר מבצעים קומיט במע' – ה branch האקטיבי יעודכן להצביע עליו. כך שלמעשה ה branch האקטיבי "מתקדם" לאורך ציר הזמן יחד עם הקומיטים השונים שנוצרים.

המשתמש יוכל לייצר branch'ים חדשים במע' תוך הפנייתם לקומיט הנדרש עליו הם צריכים להצביע.

המשתמש יוכל להפעיל פקודה המחליפה את ה branch האקטיבי. בעשותו כך, M.A.Git תמחק את תוכן הספרייה המנוהלת ו"תפרוס" מחדש את מע' הקבצים כפי שהיא נגזרת מהקומיט המוצבע ע"י ה branch האקטיבי. כמו כן בפעולה זו יעודכן גם קובץ ה HEAD.

מרגע שבמע' מתקיימים מספר branch'ים – הרי שיש לנו את היכולת להציג לא רק את ההסטוריה של הפרוייקט לאורך ציר הזמן, אם כי את היכולת להציג התפצלויות (ענפים) מקבילות של מע' הקבצים שלנו (!) המשתמש יוכל לנוע בין ההתפצלויות הללו באמצעות החלפת ה branch האקטיבי. כמו כן ניתן יהיה גם "למזג" (to merge) 2 התפצלויות ובכך לאחד את כל השינויים שבוצעו במע' קבצים אחת עם מע' קבצים אחרת **(!!!)**.

M.A.Git תגדיר לפחות branch אחד שייקרא master והוא יהיה ה branch האקטיבי הראשוני.

כל המידע על ה branch'ים המתקיימים במע' ינוהל בספרייה בשם branches בתוך ספריית M.A.Git. כל branch יהיה קובץ בפני עצמו כאשר שם הקובץ הוא שם ה branch. התוכן של הקובץ הוא..... בדיוק 40 תווי אקסה של SHA-1 של קומיט המוצבע ע"י ה branch הזה... (amazing !). קובץ ה HEAD יישמר גם הוא בתיקייה branches.

## שיתוף פעולה מרוחק

M.A.Git תאפשר למשתמשים לייבא מידע ונתונים מ repository אחר כמו גם לעדכן במידע חדש respository מרוחק.

ההגדרה "מרוחק" מתכוונת לכל repository אחר שיש אליו גישה, בין אם במע' הקבצים הלוקלית של המשתמש הקיים ובין אם באמצעות גישה למע' קבצים הנמצאת על מחשב מרוחק.

M.A.Git תאפשר למשתמש לייבא את האובייקטים השונים של repository מרוחק אל repository קיים, כולל כל הקבצים, הספריות, הקומיטים וה branch'ים השונים. כמו כן אפשר יהיה לייצר אובייקטים ב repository קיים ולעדכן repository מרוחק איתם (ובכלל זה קבצים, ספריות, קומיטים ו branch'ים).

M.A.Git תאפשר למשתמש לאתחל repository חדש על סמך repository קיים (clone) ; למשוך ולהתעדכן בכל המידעים החדשים שנוצרו ב remote repository (fetch) ; למשוך מידע ועדכונים (קומיטים) שהתרחשו ב branch מסויים (pull) ; לדחוף מידע חדש באמצעות branch (קומיטים) ל remote repository (push).

פירוט של אופן פעולה זו מתואר בפרוטרוט בנספח ג', ובתרגיל 2 סעיף 10.

## הנחיות ספציפיות למימוש מערכת Git.M.A

1. **המטרה היא לבנות מנוע מערכת גנרי, כזה שידע לקבל את הפרטים לגבי מבנה ה repository מתוך קובץ נתונים בפורמט XML (עבודה עם XML'ים תילמד במהלך הקורס כמובן).**

**מנוע המערכת הגנרי ילך וישתכלל מתרגיל לתרגיל, בהתאם לפיצ'רים השונים. כך תוכלו לחוות מהלך שלם של מוצר החל מרעיון קטן במימוש בסיסי וכלה במנוע משחק שמניע אפליקציית ווב שלמה.**

1. מטעמי נוחות, המע' תכלול תמיכה בקבצי טקסט בלבד.
2. **שמות הספריות של M.A.Git הינם בגדר חובה:**
   1. **ספריה ניהול ראשית של M.A.Git: .magit (שימו לב, התו נקודה '.' הוא חלק משם הספרייה)**
   2. **ספריית אחסון כל האובייקטים המכווצים: .magit\objects**
   3. **ספריית אחסון כל ה branch'ים: .magit\branches**

**להוציא את שמות אלו, אתם רשאים להוסיף איזה מידע נוסף שתרצו לתוך ספריית .magit אשר יסייע לכם בניהול הספרייה.**

1. **שמות האובייקטים המכווצים יהיו ה SHA-1 הרלבנטי לקובץ זה. (אין צורך לשים סיומת)**
2. **בתרגיל תצטרכו לשמור מידע של תאריך וזמן שבו קרו דברים.**

**הפורמט של התאריך והזמן הוא**

**dd.mm.yyyy-hh:mm:ss:sss**

**מפריד '-' בין היום לזמן ; מפריד '.' בין הרכיבים השונים של היום/זמן ; כל ספרה מופיעה בצורה מלאה (למשל 08 ולא 8)**

**יש ללמוד לבד איך לייצר את חותמת הזמן בפורמט זה. שימו לב כי הפורמט מעלה הוא בגדר הסבר כללי ; זהו לא הפורמט לשימוש copy-paste...**

**(רמז עבה מאוד: חפשו מידע על מחלקה בשם SimpleDateFormat)**

1. **בתרגיל תדרשו הרבה פעמים להשוות בין 2 קומיטים, או בין מצבם של 2 repositories שונים. יש לחשוב על מבנה נתונים מתאים שיאפשר לכם מעבר על המידעים השונים תוך השוואתם למידע מקביל וגזירת השינויים. התהליך רקורסיבי באופן טיבעי ומומלץ מאוד לחשוב בכיוון זה.**
2. **Repository שנוצר (כך או אחרת) לא יימחק ויש לאפשר לדעת לחזור אליו מאוחר יותר ולהמשיך את העבודה איתו מאותו מקום שעצרנו**

תרגיל 1 – מימושM.A.Git" " כאפליקציית Console (25%) - הגשה: 15.8.19

## פרטים יבשים

צפי תחילת עבודה: **19.7.19** תאריך הגשה: **15.8.19**

צפי זמן לביצוע: **3 שבועות** ציון אפשרי מקסימלי: **130**

משקל התרגיל: **25%** קושי: **סביר**

**מטרת התרגיל העיקרית**

1. הקמת מנוע המע' הבסיסי
2. יצירת ממשק console לתפעול המערכת

## דרישות

1. בתרגיל זה תממשו את מנוע המערכת, אשר יודע לקבל נתונים על מבנה ה repository ולאפשר למשתמש לתפעל אותה ועוד repositories במקביל לה. המערכת תכיל את הפיצ'רים המינימלים הבסיסים ביותר הנדררשים לתפעולה: blob, folder, commit, branch.
2. במסגרת תרגיל זה **לא תדרשו** לתמוך ב remote collaboration ואפשר להניח כי כל branch הוא Is-remote=false ו tracking=false. כמו כן לא יופיע האלמנט remote בקובץ המע'. כמו כן אמנם תייצרו branch'ים, אולם **לא תדרשו** לתמוך בתהליך של מיזוג (Merge) בין 2 branch'ים.
3. פרטיה הטכנים של המערכת יינתנו מקובץ XML (כמפורט [בנספח](#xmlAppendix)).

עליכם לוודא בדיקת קלט לקובץ ה XML ולוודא כי הקובץ מכיל מידע תקין ואמין.

(מובטח כי הקובץ יהיה תקין schema-wise אבל לא בהכרח תקין application-wise...)

במידה והקובץ מתגלה כתקול, יש להודיע על כך בצורה מסודרת ונאותה למשתמש. יש לפרט מהי סיבת התקלה (ולא להסתפק במשפט סתום כמו: "הקובץ תקול...").

בפרט יש לוודא את הפרטים הבאים:

* 1. הקובץ אכן קיים, והוא מסוג XML (די לבדוק לשם כך כי הוא נגמר בסיומת xml)
  2. בתוך כל אחד מסוגי האלמנטים ("blobs", "folders", "commits") אין 2 אלמנטים בעלי id זהה.
  3. Folders מצביעים ל blobs ע"י id. יש לוודא כי כל הצבעה כזו אכן קיימת (כלומר אין הפנייה מ folder ל id של blob שלא מוגדר)
  4. Folders מצביעים ל folders אחרים ע"י id. יש לוודא כי כל הצבעה כזו אכן קיימת (כלומר אין הפנייה מ folder ל id של folder אחר שאינו לא מוגדר)
  5. יש לוודא כי אין הפניה של folder ל id של עצמו
  6. יש לוודא כי כל הפנייה מ commit ל folder היא רק ל id של folder שקיים ומוגדר תחת folders
  7. יש לוודא כי כל הפנייה מ commit ל folder היא רק ל folder המסומן כ is-root="true"
  8. יש לוודא כי כל branch מצביע ל id של commit שמוגדר
  9. יש לוודא כי head מצביע ל name של branch שמוגדר.

1. יש לוודא תקינות קלט כחלק מכל אינטרקציה עם המשתמש בכל מקום שבו זה רלבנטי:

אם אתם מצפים לקבל מספר – לא לקרוס כי הכניסו לכם בטעות (או בכוונה) טקסט וכו'.

בכל מקרה של תקלה יש להיות מאוד ברורים במסר שמעבירים חזרה למשתמש: מה קרה ? מה הייתה מהות התקלה ? היכן שזה רלבנטי, איך לתקנה וכו'.

חישבו איך להיות ידידותיים למשתמש ולעולם אל תניחו כי מי שמשתמש באפליקציה שלכם הוא מתכנת בעצמו או מישהו שמגיע מהתחום ו"מכיר" איך דברים עובדים לבד... (זה הזמן לחשוב על...)

1. **אין צורך** להשתמש בצבעים שונים במהלך תרגיל זה בעת ההדפסה ל console.

כמו כן **אין צורך** לנקות את המסך בין פקודה לפקודה.

יתרה מזאת, ישנו צפי רב (ניסיון מהסמסטרים הקודמים) כי ניסיון לעשות כן תוך שימוש בספריות צד שלישי קורס אצל הבודק,

מעוות את כל תצוגת המסך וגורם לחוסר יכולת לבדוק את ההגשה.

גם אם בדקתם את זה אצלכם וזה עבד.

גם אם בדקתם במחשב של השכנה וזה עבד.

1. בכל רגע נתון המע' תדע לנהל ולעבוד על repository אחד בלבד (בין אחד שקיים מבעוד מועד ובין אם אחד חדש שנטען זה הרגע באמצעות קובץ XML).

כמו כן בכל רגע נתון המע' מכירה את שם המשתמש שמבצע את כל הפעולות. כשהמע' רק עולה (ואף אחד עוד לא הגדיר או טען שם משתמש בצורה מפורשת) שם המשתמש הדיפולטי הוא Administrator. כמו כן, ולמען הסר כל ספק, כשהמע' רק עולה – אין שום repository שמנוהל על ידה.

1. עליכם לכתוב ממשק משתמש בתצורת console.

ממשק המשתמש יכיל סט סופי של פקודות שדרכן ניתן יהיה להפעיל את המשחק.

**שימו לב**: ישנן פקודות שאין הגיון לבצע אותן אם לא קדמו להם פקודות אחרות. במידה וזה קורה יש להציג הודעת שגיאה רלבנטית למשתמש ולאפשר את המשך מהלך המשחק.

רשימת הפקודות שיש לתמוך בהן:

1. עדכון שם משתמש

הפקודה תאפשר למשתמש לעדכן את שם המשתמש האקטיבי שבגינו מבוצעות כל הפעולות. שם המשתמש הוא string פשוט שיכול להכיל כל תו שרוצים.

1. קריאת קובץ פרטי repository

פקודה זו טוענת את פרטי המערכת למנוע מתוך קובץ נתונים בפורמט XML.

קבצי דוגמא מתאימים הועלו מבעוד מועד לאתר הקורס ואתם מוזמנים להורידם ולבחון אותם בהתאם.

הקובץ יהיה קובץ XML שפרטיו מובאים [בנספח](#xmlAppendix) לתרגיל זה.

(שימו לב כי ייתכן וקובץ ה xml מכיל פרטים נוספים שאין להם חשיבות לתרגיל זה אלא רק לתרגילים הבאים וזאת כדי לפשט את התהליך בו אתם משתמשים בקובץ – עדיין זהו קובץ תקין ואתם צריכים לדעת להתמודד איתו).

יש לוודא תקינות הנתונים המופיעים בקובץ כפי שהוגדרו בסעיף 3.

במידה והקובץ לא תקין יש לדווח זאת למשתמש בצורה ברורה כך שניתן יהיה להבין מה לא תקין בקובץ.

אין לקרוס על exception במידה וקובץ מתגלה כאינו תקין ; יש לאפשר למע' להמשיך לפעול במצב זה.

(כחלק מבדיקת התרגיל יטענו למערכת קבצים שאינם חוקיים כדי לבדוק מהי התגובה).

במידה והקובץ נמצא תקין – יש לבדוק האם כבר יש repository ב location המוגדר בקובץ זה. בדיקה אם location הוא M.A.Git respository יכולה להיעשות ע"י בדיקה אם יש בתוכו את ספריית magit.

**אם לא** – אזי המע' מייצרת את ספריית ה repository ומאתחלת בתוכה את כל המידע כפי שנגזר מתיאור קובץ ה xml.

אם יש שם תוכן אחר שאינו repository – אפשר פשוט להודיע על כך למשתמש ולא לעשות כלום (כלומר הפעולה בטלה)

**אם כן** – יש להתריע על כך למשתמש ולשאול אותו איך הוא רוצה להמשיך:

1. למחוק את ה repository שכבר קיים ולייצר אחד חדש כפי שעולה מהגדרת קובץ זה.

במקרה זה המע' מתחילה לבנות את מע' הקבצים הנגזרת מתיאור repository זה על פי כל הכללים שהוצגו בתיאור הכללי של התרגיל.

1. להמשיך להישאר עם ה repository שכבר קיים ולהציג את המידע הנגזר ממנו (כלומר "לוותר" על טעינת המידע מהקובץ)

כך או אחרת, יש לסיים פקודה זו במסר למשתמש לגבי המתקיים במע' (האם נטען ונוצר בהצלחה או נכשל וכן הלאה)

ניתן להפעיל פקודה זו כמה פעמים רצוף. כאשר בכל רגע נתון המע' "עובדת" על ה repository התקין האחרון שנטען בהצלחה. שימו לב כי במעבר בין repository אחד למשנהו אין למחוק אף אחת מספריית ה repositories !

1. החלף repository

פקודה זו מאפשרת לעבור ולהחליף את ה repository האקטיבי. יש לבקש מהמשתמש נתיב מלא (full path) לספרייה בה שוכנת repository היעד. יש לוודא כי אכן ספריית היעד קיימת, וכי היא אכן ספרייה של M.A.Git repository (מספיק ע"י בדיקה שיש בתוכה את הספריה .magit)

לאחר ביצוע פקודה זו ה repository העדכני הוא זה שהמשתמש ביקש לעבור אליו וכל המע' יודעת להציג את נתוני ה repository החדש.

1. הצגת כל הקבצים של הקומיט הנוכחי בליווי מידע היסטורי

פקודה זו תציג את כל הקבצים והספריות במע', כפי שהם נגזרים מהקומיט המוצבע ע"י ה HEAD branch.

על כל קובץ/ספרייה יש להציג:

* שם (נתיב מלא)
* סוג (Folder או Blob)
* SHA-1
* מי ביצע את השינוי האחרון שבו.
* מתי חל העדכון האחרון שחל בו

1. הצגת מצב ה working copy (show status)

פקודה זו מציגה את מצב הנתונים במע':

ראשית יש להציג את שם ה repository ואת המיקום שלה (היות וה repository יכול להשתנות בעקבות טעינת קובץ חדש או מעבר ל repository אחר חשוב מאוד תמיד לדעת על איזה respository אנו מסתכלים)

שנית יש להציג את שם המשתמש האקטיבי כרגע.

שלישית (והכי חשוב) יש להציג את השינויים הפתוחים המתקיימים ב working copy (דורש מעבר על כל הקבצים ב repository הנוכחי והשוואתם למצב מע' הקבצים כפי שהיא נגזרת ונפרסת מהקומיט הנוכחי המוצבע ע"י ה head branch.)

שימו לב כי יש גם להציג קבצים שנמחקו יחסית לקומיט הנוכחי ; קבצים חדשים שנוצרו יחסית לקומיט הנוכחי ; קבצים קיימים שעודכנו יחסית לקומיט הנוכחי.

אם קובץ לא השתנה בכלל יחסית לקומיט הנוכחי – **אין להציגו** בפירוט הפלט של פקודה זו.

1. ביצוע commit

בפקודה זו נבצע קומיט חדש במע'. הקומיט חל על הקבצים הממתינים באזור ה WC.

יש לבקש מהמשתמש הודעה המתארת את אופי הקומיט. מאחורי הקלעים המע' תוסיף לבד את שם המשתמש לקומיט, מועד ביצוע הקומיט, הכוונה לקומיט קודם, קידום ה Head branch להצביע על הקומיט החדש וכן הלאה.

בסיום פקודה זו אזור ה WC מכיל את אותם הקבצים שקיימים בו – אולם עכשיו הוא כבר "נקי", ואין "שינויים פתוחים" במע'.

1. הצגת כל ה branch'ים במע'

פקודה זו תציג מידע על כל ה branch'ים המתקיימים במע'.

עבור כל branch יש להציג את המידעים הבאים:

1. שם ה branch
2. Sha-1 של הקומיט המוצבע ע"י ה branch
3. הודעה של הקומיט המוצבע ע"י ה branch

בנוסף יש להציג חיווי מסויים (לבחירתכם) ליד ה Head branch כדי שקל יהיה לזהות להבין מיהו...

1. יצירת branch חדש

פקודה זו מייצרת branch חדש במע'. יש לבקש מהמשתמש שם של branch ולוודא כי אין כבר branch תחת השם הזה.

ה branch יווצר תוך שהוא מצביע על הקומיט המוצבע ע"י ה head branch הנוכחי. לא מבצעים checkout ל branch החדש שנוצר. כלומר לא משנים את HEAD !

1. מחיקת branch

פקודה זו תמחק branch במע'. יש לבקש מהמשתמש שם של branch למחיקה.

שימו לב:

1. כי אין לאפשר למשתמש למחוק את ה head branch (מה שאומר שבמע' תמיד יישאר לכל הפחות branch אחד ויחיד..)
2. ייתכן שמחיקת branch תגרום לכך שיהיו קומיטים ש"ילכו לאיבוד" – כלומר לא יהיה אף אחד שמצביע עליהם. קומיטים אלה אכן "נעלמים" מן המערכת ואין צורך לשמור, להציג או להתחשב בהם במהלך הפקודות הבאות בתור
3. בחירת head branch (checkout)

פקודה זו מבצעת החלפה של ה Head branch. כפועל יוצא יש "לפרוס" ב working copy את מבנה מע' הקבצים כפי שהוא נגזר מן הקומיט המוצבע ע"י ה branch החדש.

במידה ויש שינויים "פתוחים" במע' (כלומר קבצים שהשתנו בכל אופן וצורה, ב working copy, יחסית לקומיט הנוכחי) יש להתריע על כך למשתמש. יהיה עליו לבחור אם לעצור וללכת לקמט את הקבצים הללו (פקודה 5 ו 6) או להמשיך ובכך למעשה להתעלם מהקבצים הללו וכל השינויים שבהם יבוטלו לאלתר ללא יכולת לשחזרם !

1. הצגת הסטוריה של ה active branch

פקודה זו מאפשרת לראות את כל הסטוריית הקומיטים של ה active branch, החל מהקומיט הנוכחי (המוצבע ע"י ה head branch) ועד לראשית הזמנים (!)

על כל קומיט יש לתת את המידע הבא:

1. SHA-1
2. הודעת הקומיט
3. מתי נוצר
4. ע"י מי נוצר
5. יציאה מהמע'

פקודה זו יוצאת מהמערכת.

## חלוקה למודולים

בתרגיל זה **חובה** לייצר (לפחות) 2 מודולים (מהם תפיקו בהמשך 2 jar'ים):

* + 1. ממשק ה ui, המציג את התפריטים השונים, אחראי על קליטת קלט מהמשתמש והחזרת הפלט למשתמש.

שימו לב זהו המודול "האקטיבי", המניע את כל המע'. הוא זה האחראי על הפעלת ותפעול מנוע המערכת.

* + 1. מנוע המערכת – האחראי על קבלת הפקודות (ממודול ה ui), ביצועם והחזרת פלטים מתאימים.

שימו לב שמודול זה "פסיבי", ורק מגיב לבקשות ולפקודות המתקבלות ממקורות בלתי ידועים לו (בתרגיל זה מודול #1). בתרגילים הבאים מקורות נוספים יפנו אליו לקבלת מידע וחשוב מאוד להקפיד על כך **שמודול זה אינו מכיר/מודע למי פונה אליו.**

## איך מתחילים ? (המלצה...)

אני ממליץ להתחיל דווקא במנגנון שיאפשר לכם ליצור ולהתחיל repository בצורה ריקה בספרייה מסויימת (ולא דרך קובץ XML). (כן, בונוס מס' 1. הוא לא באמת בונוס – סתם מתנה ממני אליכם – זה אמור להיות פשוט ביותר ויהווה נק' פתיחה נקייה יותר למע'))

בצורה זו תאלצו ליצור את האובייקטים הראשיים (repository, branch) שיהוו את הבסיס להמשך הדרך. תייצרו את ספריית האם (magit.) ותוודאו כי כל התהליך עובר חלק מה UI אל המע' וחזרה עם הפלט הרלבנטי.

אחרי כן מומלץ להמשיך לתהליך של יצירת הקומיט הראשון. זהו התהליך הראשי שיגרום לכם להכיר את מרבית התרחישים (סריקת WC, סריקת קומיט קודם) והדרכים לעבוד איתם (רקורסיבית בעיקר). שימו לב כי הקומיט הראשון שונה משאר הקומיטים מאחר ואין קומיטים לפניו, אין למי להשוות את WC אליו וכן הלאה. אולם בעבודה על קומיט זה כבר תייצרו קבצים ו sha1'ים, תכווצו בזיפ, תעבדו ותבחנו את מבני הנתונים שישרתו אתכם להמשך הדרך המתארים את הקומיט/ספרייה/בלוב וכו'.

מכאן כדאי להמשיך להתמודדות עם בניית קומיט אמיתי (כזה שיש לו אבא) וכאן האתגר העקרי הוא ההבנה ויצירת הדלתא של WC מול ה"קומיט הנוכחי".

השלב הבא זה להיות מסוגל לייצר branch חדש (בקטנה, כולה קובץ עם פויינטר...) ואז להתמודד עם התהליך של checkout (מעבר ל branch אחר) שבו אתם "פורסים" קומיט קיים לתוך WC.

אחרי שהצלחתם את זה – שאר הדברים ייזרמו הרבה יותר קל... (כל ההתעסקות עם יצירת/מחיקת branch'ים, החלפת שם משתמש, מעבר ל repository אחר וכן הלאה).

ממליץ לשמור את העבודה עם קובץ ה XML קצת יותר להמשך הדרך, אחרי שכבר יש לכם מע' חיה, קיימת ועובדת.

## בונוסים

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | סוג | מהות | למה שווה לי ? | כמה שווה לי ? |
| 1 | הקדמת תרופה למכה | ממשו כבר עכשיו את הפעולה של "אתחול" repository חדש מתרגיל 2, כפי שהוגדר בסעיף 5.ג.  יש להוסיף פעולה נוספת לתפריט המשתמש שבו הוא נדרש לתת שם ומיקום ל repository חדש ואתם תאתחלו שם repository חדש ותעברו לשרת אותו משם. | כי זה מה-זה קל שחבל... | 5 נקודות  (ועד ל 100) |
| 2 | הקדמת תרופה למכה | ממשו כבר עכשיו את הפעולה של "איפוס" branch ל SHA-1 מסוים. בפעולה זו יש לאפשר למשתמש לשנות את הקומיט המוצבע ע"י ה head branch ל SHA-1 אחר. אם יש שינויים פתוחים במערכת – יש להתריע על כך למשתמש. אם בוחר להמשיך – אזי כל השינויים האלה נמחקים ללא יכולת לשחזרם. בגמר הפעולה המערכת תציג את המצב של הקומיט החדש המוצבע ע"י ה branch. (פקודה מס' 4 בתפריט) | לא כזה מסובך כשחושבים על זה... | 10 נקודות  (ועד ל 100) |
| 3 | הגדלת ראש נאה | ביצוע checkout מייד במעמד יצירת branch חדש.  במהלך יצירת branch חדש תשאלו את המשתמש אם הוא חפץ גם לעבור אליו אחרי כן (checkout) או לאו. אם יענה כן תעברו אליו אחרי יצירתו.  שימו לב כי אם יש שינויים פתוחים – תצטרכו לייצר את ה branch אבל רק להתריע בפני המשתמש כי יש שינויים פתוחים ולכן לא בוצע ה checkout | מתנה משמיים.  לי זה עולה יותר. | 5 נקודות  **(מעל ל 100)** |
| 4 | הגדלת ראש על שם נווה פרג'ון | תמיכה ביכולת של שינוי שם/הזזת קבצים/שכפול קבצים ושמות ספריות  אפשרו למשתמש לבצע שינוי שם, שכפול קבצים לספריות אחרות שכפול ענפי ספריות שלמים תחת ספריות אחרות ועדיין לזהות נכונה את השינויים הנכונים הנגזרים מכך | יש ניואנס מסוים שצריך לעלות עליו – ואז זה הופך להיות פינטס | 10 נקודות  **(מעל ל 100)** |
| 5 | הגדלת ראש מגניבה | ייצוא תוכן ה repository לקובץ XML:  פקודה זו מאפשרת למשתמש להכניס שם של קובץ XML (full path) ולייצר את כל ה repository למבנה של קובץ XML הזהה לזה שקיבלתם במעמד הטעינה של המע'.  שימו לב כי יש לייצא לקובץ את כל המידע ב repository, כולל כל ההסטוריה, כל ה branch'ים וכן הלאה. המטרה תהיה לאפשר בהמשך לטעון מחדש את קובץ ה XML שיצרתם וכך "לשכפל" repositories.. | קוד טכני בסופו של דבר אבל עובר דרך עבודה מאומצת עם מבני הנתונים של ג'אווה | 15 נקודות  **(מעל ל 100)** |

## סבבה, סיימתי. מה ואיך להגיש ?

יש להגיש קובץ zip המכיל:

1. 2 jar'ים (לפחות) שהם כל הקוד שלכם, בצירוף קובץ אצווה (batch) שהפעלתו תריץ את התוכנית

(כלומר תבצע java -jar <class name> וכו').

1. קבצי קוד המקור של הפרויקט שלכם.
2. קובץ ההגשה יכיל גם קובץ **readme** שיכיל הסבר על המערכת, בחירותיכם השונות במקומות שבהם היו לכם בחירה, כמו גם כל דבר נוסף העולה על דעתכם שחשוב שהבודקת תדע.
3. יש לכלול בקובץ ה **readme** גם תיעוד והסבר כללי (וממצה) של המחלקות העיקריות ותפקידם.
4. יש לכלול בקובץ ה **readme** גם פירוט של המגישים שם, ת.ז. ואי מייל זמין ורלבנטי (!!) – במידה ויהיה צורך ליצור קשר.
5. במידה ומימשתם את מי מהבונוסים לעיל, ציינו זאת בקובץ ה readme כדי שהבודק ידע לבדוק בהתאם.

בונוס שימומש אבל לא יתועד – לא ייבדק !

## שאלות ותשובות

|  |  |  |  |
| --- | --- | --- | --- |
| # | תאריך | שאלה | תשובה |
| 1 | 20.7.19 | האם במעמד יצירת branch גם צריך לעשות לו checkout (כלומר לשנות את ה – Head ?) | לא. אין לעשות זאת. ה branch החדש רק נוצר על הקומיט של ה head הנוכחי. אם המשתמש ירצה יוכל לעשות checkout בעצמו כשיחפץ בכך. |
| 2 | 22.7.19 | איזה מידע יש להציג עבור כל branch בפקודה המציגה את פרטי ה branch'ים ? | מפורט כעת בסעיף 7.7 |
| 3 | 25.7.19 | מה צריך לכלול repository חדש שמייצרים במהלך בונוס מס' 1 ? | ההגדרה המלאה נמצאת בתרגיל 2 בסעיף 5.ג. |
| 4 | 31.7.19 | כמה קומיטים אחורה צריך לזכור כל קומיט ? | בתרגיל זה בדיוק אחד אחורה – האב הישיר של הקומיט הנוכחי. יוצא דופן הוא הקומיט הראשון במע' שהוא היחיד שאין לו אבא – מאחר והוא הראשון  (עדכנתי את ההגדרה המתאימה לכך בסוף עמוד 13) |
| 5 | 31.7.19 | האם אפשר "ללכלך" את WC גם כשהמע' לא עובדת או כשהיא "רואה" repository אחר ? | בוודאי שכן !  WC הוא אסופת הקבצים היושבת כרגע ב repository ואותה תשווה לקומיט הנוכחי **אך ורק** כשתדרשו לכך ע"י המשתמש בפקודה 5 או 6. כל אחד יכול לעדכן את הקבצים שם בכל מועד שיחפץ. |
| 6 | 1.8.19 | איך להגיב אם בטעינת repository מ xml כבר יש ב location קבצים שאינם repository חוקי ? | מפורט כעת בסעיף 7.2, סוף עמוד 20. |

תרגיל 2 – מימוש "M.A.Git" כאפליקציית JavaFX (35%) – הגשה: 21.9.19

## פרטים יבשים

צפי תחילת עבודה: **11.8.19** תאריך הגשה: **21.9.19**

צפי זמן לביצוע: 5 **שבועות** ציון אפשרי מקסימלי: **130**

משקל התרגיל: **35%** קושי: **מאתגר**

**מטרות התרגיל העיקרית**

1. מימוש האפליקציה כ desktop application באמצעות Java FX
2. הרחבת מנוע המערכת לתמיכה בפיצ'רים נוספים (merge, collaboration)

## דרישות

* 1. בתרגיל זה תממשו את המערכת כ desktop application באמצעות ארכיטקטורת JavaFX כפי שנלמד בכיתה.
  2. בתרגיל זה אין יותר סדרה של "תפריטים" כפי שהיה בתרגיל 1. כל ממשק המשתמש הוא ממשק גרפי בלבד, המבוסס על JavaFX. כל האינטראקציה עם המשתמש מתרחשת דרך פעולותיו על גבי הפקדים השונים שתציגו לו (כפתורים, תיבות טקסט וכו'). מצב המערכת כולה מוצג בכל רגע נתון, ובפרט ה branch הנוכחי, הקבצים שלו, עץ הסטוריית הקומיטים וכו'
  3. המע' תציג ברגע נתון את מצבה המלא של ה repository האקטיבי. יש להציג תמיד את השם והמיקום של ה repository המוצג ע"י המערכת בכל רגע.

הצגת המידע כוללת את כל הקומיטים, branch'ים, יכולת לבדוק תוכן של קומיט מסוים וכן הלאה. בנוסף על אלה, המע' תאפשר לבצע את מגוון הפעולות שמומשו בתרגיל 1 ובנוסף מספר פעולות חדשות. יש להוסיף כפתור/תפריט/תפריט הקשר (context menu) שיאפשר לבצע את כל אחת מהפעולות השונות (מפורטות בהמשך). תוכלו לחשוב ולהגות איזה סדר שתרצו לטובת הדבר ומלבד שיהיה סביר, הגיוני ונוח לשימוש (תפריטים, קבוצות כפתורים, תפריטי הקשר וכן הלאה)

* 1. בתרגיל זה תתמכו ביכולת נוספת של שיתוף פעולה בין repositories שונים. יכולת זה באה לידי ביטוי גם בקובץ ה XML. (פירוט נוסף בסעיף 10). היות וכך, במעמד טעינת קובץ ה XML יש לתת את הדעת לבדיקות הקלט הבאות (בנוסף לכל הבדיקות שכבר מימשתם בתרגיל 1):

1. אם מופיע האלמנט MagitRemoteReference יש לוודא כי אכן יש repository ב location המוצבע על ידו.
2. אם branch מוגדר כ tracking=true יש לוודא כי האלמנט tracking-after מצביע לשם של branch אחר שחייב להיות מוגדר כ is-remote=true.
   1. פעולות כלליות ו/או הקשורות ל repository:
3. **הגדרת שם המשתמש** – כמו בתרגיל 1 יש לאפשר להגדיר את שם המשתמש הנוכחי שמבצע את הפעולות. כל עוד לא הוגדר שם משתמש – שם ברירת המחדל הוא Administrator.
4. **טעינת repository חדש באמצעות קובץ xml.**

כשם שהיה קודם, יש לבקש מהמשתמש נתיב מלא (באמצעות file chooser) ולטעון משם את ה repository. במידה ולא תקין יש להעביר הודעה מתאימה למשתמש. הטעינה תתבצע באמצעות task באמצעות thread אחר ותעביר חזרה ל UI חיווים נדרשים.

1. **אתחול repository חדש.**

יש לאפשר למשתמש לבחור שם של ספרייה, לוודא כי אינה קיימת ולייצר בה repository חדש. ב repository חדש אין קומיטים ושום קבצים בכלל. כן יש בה branch אחד ויחיד בשם master שהוא גם ה HEAD. בשלב ראשוני branch זה לא מצביע לשום קומיט במע'. רק בביצוע הקומיט הראשון הוא יקבל את ההצבעה אליו.

**שימו לב** כי זהו מקרה הקצה היחיד שבו ל branch לא תהיה הצבעה לקומיט.

1. **החלפת reposisotry לאחר.**

יש לבקש מהמשתמש נתיב לספריית repository אחרת ולעבור אליה, כמובן אחרי שבודקים כי הספרייה היא ספריית של magit repository. מייד אחרי שעוברים אליה יש לתאר את כל המע' בהתאם למצב ה repository.

* 1. פעולות על קבצים/קומיטים:

1. **הצגת תוכן של קומיט**

הציגו כפתור המאפשר לראות את תוכנו של קומיט מסוים: כל הקבצים/ספריות המתקיימים בתוכו. אפשרו לבחור את אחד הקבצים ואז לראות את תוכן הקובץ כפי שהוא היה בעת ביצוע הקומיט הנ"ל.

1. **הצגת מצב השינויים הפתוחים ב WC**

הציגו כפתור שמאפשר לראות את כל השינויים הפתוחים במע' כפי שהוגדר בתרגיל 1 - קבצים שהשתנו (כך או אחרת) יחסית לקומיט הנוכחי.

1. **ביצוע commit**

הציגו כפתור המאפשר לבצע קומיט. כמו בתרגיל 1 יש לבקש מהמשתמש להכניס הודעה מתאימה המתארת את מהות הקומיט.

* 1. פעולות על branch'ים:

1. **הצגת רשימת ה branch'ים**

הציגו כפתור המאפשר לראות את רשימת כל ה branch'ים הפעילים, כולל חיווי ברור לגבי זהותו של ה head branch

1. **יצירת branch חדש**

הציגו כפתור המאפשר יצירת branch חדש המצביע על הקומיט הנבחר. יש לבקש מהמשתמש שם עבור ה branch ולוודא כי הוא ייחודי. כלומר לאפשר לייצר branch לא רק על בסיס ה HEAD הראשוני אלא גם ע"י מתן SHA-1 שעליו רוצים שה branch יצביע.

1. **מחיקת branch קיים**

הציגו כפתור המאפשר מחיקת branch. יש לבקש מהמשתמש את שם ה branch המיועד למחיקה ולוודא את קיומו. אין לאפשר את מחיקת ה HEAD ! במידה וזה מה שמבקש המשתמש יש להתריע על כך בפניו ולא לאפשר את הפעולה.

1. **החלפת head branch (checkout)**

הציגו כפתור המאפשר החלפת ה head branch ל branch אחר. יש לקבל את שם branch היעד, ולוודא כי הוא קיים. פעולה זו יכולה להתבצע רק אם ה WC נקי משינויים. אם זה לא המצב יש לעצור ולהתריע על כך בפני המשתמש.

1. **אתחול מיקום של branch (reset)**

הציגו כפתור המאפשר את אתחול ה head branch להצביע על קומיט אחר, תוך התעלמות ומחיקת כל "השינויים הפתוחים" שיש כרגע ב working copy. יש לבקש מהמשתמש SHA-1 של קומיט היעד להצבעה. במידה ויש שינויים פתוחים במע' בזמן ביצוע פקודה זו – יש להתריע זאת בפני המשתמש ולאפשר לו לבחור אם להמשיך או לבטל את הפעולה.

* 1. רכיב מרכזי בתרגיל זה הוא תצוגה גרפית של עץ הקומיטים וה branch'ים המוגדרים ב repository. באמצעות רכיב גרפי זה המשתמש יוכל לנווט ביתר קלות (בהשוואה ל console...) בין הקומיטים השונים, לקבל הבנה של מצב ה branch'ים וההצבעות, לבצע פעולות וכו'.

את גרף הקומיטים יש לצייר כעץ אנכי, הממויין על פי ציר הזמן, כך שהקומיטים החדשים יותר צריכים להופיע ראשונים והקומיטים הישנים יותר צריכים להופיע אחרונים. העץ מצוייר מהעדכני ביותר (למעלה, בראש הגרף) ועד לישן ביותר (למטה, בתחתית הגרף). שורש העץ מצוייר בתחתית.

יש להציג את כל ה branch'ים השונים על גבי העץ, במיקום המתאים שלהם מבחינת הקומיטים.

המשתמש יוכל לבחור כל קומיט בעץ, ולקבל מידע לגביו:

* SHA-1
* הודעת קומיט
* מי יצר אותו
* מתי נוצר
* SHA-1 של קומיט(ים) קודמ(ים) (הורים ישירים בלבד)
* פרטים על המידע שהשתנה בקומיט זה (דלתא מול קומיט קודם)

יש להציג זאת כרשימה של שינויים, כפי שמציגים את פירוט ה WC יחסית לקומיט הנוכחי:

* שם הקובץ שהשתנה
* האם עודכן/נוצר/נמחק

שימו לב כי אם מדובר בקומיט שהוא תוצאה של merge, הרי שאז יש לו 2 אבות, והדלתא צריכה להיות מחושבת יחסית ל 2 האבות. משאיר זאת לשיקולכם כיצד להציג את המידע למשתמש (אפשר לאפשר לו לבחור דלתא מול מי הוא רוצה לראות, או להראות תמיד את הדלתאות מול 2 האבות בצירוף חיווי שמציג כל קובץ מאיזה דלתא הוא מגיע וכו').

כך או אחרת תעדו זאת כמובן בקובץ ה readme.

* 1. בתרגיל זה תתמכו באפשרות למזג 2 branch'ים שונים.

כחלק מתהליך המיזוג, תתבצע הכלאה של מצב ה respoitory כפי שהוא משתקף מ 2 ה branch'ים השונים. התוצאה של המיזוג הוא קומיט חדש המכיל את סך השינויים והאיחודים שבוצעו בין 2 ה branch'ים

בתהליך המיזוג יש לבחור branch נוסף אשר ימוזג אל תוך ה head branch.

את התהליך אפשר להתחיל **אך ורק** כשאין שינויים פתוחים במערכת. (יש להתריע אם זה לא המצב בפני המשתמש).

תוצאת המיזוג תבוא לידי ביטוי בדמות קבצים שמשתנים על גבי ה WC, וייחשבו כשינויים פתוחים, להם פשוט נבצע קומיט (כמעט) רגיל כפי שבוצע בתרגיל הקודם...

שימו לב: מדובר בפעולה אחת המורכבת ממספר שלבים. היא מתחילה במיזוג עצמו (המייצר סדרה של שינויים פתוחים ב WC), עוברת דרך פתרון קונפליקטים אינטראקטיבי במידה וקיימים (יפורט בהמשך) ונגמרת בפעולת קומיט חדש. לא ניתן ולא צריך לעצור את הפעולה באמצע. לא צריך לתמוך במצב שבו הפקודה התבצעה רק בחלקה. או שהיא התבצעה במלואה (ונגמרה בקומיט חדש) או שהיא בוטלה לחלוטין (ואז לא קרה כלום)

לטובת תהליך זה נגדיר את ה Head branch כ'בסיס' ואת ה branch הנוסף כ'מטרה/יעד'.

(כינוי נוסף (הנהוג בגיט) להבדלה בין 2 ה branch'ים השותפים לתהליך ה Merge הוא **'ours'** עבור ה Head branch ו **'theirs'** עבור branch המטרה.)

כדי לבצע את תהליך המיזוג תדרשו למצוא את האב הקדמון המשותף ל 2 ה branch'ים, ובאמצעותו תקבלו הבנה טובה יותר של מהם השינויים שבוצעו במי משני ה branch'ים.

חישבו על תהליך (רקורסיבי) שבו תשוו מצבו של קובץ אחר קובץ בין הבסיס ליעד:

* כל קובץ חדש שקיים **רק** באחד משני הצדדים – יישמר כשינוי פתוח.
* כל קובץ שנמחק **בשני** הצדדים (והיה קיים באב הקדמון) – ייחשב כקונפליקט.
* כל קובץ שנמחק בצד אחד בלבד:
  + אם מצבו בצד השני הוא **כשם שהיה באב הקדמון** – יימחק וייחשב כשינוי פתוח.
  + אם הוא **שונה** בהשוואה למצבו באב הקדמון – הוא יישמר לרשימת הקונפליקטים.
* כל קובץ שעודכן **בצד אחד בלבד** (יחסית לאב הקדמון) – העדכון שלו יישמר כשינוי פתוח.
* כל קובץ שעודכן **בשני הצדדים** (יחסית לאב הקדמון) – יישמר לרשימת הקונפליקטים.
* וכן הלאה וכן הלאה (ויש עוד הרבה הלאה והלאה...)

מהם קונפליקטים ?

כדי לשמור על אחידות ובהירות בהגדרה, במקרים בהם הקובץ שונה (כך או אחרת: עדכון, מחיקה, יצירה) **גם** ב branch הבסיס **וגם** ב branch היעד, **יחסית לאב הקדמון**, הרי שיש קונפליקט והמשתמש יצטרך להכריע כיצד לפתור את הקונפליקט הנ"ל (כלומר איזו גרסה של קובץ היא הנכונה וצריכה להישמר כחלק מתהליך המיזוג). בכל המקרים שבהם נדמה כי **רק** צד אחד התעסק עם קובץ כלשהוא, וצד אחר **התעלם מקיומו של קובץ זה לחלוטין** – הדבר לא ייחשב כקונפליקט (גם אם תאורטית זה יכול להביל לאסון...)

בגמר תהליך זה תציגו למשתמש חלון של רשימת הקונפליקטים שנצברה ותאפשרו לו לפתור אותם. (שימו לב, ייתכן גם תהליך merge שלא יהיו בו שום קונפליקטים... במקרה זה ממשיכים הלאה)

פתרון קונפליקט בודד כולל (בעיקר) את הצורך למזג את מצבם של 2 קבצים שונים. תוצאת הקובץ הממוזג תשמר במקום הרלבנטי ב WC, ותחשב כשינוי פתוח.

המשתמש יוכל לראות את רשימת הקונפליקטים, לבחור אחד מהם ואז ייפתח לו חלון חדש (stage חדש) שבו תתבצע פעולת פתרון הקונפליקט.

לטובת תהליך פתרון הקונפליקטים תצטרכו למצוא ולהיעזר במצבו של הקובץ המדובר באב הקדמון. (לתהליך זה קוראים 3-way merge). וכך, יש להציג למשתמש 4 'אזורים' המציגים את תכולת הקובץ עם הקונפליקט במצבו השונה:

* + 1. תכולת הקובץ בבסיס (ours)
    2. תכולת הקובץ באב הקדמון
    3. תכולת הקובץ ביעד (theirs)
    4. מצב הקובץ אחרי המיזוג (קובץ התוצאה)

המשתמש יחליט אילו חלקים הוא מעתיק מגרסת קובץ הבסיס/יעד/אב קדמון לקובץ התוצאה.

כשיסיים, ישמור המשתמש את קובץ התוצאה (במקומו הרלבנטי ב WC) ויוכל לבחור את הקונפליקט הבא מרשימת הקונפליקטים לטיפול.

כשכל רשימת הקונפליקטים ריקה – יש כרגע למעשה אסופה של שינויים פתוחים ב WC, עבורם יש לייצר קומיט, כמו בתרגיל 1.

**שימו לב** שקומיט התוצאה יירשם על branch ה ours, ועל כן רשימת השינויים הפתוחים בעצם נקבעת יחסית ל branch הבסיס (ours). אם בתהליך המעבר על הקבצים מסתבר שהשינוי הוא אצל Ours יחסית ל theirs ולאב הקדמון (נגיד קובץ שנוצר, נמחק או התעדכן רק ב ours) – הרי שאז הוא כבר קיים בגרסתו זו ב WC (המייצג את ours), ועל כן לא נחשב כ"שינוי פתוח" בצורה פורמלית.

כפועל יוצא מכך לזהות ה ours בפעולת ה Merge יכולה להיות השפעה על מיהם השינויים הפתוחים דה-פקטו שיופיעו כחלק מהפעולה...

יש לבקש מהמשתמש הודעה המתארת את הקומיט ולבצע קומיט של כל הקבצים הפתוחים במערכת.

שימו לב כי לקומיט זה **יש 2 אבות** (קומיטים קדומים) ויש לציין את ה SHA-1 של שני האבות בתיאור מבנה הנתונים של הקומיט.

(לטובת הבנה טובה יותר של תהליך ה merge בידקו גם את הדוגמא המוצגת בנספח ב')

**Fast Forward Merge:**

מקרה פרטי של מיזוג בין branch'ים שיש לתמוך בו הוא מקרה של Fast Forward (ידוע גם כ merge FF).

F2 **>>**

F1 **>>**

במקרה זה אנו מנסים למזג בין 2 ענפים כך שאחד מהם כבר מכיל את השני,

כלומר, יש "קו ישיר" ללא פיצולים המחבר בין האחד לשני. ה"קו הישיר" מסמן למעשה קשר של הכלה מלאה של אחד ה branch'ים את האחר.

במקרה זה, למעשה, אין באמת צורך לבצע את פעולת המיזוג:

* אם F1 הוא ה branch האקטיבי וממזגים לתוכו את F2, היות ו F2 כבר מכיל את F1 בעברו – המיזוג פירושו רק הכוונת F1 להצביע ל F2. (אין קונפליקטים, אין דילמות.. תוצאת המיזוג היא כבר F2 !)
* אם F2 הוא ה branch האקטיבי וממזגים לתוכו את F1, היות והוא כבר מכיל את F1 – אין מה למזג ! (הודעה מתאימה כמובן למשתמש...)
  1. בתרגיל זה תתמכו גם בעבודה ובשיתוף מידע עם repositories אחרים, מרוחקים.

שימו לב כי ב"מרוחקים" הכוונה היא אך ורק ל repositories אחרים היושבים בתיקיות אחרות על גבי מע' הקבצים.

**לא נתמוך** (בתרגיל זה) בגישה ל repositories המצויים במחשבים אחרים ממש דרך הרשת...

לכל repository המעורב בעבודה עם repository מרוחק יהיה ידע לגבי מיקום ה repository המרוחק (להלן ה remote repository), שכן הוא יצטרך לדעת "למשוך" (pull) ממנו מידע כמו גם "לדחוף" (push) לו מידע חדש.

אם כן, בשיתוף פעולה בין repositories נגדיר 2 צדדים:

* + 1. Local repository (LR) – ה repository הנוכחי שבמסגרתו אנו עובדים ואת המידע שלו אנו רואים.
    2. Remote repository (RR) – ה repository המרוחק (ספרייה אחרת על גבי מע' הקבצים) שמולה אנו מסתנכרנים, מעדכנים ומתעדכנים במידע.

ל repository יכול להיות לכל היותר קשר עם remote repository אחד.

כאשר עובדים עם RR, פירושו כי ב LR יושבים כל הקומיטים וה branch'ים שקיימים גם ב RR, הכל כולל כל ההסטוריה !

כדי לתעד את המיקום של ה branch'ים השונים של RR ב LR נוסיף הגדרה נוספת על branch: remote branch (RB).

התפקיד של RB הוא לייצג ב LR את המקום האחרון בעת העדכון של ה branch ב RR. היות וכך אין "לעבוד" על RB ; הוא ישתנה **אך ורק** באמצעות פקודות המעדכנות מידע הלוך וחזור בין 2 ה repositories. (יפורטו בהמשך)

(בקובץ ה XML, RB מיוצגים כך ש is-remote=true)

כדי להיות מסוגל לעבוד על branch שהגיע מה RR, נגדיר סוג חדש של branch הנקרא remote tracking branch (RTB). זהו branch לוקלי, רגיל, שכל תפקידו "לעקוב" אחרי RB מסויים. על ה RTB נוכל "לעבוד" ולשנותו (כפונקציה של התקדמות הקומיטים) ובאמצעותו ובאמצעות היחס שלו ל RB – M.A.Git תדע לחשב את דלתאות הקומיטים החדשים שיש להעביר/למשוך ל/מ RR. (ראו דוגמאות מפורטות בנספח ג').

(בקובץ ה XML, RTB מיוצגים כך ש tracking=true והאלמנט tracking-after יכיל את שם ה RB שאחריו עוקב RTB זה)

שם branch ה RTB , לשם הפשטות, יהיה שם ה branch כמו של ה branch ב RR.

**דוגמא**: אם ב RR יש branch בשם f1 ב repository ששמו repo1 אזי ה RB המתאים ב LR מסויים ייקרא repo1\f1. ה RTB המתאים ייקרא אם כן פשוט f1.

במסגרת יכולת זו, במידה ורוצים לייצר branch שהוא RTB על RB קיים, ייתכנו מספר דרכים:

1. יצירת branch חדש המצביע על קומיט שעליו מצביע גם RB מסוים (כחלק מסעיף 7ב). במידה ו SHA-1 זה הוא למעשה SHA-1 עליו מצביע RB מסויים (או כמה), יישאל המשתמש האם הוא רוצה לייצר את ה branch החדש כ RTB על ה RB המדובר, או כ branch רגיל.
2. במידה ומנסים לבצע checkout ל branch שהוא RB, יש למנוע זאת, ליידע את המשתמש כי ביכולתו לבחור לייצר במעמד זה RTB ולעשות checkout אליו.. (או כמובן לוותר על כל הפעולה, אם זה רצונו...)

**שימו לב:**

* אם עובדים עם LR המחובר ל RR – אפשר להניח כי זה יקרה למן ההתחלה (כלומר לא יהיו קומיטים קדומים ב LR שאינם קיימים ב RR. במילים אחרות – בעבודה עם collaboration ה LR תמיד יתחיל מ clone מ RR מסויים.
* לא אתעלל באפקליציה שלכם ואשנה, אמחק או אעדכן (שלא מן המניין) את RB או את RTB שלא דרך הפקודות החוקיות המאפשרות לעשות זאת. כלומר אפשר להסתמך על מצב תקין של המע' ולא צריך להתמודד עם כל מיני מצבי קצה שבהם יש RB ב LR אבל אין לו תואם ב RR וכן הלאה.

תהליך העבודה עם RR כולל מספר פקודות חדשות למימוש:

**Clone:** מייצר repository חדש על ידי שכפול repository קיים.

המשתמש יכניס נתיב מלא לספרייה קיימת של M.A.Git (ה RR) כמו גם נתיב מלא לספריית היעד אליה הוא ירצה לשכפל את ה repository (ה LR). בנוסף יכניס המשתמש את שם ה repository החדש שיווצר. (לכל repository יש שם...)

השכפול הוא מלא, וכולל את כל המידע שיש ב repository המקורי: כל הקבצים והספריות לדורותיהם, כל הקומיטים וכל ה branch'ים.

עבור כל branch ב RR יווצר RB ב repository המשוכפל. יש לסמן אותו בצורה מיוחדת (עדיף צבע אחר).

השם של הRB יהיה בפורמט הבא: <remote repository name> \ <remote branch name>

מאחר ולא ניתן לייצר ב windows שם קובץ המכיל בתוכו את התו '\' או '/', יש לתחזק את ה branch'ים האלה בצורה הבאה:

כל הפרדה עם '\' למעשה תכתיב שם ספרייה שבתוכה ישב הקובץ

דוגמא:

אם משכנו מידע מ repository בשם repo 1, ובה היה branch בשם feature1, אזי שם ה branch, בשבתו כ RB, הוא repo 1\feature1. אם כך, קובץ ה branch ישב במיקום הבא:

…\.magit\branches\repo 1\feature1

רק עבור ה HEAD ב RR יווצר RTB אוטומטית מתאים ב LR והוא כמובן יהיה ה HEAD ב LR. שמו של ה RTB, כאמור, יהיה בדיוק כמו שמו של ה branch ב RR...

**Fetch:** מייבאת את כל המידע החדש שהצטבר ב RR אל ה LR.

לאחר פקודה זו ה RR מסונכרן מול ה LR (אך לא להיפיך: ייתכנו מידעים חדשים ב LR שאינם מופיעים ב RR...)

המידע החדש כולל branch'ים חדשים שנוצרו, עדכונים למיקומים של branch'ים קיימים, קומיטים, ספריות וקבצים חדשים שהתווספו.

לאחר פקודה זו כל ה RBs יתעדכנו למיקומים החדשים שלהם כפי שהם מוצגים כרגע ב RR.

שימו לב כי ה RTB נותרים במקומם !

כמו כן אפשר לבצע פקודה זו גם אם WC מלוכלך. (הרי משתנים מאחורי הקלעים רק RB'ים שגם ככה אף אחד לא יכול לעשות להם checkout...

**Pull:** מייבאת רק מידע של branch מסויים, ה head branch, מן ה RR.

בפעולה זו נייבא מה RR **רק** את הקבצים, ספריות, קומיטים של ה branch המדובר. לאחר מכן נעדכן את ה RB, כמו גם את RTB להצביע על המיקומים הרלבנטים שלהם.

שימו לב:

* יש טעם לבצע פקודה זו אך ורק אם ה head branch הוא כמובן RTB אחרי RB אחר.
* פעולה זו ניתנת לביצוע אך ורק אם המערכת נקייה משינויים. (יש להתריע על כך למשתמש אם לא כך הדבר)
* פעולה זו לא מייבאת שינויים נוספים שקרו ב RR יחסית ל LR ב branch'ים אחרים...
* פעולה זו לא תתבצע אם היו עוד שינויים ב LR על ה RTB שטרם עברו Push (ראו פקודה הבאה)

**Push:** דוחפת מידע חדש (קומיטים, ספריות, קבצים) מ LR ל RR.

את המידע החדש ניתן לדחוף אך ורק אם ה RR "נקייה" משינויים. אם זה לא המצב יש להתריע על כך בפני המשתמש.

הפקודה הזו מתבצעת בהקשר ה head branch, תחת האילוצים הבאים:

* אם ה head branch הוא RTB העוקב אחרי RB מתאים (ב LR).
* אם ה branch הרלבנטי ב RR נמצא **בדיוק** היכן שהוא מסומן על פי ה RB בLR .

במילים אחרות: אם לא בוצעה עוד עבודה על branch זה ב RR מאז הפעם האחרונה שהמידע שלו סונכרן ל LR.

בסיום פקודה זו מיקום ה branch הרלבנטי ב RR יתעדכן בהתאם וכך גם ה RB יתעדכן להצביע על מיקומו של ה RTB שבגינו בוצעה הפעולה (שכן עכשיו הוא מתואם מול ה branch המתאים ב RR...)

1. כחלק מבדיקת המערכת ישונה גודל המסך (resize) ותיבדק המע' שלכם במסך בגודל שונה. עליכם לדאוג לסידור נכון של רכיבי ה ui ולוודא את תקינותם גם בגודל קטן. מומלץ מאוד להשתמש ב scroll pane (יש ללמוד עליו לבד) כדי להציג תוכן גדול בתוך שטח מסך קטן. בפרט עליכם לדאוג להראות את לוח המשחק (שיכול להגיע בגדלים שונים ומשונים) בצורה נאותה כך שאף אחד מרכיביו לא יוסתר בגלל גודל מסך קטן מדי. כמו כן חישבו איך לא להסתיר את טבלת השחקנים במשחק כמו גם את היכולת לבצע מהלך ולהתקדם במשחק.

**למען הסר כל ספק ומניעת כל התחכמות שהיא, אין "לטפל" ב resize ע"י כך שפשוט תמנעו מהמסך להיות resizable.**

**במידה ומימשתם בונוסים בתרגיל הקודם אין הכרח לגרום להם לפעול גם בתרגיל זה, אולם אם זה מתאפשר זה יחמם את ליבי (למען הסר ספק, חימום ליבי אינו מתורגם להעלאת נקודות).**

## חלוקה למודולים

בתרגיל זה יש לממש את מודול ה Java FX כ jar עצמאי שעובד מול מודול מנוע המערכת (שגם הוא משתכלל כפוקנציה של הדרישות השונות).

מומלץ לפתח את המודול של JavaFX באותו פרוייקט ב intelij, כמודול נוסף ליד מודול מנוע המערכת.

## איך מתחילים ? (המלצה...)

ממליץ להתחיל ממימוש האפליקציה כולה כ Java FX. שם טמון עיקר הקושי. בתוך עולם ה java fx השאירו את עץ הקומיטים לסוף. במקומו התחילו במידת הצורך באזור המכיל פירוט רשימת הקומיטים בצורה שטוחה בלבד (כמלל) ורק אחרי שרוב האפליקציה קיימת ועובדת (מבחינת ה java fx) התחילו להתעסק עם מימוש עץ הקומיטים (אין ספק שהוא הרכיב "המורכב" יותר בסיפור).

רק אחרי שסיימתם עם זה אני ממליץ להתחיל להתעסק עם ה Merge. זו יכולת לוגית בעיקרה, וקשה ומורכבת ככל שהיא נראית - בסוף זה רק לוגיקה דטרמיניסטית וכך או אחרת תצלחו אותה (וגם אני פה כדי לעזור... 😊 ) אחרי שסיימתי את המרג' תרכיבו לו כמובן את ה UI המתאים (שידבר עם הלוגיקה שלכם).

רק אחרי ה Merge התפנו לממש את ה collaboration. ולבסוף את ה UI התומך בו. (זה נראה מפחיד אבל זה בעצם רק אסופה של branch'ים שעושים להם Merge...

## בונוסים

היכן שזה רלבנטי (בונוס 1 ו 2) על הבונוסים להגיע "מכובים" בתור התחלה כך שהבודק "יפתח" אותם רק לטובת הבדיקה שלהם

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | סוג | מהות | למה שווה לי ? | כמה שווה לי ? |
| 1 | פשוט מגניב ! | אנימציות  ממשו (לפחות) את האנימציות הבאות:   * איפוס branch מזיז אותו למיקום החדש על עץ הקומיטים * כשלוחצים על branch בעץ הקומיטים מדגישים את כל הקומיטים השייכים לו ב"פעימת לב" * כל אנימציה נוספת שאתם יכולים לחשוב עליה.   האנימציה צריכה להתבצע לכל היותר במשך 2 שניות  יש לאפשר גם לנטרל את האנימציות הנ"ל, כך שהיא לא תאיט את התקדמות המערכת | כי זה מה-זה קל... | 10 נקודות  (ועד ל 100) |
| 2 | הגדלת ראש נאה | אפשרות החלפת skin למשחק. בבונוס זה תממשו (לפחות) 2 ערכות צבעים נוספות על הערכה הדיפולטית, ותאפשרו למשתמש להחליף את ערכות הצבעים. שימו לב ערכת הצבעים כוללת (לכל הפחות):   1. החלפת רקע של כל המסך 2. החלפת המראה של הכפתורים על המסך 3. החלפת הפונט והגודל של כל ה label על המסך | כי זה בכל זאת תרגיל בממשק גרפי – ומה יותר גרפי מזה ??? | **5 נקודות**  **(ועד ל 100)** |
| 3 | הגדלת ראש נדרשת ! | הוסיפו תפריט הקשר (context menu) לבחירה של כל קומיט בעץ הקומיטים. מתפריט ההקשר ניתן יהיה לבצע את הפעולות השונות בהקשר של קומיט/branch זה ביתר קלות:   1. יצירת branch חדש המצביע על קומיט זה 2. איפוס head branch לקומיט זה 3. מיזוג ה branch המצביע על קומיט זה עם head branch 4. מחיקת branch (במידה ויש כזה המצביע על קומיט זה) | קל.... שחבל ! | **10 נקודות**  **(מעל ל 100)** |
| 4 | הקדמת תרופה למכה | ממשו את היכולת לדחוף branch חדש ל RR כפי שמוגדר בסעיף 6 בתרגיל 3 | יחסוך לכם מעט זמן מתרגיל 3 העמוס ממילא | 5 נקודות  (ועד ל 100) |
| 5 | יצירתיות  (I) | גרף עץ קומיטים מרהיב ביופיו, שנינותו וייצוגו !  בונוס זה ייבחן את אלגוריתם ה layout שלכם לקומיטים השונים בעץ הקומיטים, כמו גם את הנראות והשימושיות של כל הגרף.  קשה לי להצביע על דבר מסויים שצריך לבצע בבונוס זה – זה יהיה יותר בכיוון שאבקש מהבודק לאמר לי מיהי ההגשה שבה עץ הקומיטים היה המרהיב, הנהיר, הבהיר והיפה ביותר.... כך ש... תפציצו ! | אם יש לכם זמן ובא לכם להתפרע... | **5 נקודות**  **(מעל ל 100)** |
| 6 | יצירתיות  (II) | הדגשת branch בגרף עץ הקומיטים:  עץ הקומיטים שלכם צפוי להראות הרבה branch'ים וככל שיגדל כך סבך הקשרים בין הקומיטים צפוי להיות מבולבל יותר ויקשה על המשתמש לעקוב איזה קומיטים שייכים ל branch מסויים ? מיהו האבא של קומיט מסויים וכן הלאה. כדי לפתור זאת אפשרו להדגיש בצורה מסויימת לבחירתכם (הבלטה (bold), שינוי צבע, הגדלה...) את כל הקומיטים הקשורים ל branch מסויים.  לטובת כך יש לאפשר למשתמש פעולה של בחירה של קומיט אחד בעץ ואז יודגשו/יובלטו כל רצף הקומיטים שלו (כולל הקווים המחברים בין קומיטים, כולל העיגולדים המתארים קומיטים) יחסית לשאר המידע על העץ. בדרך זו יוקל על המשתמש להבין מיהם כל הקומיטים לפני/אחרי הקומיט המסויים ולסייר בעצים מורכבים וארוכים ! | המידע כבר שם – רק להדגיש אותו... | **7 נקודות**  **(מעל ל 100)** |
| 7 | הגדלת ראש על שם נווה פרג'ון (שוב...) | תמיכה ב(קומיטים) יתומים ואלמנות:  בברירת המחדל עץ הקומיטים מכיל אך ורק את הקומיטים המוצבעים ע"י branc'ים ו/או קומיטים אחרים. בבונוס זה תדרשו להציג עץ קומיטים "מלא" המכיל גם את כל הקומיטים היתומים שאין להם (כך או אחרת) ראשית בהצבעה ע"י branch.  בבונוס זה תדרשו לאפשר למשתמש לבחור במצב תצוגה של 2 עצים שהוא יוכל לברור בינהם:   1. עץ דיפולטי – כזה המציג **רק** את הקומיטים המוצבעים ע"י branch'ים ו/או קומיטים אחרים 2. עץ מלא – כזה המציג את **כל** הקומיטים, כולל היתומים מהצבעות מקור | תלוי באלגוריתם סידור הקומיטים שלכם – יכול להיות משימה של מה בכך... | **8 נקודות**  **(מעל ל 100)** |

## סבבה, סיימתי. מה ואיך להגיש ?

יש להגיש קובץ zip המכיל:

1. 2 jar'ים (לפחות) המהווים את כל הקוד שלכם, בצירוף קובץ אצווה (batch) שהפעלתו תריץ את התוכנית
2. קבצי קוד המקור של הפרויקט שלכם.
3. קובץ ההגשה יכיל גם קובץ readme שיכיל הסבר על המערכת, בחירותיכם השונות במקומות שבהם היו לכם בחירה, כמו גם כל דבר נוסף העולה על דעתכם שחשוב שהבודקת תדע.
4. יש לכלול בקובץ ה **readme** גם תיעוד והסבר כללי (וממצה) של המחלקות העיקריות החדשות ותפקידם. (חישבו מה יסייע לבודק להיכנס ביתר קלות לקוד שלכם ולהבין מי נגד מי...)
5. במידה ומימשתם את מי מהבונוסים לעיל, ציינו זאת בקובץ ה readme כדי שהבודק ידע לבדוק בהתאם.

בונוס שימומש אבל לא יתועד – לא ייבדק !

## שאלות ותשובות

|  |  |  |  |
| --- | --- | --- | --- |
| # | תאריך | שאלה | תשובה |
| 1 | 21.8.19 | צריך להציג עבור כל קומיט דלתא מול קומיט קודם (בעץ הקומיטים). מה עושים עם קומיט שהוא Merge ושיש לו 2 אבות ? איזו דלתא מציגים ? | כך או אחרת צריך לאפשר להציג את הדלתא מול כל אחד מהאבות. פירטתי בסעיף המדובר (8) גם את המקרה הזה ביתר פירוט |
| 2 | 21.8.19 | כדי לגלות דלתא של WC משווים אותו מול הקומיט הנוכחי או מול הקודם | מול הקומיט הנוכחי. בטקסט המקורי נפלה טעות סופר. תיקנתי אותה עכשיו (סעיף 6ב') |
| 3 | 26.8.19 | האם כל שינוי כתוצאה מ merge נחשב כשינוי פתוח ? | לא. אם השינוי הוא כבר ב branch הבסיס, שעליו יתבצע הקומיט בסופו של דבר, הרי שאז הקובץ כבר נמצא במצבו הסופי ב WC ועל כן הוא לא ייחשב כשינוי פתוח. מוגדר בצורה ברורה יותר כחלק מסעיף 9 בעמוד 28 |
| 4 | 26.8.19 | במהלך פעולת ה merge, מתי שינוי ייחשב כקונפליקט ? | בגדול, בכל המצבים שבהם 2 הצדדים "הביעו עניין" במצבו של קובץ מסויים, **יחסית למצבו באב הקדמון**, הרי שאז מתקיים מצב של קונפליקט.  "להביע עניין" פירושו כל פעולה על הקובץ: ליצור אותו ; למחוק אותו ; לעדכן אותו.  שימו לב כי זה כולל בתוכו כמה מקרי קצה:   * למשל קובץ שלא היה קיים באב הקדמון, ונוצר באותו השם ובאותו התוכן בשני הצדדים * קובץ שעודכן ב 2 הצדדים לאותו העדכון בדיוק |
| 5 | 26.9.19 | האם אפשר לעשות merge ואת הקומיט שלו אחר כך ? | לא. פעולת ה merge היא פעולה אחת הכוללת כמה שלבים שכולם מתבצעים כמקשה אחת או לא מתבצעים בכלל. הקומיט הוא השלב האחרון הסוגר את הפעולה. (מתואר עכשיו בראשית סעיף 9, סוף עמוד 27) |
| 6 | 28.9.19 | אחרי clone, מי אמור להיות ה HEAD ב LR ? | אחרי clone יש לייצר אוטומטית RTB מתאים ל HEAD ב RR.  מוגדר (עכשיו) גם כחלק מפעולת ה clone (עמוד 30) |
| 7 | 29.9.19 | מה צריך להיות סדר הופעת הקומיטים בעץ הקומיטים ? | הקומיטים החדשים יותר צריכים להופיע ראשונים והקומיטים הישנים יותר צריכים להופיע אחרונים. העץ מצוייר מהעדכני ביותר (למעלה, בראש הגרף) ועד לישן ביותר (למטה, בתחתית הגרף).  שורש העץ מצוייר בתחתית. |
| 8 | 9.9.19 | האם צריך לאפשר לבצע PUSH כשב RR ה RTB הוא ה HEAD וה WC (של RR) "מלוכלך" בשינויים פתוחים ? | אין לאפשר זאת, אולם יש להבין מצב זה ולהתריע בפניו בפני המשתמש. מתואר כחלק מפקודת ה push, ראשית עמוד 30) |

תרגיל 3 – מימוש "M.A.Git" כאפליקציית WEB (35%) – הגשה: 26.10.19

## פרטים יבשים

צפי תחילת עבודה: **15.9.19** תאריך הגשה:**26.10.19**

צפי זמן לביצוע: **6 שבועות** ציון אפשרי מקסימלי: **145**

משקל התרגיל: 35**%** קושי: **ווא'עליה אלל'בחטי**...

**מטרות התרגיל העיקרית**

1. מימוש המשחק כאפליקציית web.
2. מימוש תצורה של ריבוי משחקים

## דרישות

1. בתרגיל זה נפתח מע' web שתציע למשתמשים ממשק נוח לשיתוף פעולה של קוד המבוסס על יכולות M.A.Git.
2. בתרגיל זה כל משתמש מתחבר למע' מ browser שלו, והאינטראקציה שלו עם המשתתפים האחרים מתבצעת למעשה רק ע"י השרת המתווך בינהם.

פורמלית, יש לתמוך בחווית משחק תקינה מעל chrome בלבד.

גם אם פיתחתם מעל FF או IE (טפו..) – וודאו כי אתם עובדים בצורה מוצלחת מעל chrome.

האפליקציה תיבדק רק מעל chrome.

בקיצור chrome,

קאפיש ?

1. ככלל, כל פרטי המידע במע' צריכים להתעדכן בצורה אוטומטית עבור כל משתמש (=== כל browser) בצורה של Pull כפי שלמדנו בשיעור והודגם בדוגמא המסכמת.
2. אין צורך לתמוך ב back כאמצעי למעבר בין המסכים השונים. כמו כן הניחו כי לא נתעלל באפליקציה ולא נשנה את שורת הכתובת, לא נמחק cookies וכו'. הבדיקה תתבצע אך ורק ע"י שימוש בפונקציונליות שתאפשרו במסכי המשחק השונים (משל היינו הוריכם...)
3. אין צורך לבצע שמירה של נתונים מעבר למופע הנוכחי של השרת. במילים אחרות כשהשרת יורד - כל הנתונים נעלמים.

לא שומרים את המשתמשים שנרשמו, את ה repositories שהוגדרו וכו'.

1. בתרגיל זה תתמכו ביכולת "לדחוף" branch חדש ל RR. הדבר מתאפשר כמובן רק במצב שבו משתמש שיכפל את ה repository שלו מ repository אחר. המשתמש יוכל לייצר branch חדש לחלוטין אצלו (שאינו RTB), ולדחוף אותו ל RR. בעשותו כן, M.A.Git תחשב ותעביר ל RR את דלתאות הקומיטים על branch זה ל RR (הדלתאות יחושבו יחסית ל קומיט הראשון על גבי ה branch שקיים גם ב RR). לאחר סיום הפעולה יווצר ב LR (שיזם את הפעולה) RB מתאים (שכן עכשיו יש כבר את ה branch הזה ב RR) , וה branch המקורי ב LR שנדחף ל RR יהפוך להיות מוגדר כ RTB (כי עכשיו הוא הרי נועד לעקוב אחרי ה RB החדש שנוצר). בסיום תהליך זה ה branch החדש יהיה קיים כמובן גם ב RR, תחת אותו שם שהוא קיבל ב LR.

ניתן לבצע פעולה זו רק כשה branch החדש הוא HEAD branch.

1. בתרגיל זה נממש תהליך נוסף שבמסגרתו ניתן יהיה לעדכן RR בתוכן שנוצר ב LR.

M.A.Git מאפשרת רק "דחיפה" של נתונים ועדכונים מ LR ל RR. זה נחשב "לא מנומס" ומאוד לא יציב שכן מישהו מבחוץ מתיימר לבוא ולשנות מצב של אחד ה branch'ים ב RR...

כחלופה לתהליך זה, נהגה תהליך אחר, "מנומס" יותר. תהליך זה נקרא Pull Request (PR).

בתהליך זה כשמשתמש ב LR רוצה לעדכן את ה RR במידע חדש, הוא לא "דוחף" לו אותו ישירות דרך RB המדובר, אם כי מייצר "בקשה" המכילה את המידע לעדכון היושבת אצל ה RR בצד בדמות branch. המשתמש למעשה מבקש מהבעלים של RR לבחון את תוכן הבקשה ולהחליט לבד, בזמנו החופשי האם בכלל/מתי נכון לו לבצע ולהחיל את העדכון של המידע החדש לתוכו.

במילים אחרות, משתמש ה LR מבקש (Request) מהבעלים של RR "למשוך" (Pull) את המידע החדש בזמנו החופשי ולעשות בו כרצונו. על כן התהליך הזה נקרא Pull Request.

ה PR ממודל כ branch היושב ב RR המכיל את כל המידע שרצה LR לעדכן בו את RR. נכנה אותו branch המטרה (סעיף 6). את ה branch הזה יש להשוות ל branch אחר שכבר נמצא אצל RR, נקרא לו branch הבסיס.

הנחת היסוד היא כי branch הבסיס מוכל בתוך branch המטרה, כך שבחישוב ההבדלים בינהם ניתן לייצר את דלתא העבודה שהצטברה ואותה יש למרג'ג'.

ביצוע ה PR (ע"י הבעלים של RR) פירושו למרג'ג' את branch המטרה אל תוך branch הבסיס.

אם כן, PR יכלול מספר פרטי מידע:

* 1. שם של Branch המטרה המכיל את כל הקומיטים והאובייקטים הנדרשים לעדכון.
  2. שם של branch הבסיס, ב RR, אליו צריך למזג (למרג'ג') את המידע של branch המטרה.
  3. הודעה המתארת את מהות ה PR.

אם כן, סיכום תהליך ה PR כולו (סעיף 6 + 7) כולל את השלבים הבאים:

1. משתמש ה LR מייצר branch חדש (רגיל) אצלו, לוקלית, ובו מייצר קומיט(ים) המכילים את המידע הרלבנטי. המידע הרלבנטי הוא רצף השינויים בקומיטים החל מקומיט שנמצא ב RR ועד הקומיט הנוכחי המוצבע ע"י ה branch. המשתמש דוחף בזמנו החופשי את ה branch החדש ל RR (סעיף 6). פעולה זו עצמאית ואינה קשורה מבחינת הזמנים להמשך התהליך.
2. משתמש ה LR מייצר PR אצל ה RR ובו הוא מבקש למזג את תוכן ה branch החדש (המטרה) עם תוכנו של branch מסויים שכבר נמצא ב RR (הבסיס). ה PR נחשב כרגע "פתוח" שכן הוא ממתין להחלטת ה RR לגביו.
3. משתמש ה RR מקבל עדכון (כך או אחרת, יפורט בהמשך) כי ממתין עבורו PR לבחינה.
4. בזמנו החופשי, וככל שהוא חפץ בכך, משתמש ה RR בוחן את תוכן ה PR, מתרשם מאיכות וטיב השינויים ומחליט האם/מתי לבצע merge (למזג) בין branch המטרה של ה PR (ה branch "החדש" שנדחף מה LR) לבין branch הבסיס, אצלו ב RR.
5. לאחר המיזוג, המידע החדש יתעדכן ב branch הבסיס ב RR. ה PR "נסגר" והודעה מתאימה תגיע למשתמש ה LR בנוגע למהות הפעולה (כך או אחרת, יפורט בהמשך).
6. משתמש ה LR יכול עכשיו לבצע pull ל branch הבסיס ולקבל אותו בגרסתו המעודכנת המכילה כבר את התוכן "החדש" שהוא רצה לעדכן את RR בקיומו.
7. משתמשי ה LR וה RR יכולים עכשיו למחוק את branch המטרה של ה PR, או לשמור אותו לצרכיהם העתידיים.
8. לחילופין, אם משתמש ה RR מחליט כי הוא דוחה את ה PR (מסיבותיו שלו) הוא יוכל לעדכן זאת על גבי ה PR (בתוספת הסבר מדוע) וגם אז משתמש ה LR יקבל על כך חיווי מתאים. מטעמי פשטות, אם PR נדחה, הוא נחשב כסגור ולא ניתן להמשיך את הפעילות עליו (לנסות שוב להגיש עליו עוד מידע וכו').

מנומס או לא מנומס ?

1. יש לדאוג ולוודא כי מסכי המערכת לסוגם מתנהגים יפה ב resize ולא מחסירים שום פרט.

הם יכולים להיות מכוערים, אבל שישארו פונקציונלים (כשם שהיה נהוג בתרגיל 2).

1. בתרגיל זה ישנם 3 מסכים עיקריים:
   1. **מסך רישום למע' (sign up)**

במסך זה יש לאפשר למשתמש להירשם למע'.

לכל משתמש יש שם ייחודי לו ורק לו. משתמש יוכל להתחבר ולהתנתק מהמע' כמה פעמים. אם משתמש חוזר למע'

ונרשם עם אותו שם – על המע' לזהות אותו ולאפשר לו להתחבר הלאה ולראות את כל הפרטים הרלבנטים לו.

אחרי רישום מוצלח תעברו למסך מס' 2...

* 1. **מסך ריכוז פרטי המשתמש ותכולתו**

במסך זה המשתמש רואה את החשבון הפרטי שלו ב M.A.Git hub.

כל משתמש יכול לנהל כמה repositories שירצה במערכת.

כדי לייצר repository פרטי על המשתמש להעלות קובץ המתאר repository. הוא עושה זאת ע"י בחירת קובץ מע' (ממחשב הקליינט) ומעלה אותו אל השרת.

ההעלאת הקובץ יכולה (וצריכה) להתבצע בהתאם לדוגמא שנלמדה בכיתה. אין שום צורך להשתמש בספריות צד שלישי (apache commons וכיוצב') בשביל להעלות את הקובץ. אין שום צורך (ויש להימנע מכך בכל מחיר !) לשמור את תוכן הקובץ בשרת. ייתכן ולשרת (של הבודק/ת) אין הרשאות לכך ואתם תקרסו ! (הו"ב נכתבו בדם...)

כשתוכן הקובץ מגיע לשרת, יבוצעו בו בדיקות הקלט כפי שבוצעו בתרגיל 1 ו 2.

באם הקובץ תקול – הודעה מתאימה הכוללת את סיבת התקלה צריכה להופיע למשתמש (ולמשתמש הזה בלבד), והקובץ **אינו** מוכר כתקין בצד השרת.

אם הקובץ תקין – הודעה מתאימה צריכה להגיע למשתמש, ואז הוא יופיע ברשימת ה repositories שלו.

על כל repository שכזה ברשימה יש לתת את הפרטים הבאים:

* שם ה repository.
* שם ה active branch
* כמות ה branch'ים
* מועד ביצוע הקומיט האחרון על פי ציר הזמן
* הודעת הקומיט האחרון שבוצע (על פי ציר הזמן)

היכן נשמרים ה repositories ?

ככותבי האפליקציה בצד השרת תוכלו להחליט לבד איך לשמור את ה repositories של המשתמש. היות ותצטרכו לתחזק כמה משתמשים ולכל אחד מהם כמה repositories, ואולי כולם אף באותו מקום ובאותו שם – כדאי מאוד להחליט לשמור את כולם לספרייה אחת, ובה ספרייה לכל שם משתמש ובתוכה כל ה repositories של כל משתמש כאשר לכל repository יש ספרייה עם השם שלה. כלומר בתרגיל זה אין להשתמש יותר ב location מה xml !

כדי לשמור על אחידות, עליכם לשמור את כל המידע בספריית עבודה במחשב השרת תחת המיקום c:\magit-ex3, בתוכה תוכלו לשמור את ה repositories השונים לפי איזו חלוקה וסדר שתרצו.

בעליית המע' עליכם לייצר ספרייה זו בעצמכם (היא לא תיוצר עבורכם), כמו גם את כל התוכן שתרצו לאחסן בתוכה ; בירידת המע' (בצורה מסודרת בלבד, כפי שנלמד בכיתה) עליכם לדאוג לנקות ולמחוק את כל תוכנה של הספרייה הזו ולהשאיר את מע' הקבצים נקייה כפי שקיבלתם אותה (עליכם למחוק גם את הספרייה עצמה c:\magit-ex3).

(יש לעבוד כאן כמובן עם ContextListener כפי שהודגם בכיתה)

חוץ מרשימת ה repositories המשתמש יכול לראות רשימה של כל המשתמשים שאי פעם היו מחוברים למע' (לאו דווקא אלה המחוברים אליה כרגע !) ויצרו בה repositories.

בלחיצה על משתמש שכזה יש להציג את כל ה repositories של המשתמש הנ"ל. (עבור כל repository יש להציג את המידע שהוגדר לעיל עבור repository של המשתמש עצמו.)

דרך נוספת שמשתמש יכול לייצר עבורו repository במע' היא באמצעות ביצוע fork מrepository של משתמש אחר. מאחורי הקלעים תבצעו clone על פי כל החוקים וההגדרות של תרגיל 2.

בעמוד זה יש לקבל הודעות/התראות (notifications) על פעולות שקורות בהקשר המשתמש.

עבור כל הודעה יש להציג את חותמת הזמן שבו היא קרתה, ועוד מידע בהתאם לסוג הפעולה:

* משתמשים שפירקקו את ה repository שלכם (ביצעו fork מכם). בכל הודעה כזו יש לכלול את שם repository שפורקק כמו גם את שם המשתמש שפירקק אותו.
* התווספות PR למי מה repositories של המשתמש (התהליך ידובר ויוגדר בהמשך). על כל הודעה שכזו יש לכלול את שם ה repository, שם המשתמש, הודעת ה PR, שם branch המטרה ושם branch הבסיס.
* עדכון לגבי PR שהגיש משתמש זה אשר אושר/נדחה בהקשר של repository מסוים. יש לכלול את פרטי ה PR כמו גם את סיבת הדחייה.

את ההודעות יש לצבור באוזר מוגדר על המסך. ההודעות תתווספנה לפי הסדר. במידה ונגמר מקום לאזור – יש להיעזר ב scroller.

במידה והמשתמש מתנתק מהמע' ואז מתחבר אליה שוב (תחת אותו השם כמובן), יש להציג עבורו את כל ההתראות שנצטברו עבורו מאז הפעם האחרונה שהוא התעדכן בהן (כלומר **אין** להציג לו את כל ההתראות שלו מאז עליית השרת, אלא רק את הדלתא החדשה, באם קיימת).

עמוד זה יכלול גם כפתור Logout שיאפשר למשתמש להתנתק מהמע'. ברגע שמשתמש מתנתק מהמע' ה repositories שהיו ברשותו **לא נמחקים** ונשארים קיימים במע' כבמקודם ! משתמשים אחרים יכולים עדיין לפרקק את ה repositories שלו כמו גם לייצר עבורו PR. ברגע שהוא יתחבר שוב (פשוט עם אותו השם) – הוא יקבל את כל ההודעות וההתראות שנצברו עבורו מאז התנתקותו האחרונה.

המשתמש יכול לבחור את מי מה repositories **שלו** ולהיכנס לעמוד המציג את תוכנן ומאפשר לבצע בהן פעולות שונות. לחיצה על repository תעביר את המשתמש למסך 3...

(**שימו לב**: משתמש **אינו** יכול לבחור להיכנס ל repository של משתמש אחר – אם הוא רוצה לראות את תוכנה הוא יכול רק לפרקק אותן לעצמו ואז להיכנס אליהן על ה fork שלו..)

* 1. **מסך פרטי repository בודד**

במסך זה ניתן לראות פרטים על repository בודד.

יש להציג את שם ה repository במקום ברור כדי שיהיה ברור למשתמש על איזה repository הוא מסתכל. אם ל repository הזה יש RR הרי שצריך גם לתת מידע על זהות ה RR (שם ה repository ושם המשתמש ש repository זה שייך לו)

עבור כל repository יש להציג מספר פיסות מידע:

**Branch'ים:**

* מידע על כל ה branch'ים שיש במע'
* מידע מיהו ה head branch
* יכולת לייצר branch חדש
* החלפת ה head branch (checkout)

**קומיטים, ספריות, קבצים:**

יש להראות את סדר הקומיטים במע' של ה head branch.

**שימו לב**: כדי להקל, **אין צורך** להציגם כעץ כפי שהיה בתרגיל 2.

אפשר ברשימה פשוטה של קומיטים כפי שהיה בתרגיל 1. על כל קומיט את ה SHA-1 שלו, הודעה שלו, מועד יצירה, מי יצר. רשימת ה branch'ים המצביעים על קומיט זה (במידה וקיימים).

ברגע שבוחרים קומיט מסויים יש להראות את רשימת הקבצים/ספריות שבו.

**שימו לב**: מטעמי פשטות, **אין צורך** להציג זאת כעץ של ספריות וקבצים, ואפשר ומספיק אפילו ברשימה סדורה, שטוחה של קבצים/ספריות.

**עדכון WC ויצירת קומיט חדש**

אפשרו למשתמש לעדכן את WC ע"י כך שתפתחו לו אזור/מסך/עמוד חדש שבו הוא יראה את כל קבצי הקומיט הנוכחי (זה המוצבע ע"י ה Head branch). במצב זה ייפתחו לו מגוון אפשרויות נוספות שתכליתן לאפשר לו לערוך שינויים על גבי WC (וירטואלי, וובי). התוצר של כל אחת מהפקודות מטה הוא למעשה "שינוי פתוח" שנצבר בנבכי השרת עבור repository זה.

אלו השינויים שיש לתמוך בהם:

* המשתמש יכול לבחור קובץ אחד ולראות את תוכנו. אם ירצה יכול ללחוץ על כפתור מיוחד ולערוך את תוכנו של הקובץ. כשילחץ SAVE תוכן הקובץ החדש יישלח לשרת וייחשב כ"שינוי פתוח" לקראת קומיט.
* המשתמש יוכל לבחור את אחד הקבצים ולמחוק אותו (שינוי נוסף שייצבר לקומיט הבא)
* המשתמש יוכל לבחור לייצר קובץ חדש, לצקת בו תוכן ולשמור.

המשתמש יכול לחזור למסך של עמוד 3 בכל רגע נתון ולבחור לשוב למסך ה WC כל אימת שיחפץ. מסך זה תמיד יציג את מצבו העדכני של WC, כפי שהיה בתרגיל 1 ו 2, כלומר מצב הקומיט האחרון המוצבע ע"י ה HEAD בתוספת השינויים שנצברו עד עכשיו.

כמובן שכשיש שינויים פתוחים ב WC יש להציג אותם (כמו שעשיתם בתרגיל 1 ו 2) על פי סוגם (קבצים שהתעדכנו, נוצרו, נמחקו...) כך המשתמש ידע שכרגע הוא אינו יכול לבצע checkout (אחרי הכל WC מלוכלך..)

לכשיחפץ בכך, המשתמש יוכל ללחוץ על כפתור קומיט. אחרי מתן הודעה מתאימה תבצעו קומיט לכל השינויים הפתוחים שנצברו ב WC עד עכשיו. (תהליך יצירת קומיט רגיל כפי שהוגדר ומומש זה לא מכבר בתרגיל 1 ו 2)

**התעסקות עם branch'ים:**

המשתמש יוכל ללחוץ על כפתור ייעודי שיאפשר לו ליצור branch חדש, לוקלי, רגיל במע'. (branch שהוא יתעתד לדחוף ל RR בהמשך...)

שימו לב אין צורך לתמוך ביכולת למחוק branch בתרגיל זה.

**פעולות collaboration:**

המשתמש יוכל לבצע Pull ל branch שהוא RTB בדיוק כפי שיכול היה לעשות בתרגיל 2, עם כל ההנחות שנלקחו גם כן.

המשתמש יוכל לבצע Push ל branch שהוא יצר (התגשמות סעיף 6). **שימו לב:** במסגרת תרגיל זה אין צורך לתמוך בביצוע פעולה זו גם על RTB קיים, בדומה למה שאפשר היה לבצע בתרגיל 2.

הפעולה clone נתמכת מעצם פעולת ה fork במסך מספר 2.

אין צורך לתמוך בפעולת ה fetch בתרגיל זה.

**הכנת ויצירת PR**

המשתמש יכול לבחור כפתור מיוחד שמאפשר לו הגשת PR ל RR המוגדר ל LR זה (בהנחה שאכן מוגדר RR ל LR זה...)

ביצירת PR יש לאפשר למשתמש להכניס את הפרטים הבאים:

* + 1. שם branch המטרה אצל ה LR המיועד לדחיפה ולהוות את המקור למידע החדש
    2. שם branch הבסיס אצל ה RR – אליו צריך למזג את branch המטרה.
    3. הודעה המתארת את מהות ה PR.

לאחר יצירת ה PR הוא נחשב במצב "פתוח" והוא נשלח אחר כבוד ל RR, שם הוא יקבל מספר סידורי מזהה. המשתמש שהוא הבעלים של RR יקבל על כך חיווי אוטומטי (ממתין לו PR פתוח לביצוע).

**צפייה וניהול PR'ים של המשתמש**

כל משתמש צריך לראות את כל ה PR'ים הרשומים עבורו (הן הפתוחים והן הסגורים).

על כל PR יש להציג את המידעים הבאים:

* + 1. שם המשתמש היוזם את ה PR
    2. שם branch המטרה המכיל את הנתונים של ה PR.
    3. שם branch הבסיס אליו ה PR צריך להתמרג'ג'
    4. מועד יצירת ה PR.
    5. סטטוס ה PR (פתוח/סגור/נדחה)

על כל PR יש לאפשר למשתמש לבצע מספר פעולות שיאפשרו לו לדחות/לאשר את ה PR:

* + 1. יכולת צפייה בקומיטים ובקבצים/ספריות שהשתנו ב PR הזה.

עבור כל קומיט יש להציג:

* + 1. הקבצים שהשתנו (נתיב מלא)
    2. בבחירת קובץ – את תוכנו (כמובן לא רלבנטי עבור קבצים שנמחקו)
    3. אופי השינוי (עדכון, מחיקה, יצירה)

**שימו לב**: אין צורך להציג איחוד (אגריגציה) של סך השינויים על פני כל הקומיטים (זהו בונוס).

* + 1. יכולת לאשר PR – במקרה זה יבוצע merge בין branch המטרה ל branch הבסיס. **כדי להקל**, אפשר להניח כי לא יהיו קונפליקטים.
    2. יכולת לדחות PR – במקרה זה יש לאפשר למשתמש להכניס הודעה המספרת את סיבת הדחייה.

בכל מקרה של סיום ה PR (בצורה של #2 או של #3) הודעה מתאימה תשלח "אוטומטית" למשתמש שיזם את ה PR ותודיע לו על גורלו של ה PR לכאן או לכאן.

**התראות**

גם בעמוד זה יש להאזין ולהציג את ההתראות המתקבלות עבור המשתמש, כפי שהוגדרו בעמוד מס' 2. (אותם ההתראות, עם אותם פרטי המידע עליהן..)

**סיום עמוד 3**

המשתמש יוכל לבחור לעבור חזרה למסך רשימות ה repositories (מסך מס' 2) באמצעות כפתור ייעודי.

## חלוקה למודולים

יש לייצר מודול נוסף שממנו ייבנה war המכיל את כל ה jar'ים הנדרשים (ובפרט ה Jar של מנוע המשחק).

מודול זה מכיל גם את קוד ה UI (HTML, CSS, JS) וגם את קוד הסרבלטים המקשרים למודול המנוע.

## איך מתחילים ?

התחילו בעבודה על פי סדר העמודים. ראשית התחילו בעמוד הלוגין. הוא יכול ואמור להיות זהה בעיקרו לעמוד הלוגין כפי שהוצג בדוגמא המסכמת (הרגישו חופשי "להעתיק" משם את המידע הנדרש לכם. מתנה ממני אליכם.)

תו"כ כתיבת עמוד זה תתרגלו את שלב ה UI ל שרת וחזרה ל UI. זה גם ידרוש מכם להקים בצד השרת אובייקט כזה או אחר שמנהל את המשתמשים וכן הלאה.

לאחר מכן התחילו לעבוד על עמוד מס' 2. ביחרו פיצ'ר נוסף וממשו אותו בצורה מלאה מה UI עבור לסרבלט המתאים בשרת וחזרה ל UI שבו אתם מייצרים את ה HTML בהתאם לתשובה שחזרה מהשרת. ממליץ מאוד (כמו ב JAVA FX) לייצר עמוד HTML סטטי שמכיל "סקיצה" ראשונית של המידעים שאתם רוצים להציג כך שתוכלו להבין איך הוא אמור להיראות, מי הרכיבים שבו, איך לעצב אותו מבחינת CSS וכן הלאה. זה גם ייתן לכם הבנה ראשונית של מהם כל קריאות ה HTTP שאתם עתידים לייצר מכל רכיב ורכיב, באיזו תדירות הם יוצאים ? איזה סרבלטים משרתים אותם (האם סרבלט אחד לכולם או כמה "סרבלטונים" קטנים ?) וכן הלאה. אחרי שיש לכם סקיצה – תוכלו ב JS להתחיל לייצר את התוכן ה HTML של כל אזור ואזור ולעבוד על רכיב ורכיב "לעומק" (UI -> SERVER -> UI).

כך מסיימים את עמוד 2 ואז עוברים באותו אופן לעמוד 3. אחרי שכל התשתית והשלד הראשי של האפליקציה בנוי – מוסיפים את כל הכפתורים שמאפשרי את המעבר בין עמוד לעמוד (מעמוד 3 לחזור לעמוד 2, מעמוד 2 לעשות logout ולחזור לעמוד 1 וכן הלאה...

## בונוסים

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | סוג | מהות | למה שווה לי ? | כמה שווה לי ? |
| 1 | הגדלת ראש מגניבה ! | Chat:  אפשרו לשחקנים לקיים chat בינהם תו"כ התחברותם למע'.  ה chat צריך להתנהל כולו אל מול השרת כמובן בצורה שבה כולם רואים את מה שכולם כותבים. (מאוווווווווווד דומה לדוגמא הסופית)  הצ'ט מתבצע בין השחקנים הרשומים והמחוברים למערכת בלבד. | היש מדהים מזה ??? | 10 נקודות  (ועד ל 100) |
| 2 | הגדלת ראש מגניבה | תצוגת מבנה repository בצרוה מסודרת ונורמלית  ממשו יכולת לצפות בתוכן הקבצים של repository בצורה של עץ ספריות וקבצים המופיע בשמאל וברגע שלוחצים על קובץ – אפשר לראות את תוכנו מימין (ושם גם לערוך, לשמור וכו') | כדי שאפשר יהיה באמת להנות משימוש במע'. | **10 נקודות**  **(ועד ל 100)** |
| 3 | בקטנה | מחיקת branch:  אפשרו את היכולת למחוק branch מעמוד מס' 3. יש לתמוך ביכולת למחוק branch רגיל (שאינו RTB).  במידה ובוחרים למחוק branch שהוא RTB יש לבצע את המחיקה all the way – כלומר גם את ה RB המתאים וגם את ה branch המתאים אצל RR. (אפשר להניח כי בשלב זה הbranch שאנו מוחקים לא יהיה ה HEAD אצל RR)  משתמש ה RR צריך לקבל על כך התראה כשהוא במסך 2 או 3 (שאומרת שנמחק לו branch) ובוודאי שהמידע צריך להתשקף אצלו (כשיסתכל על רשימת ה branch'ים שיש לו ב repository המדובר...) | איך אומרים ססטיק ובן אל ? קללללללל....... | **7 נקודות**  **(ועד ל 100)** |
| 4 | הגדלת ראש ראויה | סרטטו את מבנה הקומיטים בצורה של עץ,  העץ צריך לכלול גם תצוגה של ה branch'ים שיש במע' ועל מי הם מצביעים (כמו גם מיהו ה head branch וכו').  בקיצור, כפי שהיה בתרגיל 2... אפשר להשתמש חופשי בכל ספריה צד שלישי שתעזור לכם בסרטוט של עץ גרפי ב HTML ו CSS... | התמודדות מעניינת והבנה טובה הרבה יותר של יכולות HTML ו CSS | 15 נקודות  **(מעל ל 100)** |
| 5 | מקצוענות לשמה ! | אגריגרציה של סך השינויים ב PR  כשמשתמש בוחן את תוכן השינויים ב PR מסויים, במקום להראות לו את השינויים עבור כל קומיט וקומיט שב PR, אפשרו לו לראות את סך כל הקבצים שהשתנו (כך או אחרת) על פני כל הקומיטים (כולל קבצים שנמחקו, נוצרו, התעדכנו כמה פעמים וכן הלאה). | מעט לוגיקה... עבודת UI הרבה יותר קלה.  ככה עובדים מקצוענים ! | 10 נקודות  **(מעל ל 100)** |
| 6 | הגדלת ראש סופר דופר ! | ממשו את שכבת ה UI של תרגיל 3 כחלק מאפליקציית Java FX. אפשר להסתמך על האפליקציה הקיימת שכבר מימשתם בתרגיל 2. בבונוס זה יש לאפשר לשחקן לבצע רישום עם שם למערכת, להציג לו את התוכן של מסך 2 (מסך כל ה repositories שלו, כמו גם מסך כל המשתמשים במע' וה repositories שלהם). בלחיצה על כניסה ל repository של המשתמש יש להציג את תוכנה כפי שכבר מימשתם בתרגיל 2 !. יש לאפשר למשתמש לייצר PR ולקבל גם מידע על ה PR'ים הקיימים וכן הלאה. עליכם לבחור ולהשתמש כאן ב HTTP Client שיפעל מול אותו השרת, כפי שקורה בתרגיל 3 (למעשה בונוס זה ייבדק כך שחלק מהמשתמשים הם ב browser וחלקם ב java fx.)  שימו לב:  אין צורך לתמוך בכך שמשתמש מ Java FX יעלה קובץ repository משל עצמו. | [Oh My God !!!](https://www.youtube.com/watch?v=iMs9feeSknk) | 20 נקודות **(מעל ל 100)** |

## סבבה, סיימתי. מה ואיך להגיש ?

יש להגיש קובץ zip המכיל:

1. WAR אחד בדיוק שיושם בספריית tomcat ויעבור deployment אוטומטי. על WAR זה לכלול את כל התלויות שלכם,

אין להניח כי יסופקו לכם תלויות מבחוץ (למשל Gson וכו').

1. קבצי קוד המקור של הפרויקט שלכם (גם צד השרת וגם צד הלקוח).
2. קובץ ההגשה יכיל גם קובץ readme שיכיל הסבר על המערכת, בחירותיכם השונות במקומות שבהם היו לכם בחירה, כמו גם כל דבר נוסף העולה על דעתכם שחשוב שהבודקת תדע.

**בפרט חייב להכיל קובץ ה readme את ה url שאיתו מנווטים אל מסך ה login של המשחק שלכם.**

1. יש לכלול בקובץ ה **readme** גם תיעוד והסבר כללי (וממצה) של המחלקות העיקריות החדשות ותפקידם. (חישבו מה יסייע לבודקת להיכנס ביתר קלות לקוד שלכם ולהבין מי נגד מי...)
2. במידה ומימשתם את מי מהבונוסים לעיל, ציינו זאת בקובץ ה readme כדי שהבודקת תדע לבדוק בהתאם.

בונוס שימומש אבל לא יתועד – לא ייבדק !

הגשה באיחור, שאינה באישור, תבטל כל מימוש בונוס. אין להגיש באיחור בשביל להספיק לעשות

בונוסים. תכננו את הזמן בהתאם.

## שאלות ותשובות

|  |  |  |  |
| --- | --- | --- | --- |
| # | תאריך | שאלה | תשובה |
| 1 | 18.9.19 | האם יש להציג התראות גם בעמוד 3 ? | כן בהחלט. את אותן ההתראות שמציגים בעמוד מס' 2. כלומר כל עוד המשתמש מחובר למע' – הוא מקבל התראות. |
| 2 | 18.9.19 | האם בהתחברות מחדש צריך להביא את כל ההתראות שוב או רק את החדשות שהצטברו ? | במידה ומשתמש יוצא וחוזר למע' הוא צריך להמשיך לקבל את ההתראות שהצטברו עבורו מאז הפעם האחרונה שהוא התעדכן בהן... |
| 3 | 20.9.19 | האם אפשר לדחוף כל branch ל RR ? | כן. במהלך הדחיפה ה branch חייב כמובן להיות ה HEAD BRANCH. |
| 4 | 9.10.19 | האם צריך להראות את כל השינויים הפתוחים שנצברו ב WC ? | בוודאי. בדומה לתרגיל 1 ו 2, על המשתמש להיות מסוגל לראות את מצב הקבצים הנוכחי (כלומר את WC). כחלק ממסך 3, כשעוברים למסך שמציגים בו את היכולת לעבוד על WC, צריכים להיות מוצגים כל הקבצים בצורתם הנוכחית (קבצי הקומיט המוצבע ע"י HEAD בתוספת אי אילו שינויים שנצברו וטרם קומטו).  מעודכן עכשיו גם בגוף התרגיל (עמוד 37) |
| 5 | 11.10.19 | האם צריך להראות בעמוד ה WC גם את כל סוגי השינויים הפתוחים שנצברו ? | כן בוודאי, בדיוק כמו שמומש בתרגיל 1 ו 2. יש להציג את סך השינויים הפתוחים לפי קבצים שהתעדכנו, נוצרו, נמחקו... אפשר כמובן לתת חיווי ליד כל קובץ או ב 3 רשימות נפרדות – איך שתמצאו לנכון.. (מעודכן עכשיו גם בעמוד 37 בפרק ה WC...) |

נספח א' – תיאור מצב של repository עם מספר קומיטים בה.

a.txt

Root folder

At the beginning, the repository had 2 fiels

1. ./a.txt
2. ./fol1/Foo.java

First commit was created (1) and there were two branches pointing to it

1. master (HEAD)
2. test

Foo.java

fol1

Root folder

a.txt

fol1

Foo.java

psvm {

s.o.p("… tao tao")

}

master

**3**

**4**

**3**

After changing file **./fol1/Foo.java** content we created one more commit (2)

Active branch is master

test

**2**

**1**

**2**

**1**

Foo.java

fol1

psvm {

s.o.p("hello world")

}

Foo.java

a.txt

fol1

a.txt content

נספח ב' – תיאור מצב של repository לפני/אחרי מיזוג בין 2 branch'ים

F2 repository state

a.txt (updated)

b.txt

fol1/c.txt (updated)

D2 repository state

a.txt (updated)

b.txt

b2.txt

fol2/d.txt

B repository state

a.txt

b.txt

fol1/c.txt

fol2/d.txt

Defects

Feature

אנו נרצה למזג (== למרג'ג') את הענף defects לתוך הענף feature.

ראשית עלינו לחשב מיהו האב הקדמון המשותף הקרוב ביותר. קל לראות בדוגמא זו כי זהו קומיט B.

נעבור ספרייה אחרי ספרייה ונשווה את התוכן של D2 ו F2. במקומות שבהם יש דילמה נשווה גם למצב באב הקדמון (B):

1. A.txt שונה בשניהם (קונפליקט)
2. B.txt שווה בשניהם
3. B2.txt קובץ שמופיע רק ב D2. הוא אינו מופיע ב B ולכן הוא קובץ חדש שנוצר ב D2
4. Fol1/c.txt נמחק ב D2, קיים ב B וגרסתו שונה ב F2 (קונפליקט)
5. Fol2/d.txt היה ב B, נותר קיים ב D2 אולם נמחק ב F2 – כלומר יישאר מחוק בקומיט התוצאה.

מרשימת השינויים הנ"ל, #2 ו #3 ו #5 אינם קונפליקטים. שניהם יישארו בקומיט המיזוג.

על המשתמש להחליט כיצד הוא פותר את הקונפליקטים #1, #4,

קונפליקט #1

המשתמש יראה את התוכן של a.txt כפי שהוא היה ב 3 המצבים השונים (B, D2, F2) ויקבע איך הוא יראה בסוף המיזוג. לגרסה הסופית נקרא a'.txt

קונפליקט #4

נניח כי המשתמש מחליט לשמר את הגרסה של הקובץ המעודכן ב F2

אם כך אחרי השלמת תהליך המיזוג בין 2 הענפים, זו תהיה מפת הקומיטים ותוכן ה repository:

Defects

D2 + F2 repository state

a'.txt (updated both)

b.txt (untouched)

b2.txt (new file from D2)

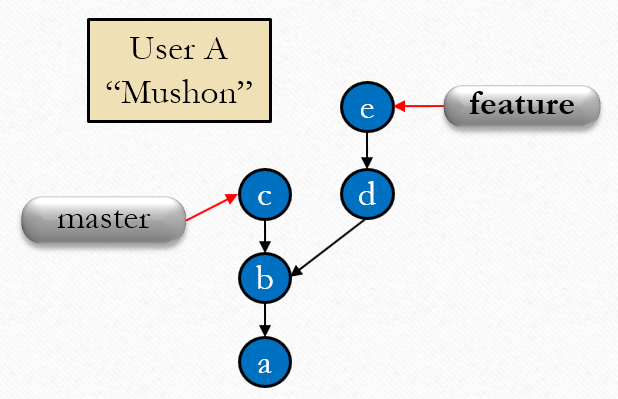
fol1/c.txt (updated at F2)

Feature

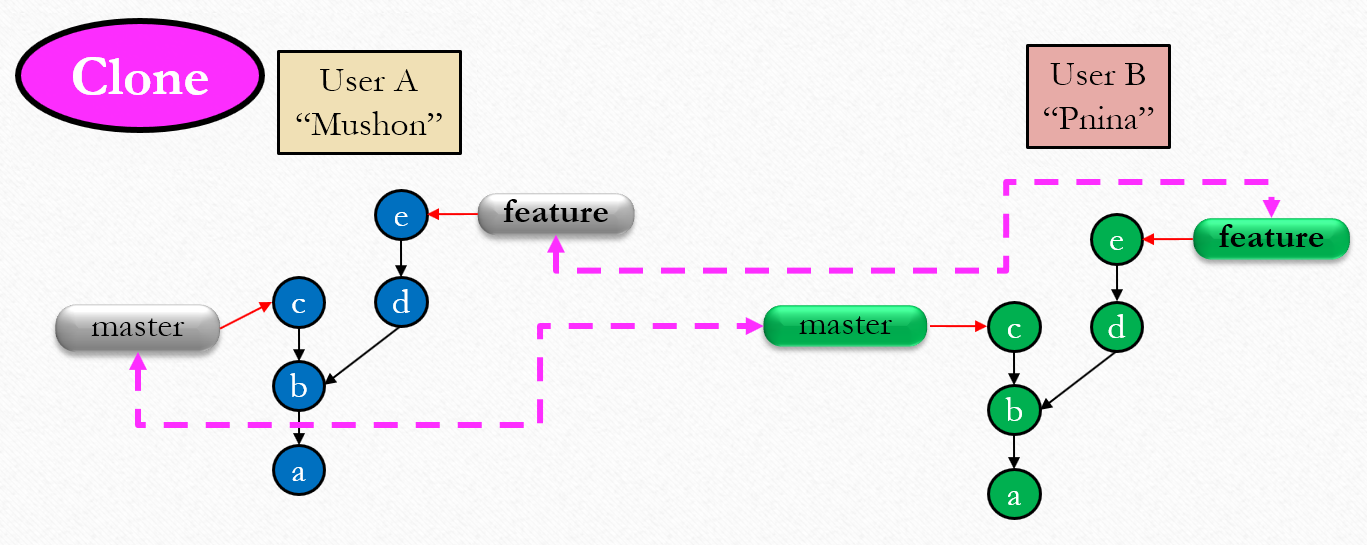
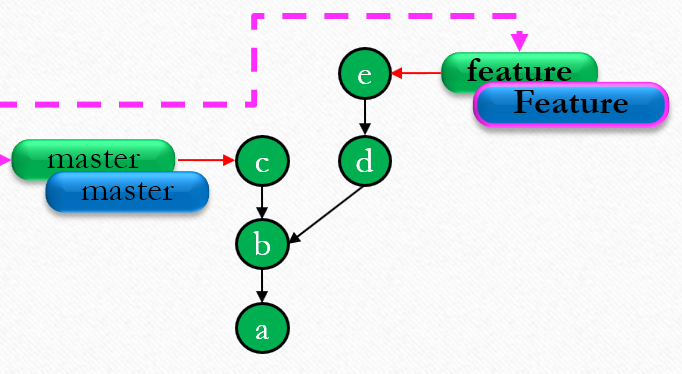
שימו לב:

* 1. הקומיט החדש יש 2 אבות, היות והוא הכלאה בין 2 קומיטים שונים.
  2. רק ה Head branch מקודם כדי להצביע על הקומיט החדש.

נספח ג' – תיאור שיתוף פעולה עם repository אחר (clone, fetch, pull, push)

נניח כי בשלב מסויים יש לנו repository שנקרא Mushon. זהו מצבו:

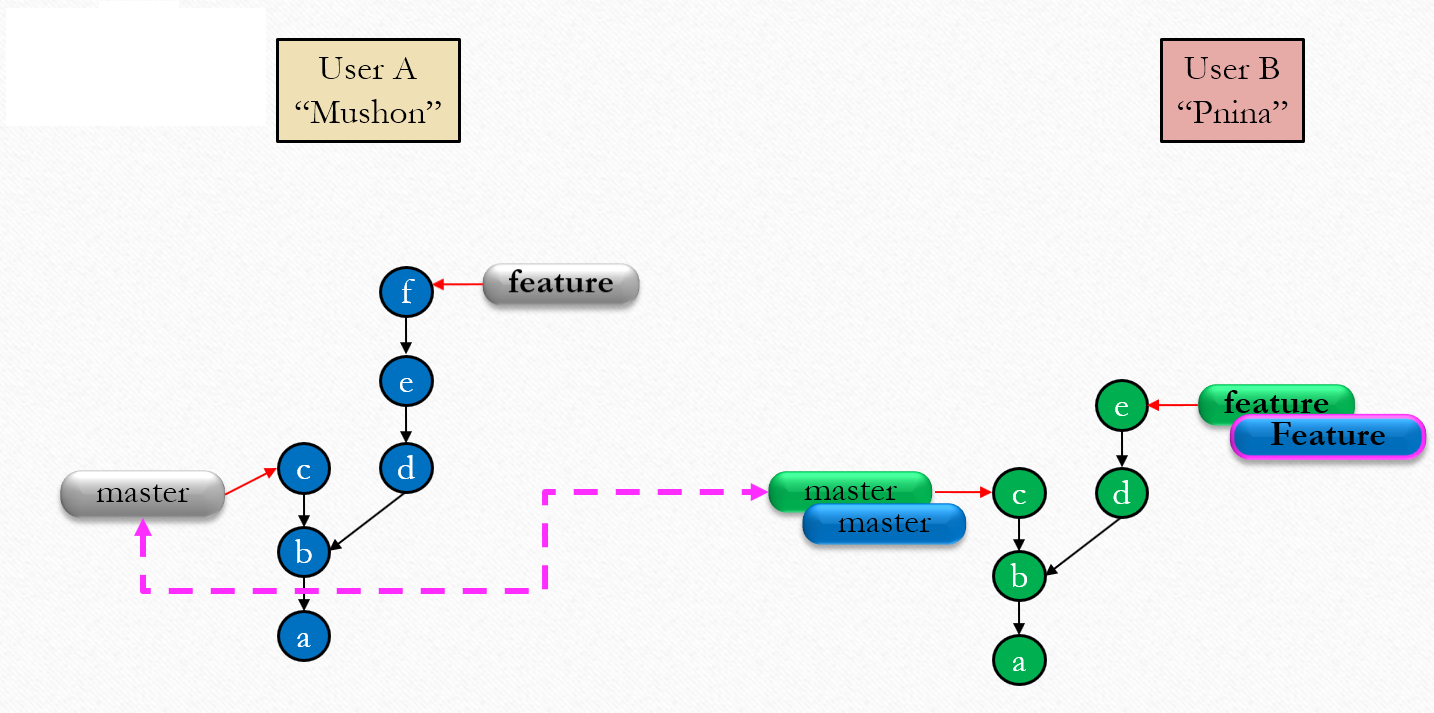
אנו מעוניינים לשכפל repository זה ל repository חדש שנקרא pnina

לאחר ביצוע פעולת clone נקבל את מצב הדברים הבא:

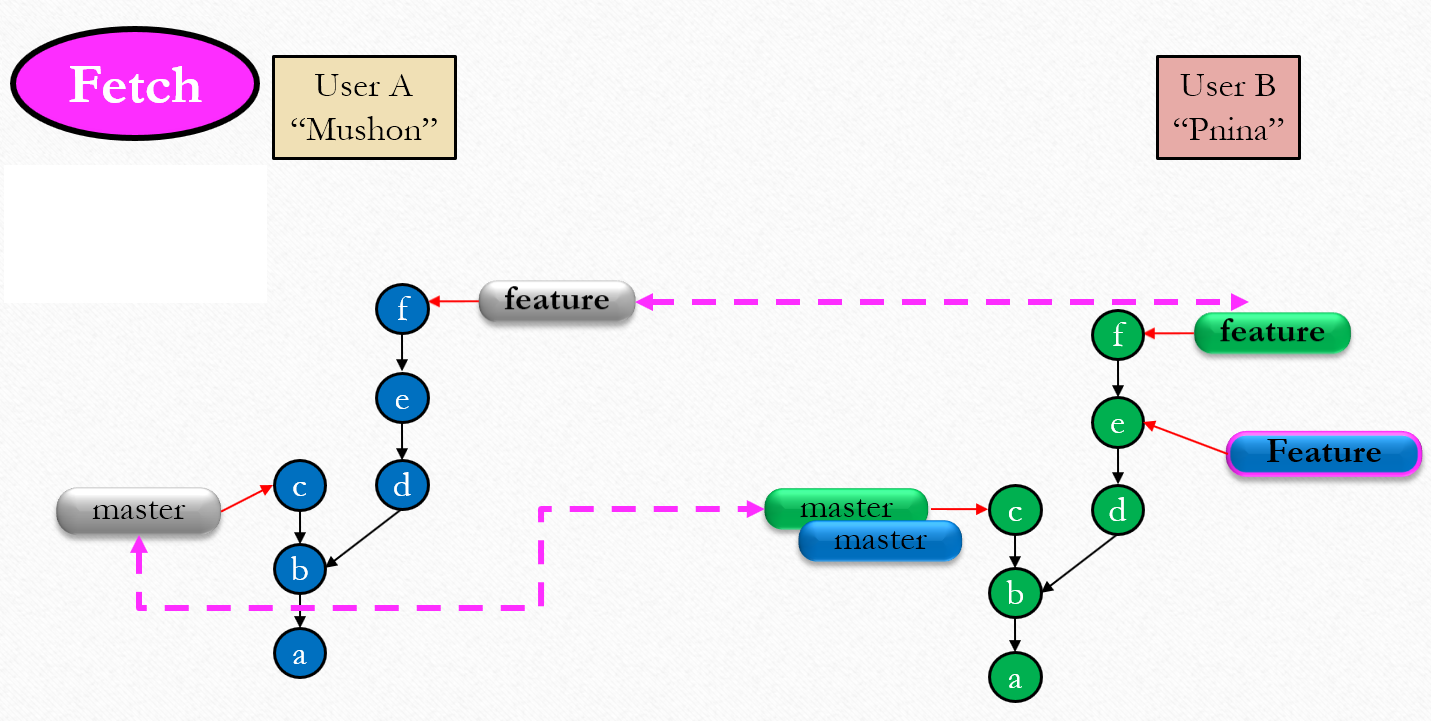
מספר דגשים:

* כל הקומיטים ו 2 ה branch'ים מ Mushon הועתקו גם ל Pnina.
* Feature ו master אצל פנינה (הצבועים בצבע ירוק) הם למעשה remote branch המתעדים את המיקום של ה branch'ים התואמים להם ב remote repository (== Mushon) בפעם האחרונה שבה בוצע עדכון. (כרגע, זה במעמד ה clone עצמו). Remote branch לא יכול להיבחר להיות head branch.
* Feature ו master (הצבועים בצבע כחול) הם למעשה remote tracking branch. אם פנינה רוצה לעבוד על remote branch היא חייבת לבחור לעבוד על ה remote tracking branch המתאים במקום.

נניח כי מושון ביצע עוד עבודה בצד שלו על branch feature:

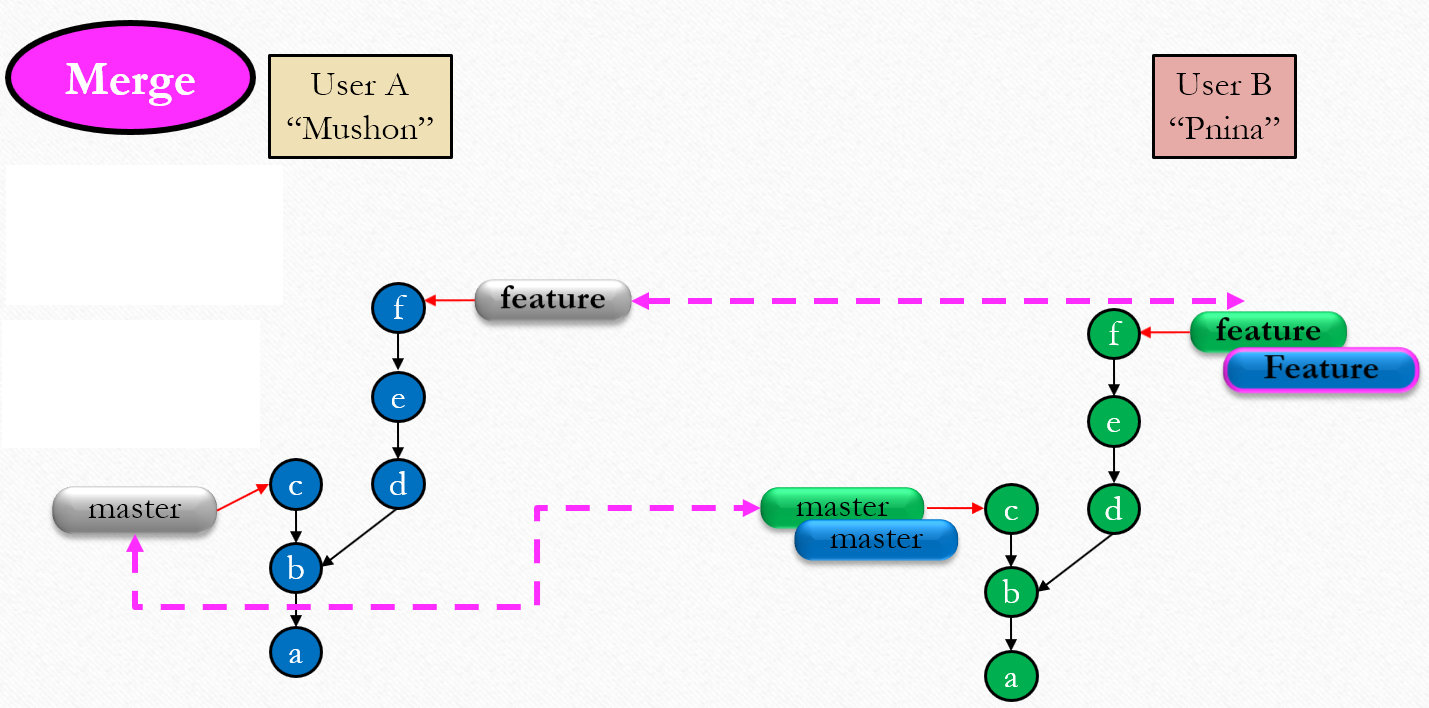
שימו לב כי עכשיו feature אצל פנינה (הירוק) כבר אינו מסונכרן עם feature אצל מושון, והוא נותר כעדות למיקום ולפוזיציה בפעם האחרונה שפנינה התעדכנה ב feature.

פנינה רוצה למשוך את כל העדכונים שקרו אצל מושון אליה ל repository. היא מפעילה את הפקודה fetch:



בזכות ה remote branch'ים של פנינה, היא ידעה לאמר מה הפוזיציה האחרונה שהיא ראתה אצל מושון. בהתאם לכך חושבו כל דלתאות הקומיטים (וכל שאר האובייקטים הנגזרים מהם) שהם הדברים החדשים אצל מושון והובאו אחר כבוד חזרה לפנינה. אחרי פעולה זו ה feature remote branch (הירוק) אצל פנינה מתעדכן להצביע על הקומיט האחרון כפי שהוא אצל מושון ועכשיו ה repositories שוב מסונכרנות. יש לציין כי כל המידעים הנוספים אצל מושון (branch'ים, קומיטים, אובייקטים) היו גם הם מיובאים אל פנינה כחלק מפקודת ה fetch. במידת הצורך עוד remote branches היו נפתחים אצל פנינה כדי לתעד את המיקומים שלהם אצל מושון.

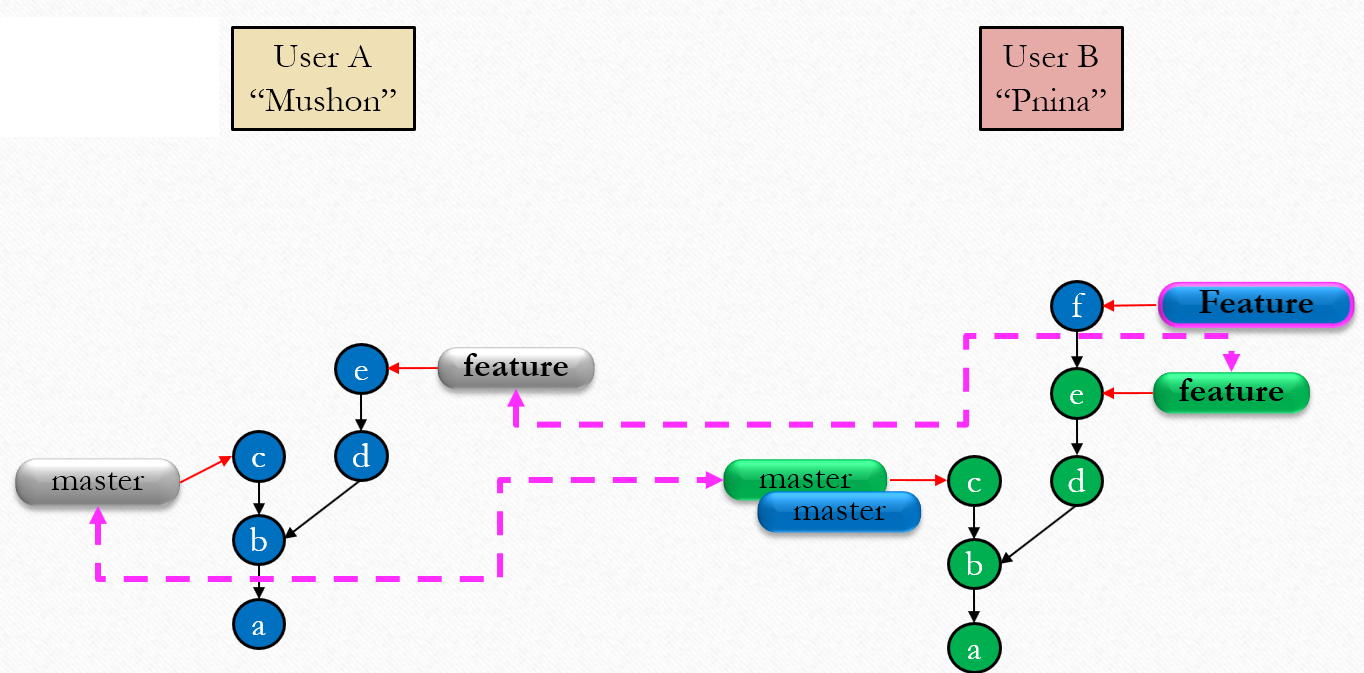
שימו לב כי ferature הכחול (ה remote tracking branch אחרי feature) היה ונותר במקומו. מה שאפשר לעשות עכשיו זה לאפס את feature הכחול כך שיצביע על feature הירוק כך ששניהם יצביעו לאותו הקומיט (f). ועכשיו פעולת העדכון הושלמה.



(פעולת האיפוס מקבילה גם לפעולת merge (מיזוג))

קיצור דרך לפעולה זו (fetch + merge\reset) היא פעולת Pull אשר בניגוד ל fetch מתבצעת עבור remote tracking branch מסויים שחייב להיות ה head branch. פעולה זו תבצע את התוכן הנ"ל עבור ה branch מסויים בלבד. תביא את התוכן שהשתנה רק בהקשרו ותסתיים כך שגם ה remote branch וגם remote tracking branch שניהם מיושרים למידע המתאים אצל מושון.

נניח כי הפעם דווקא פנינה היא זו שהייתה מייצרת תוכן חדש ורוצה לדחוף אותו למושון.

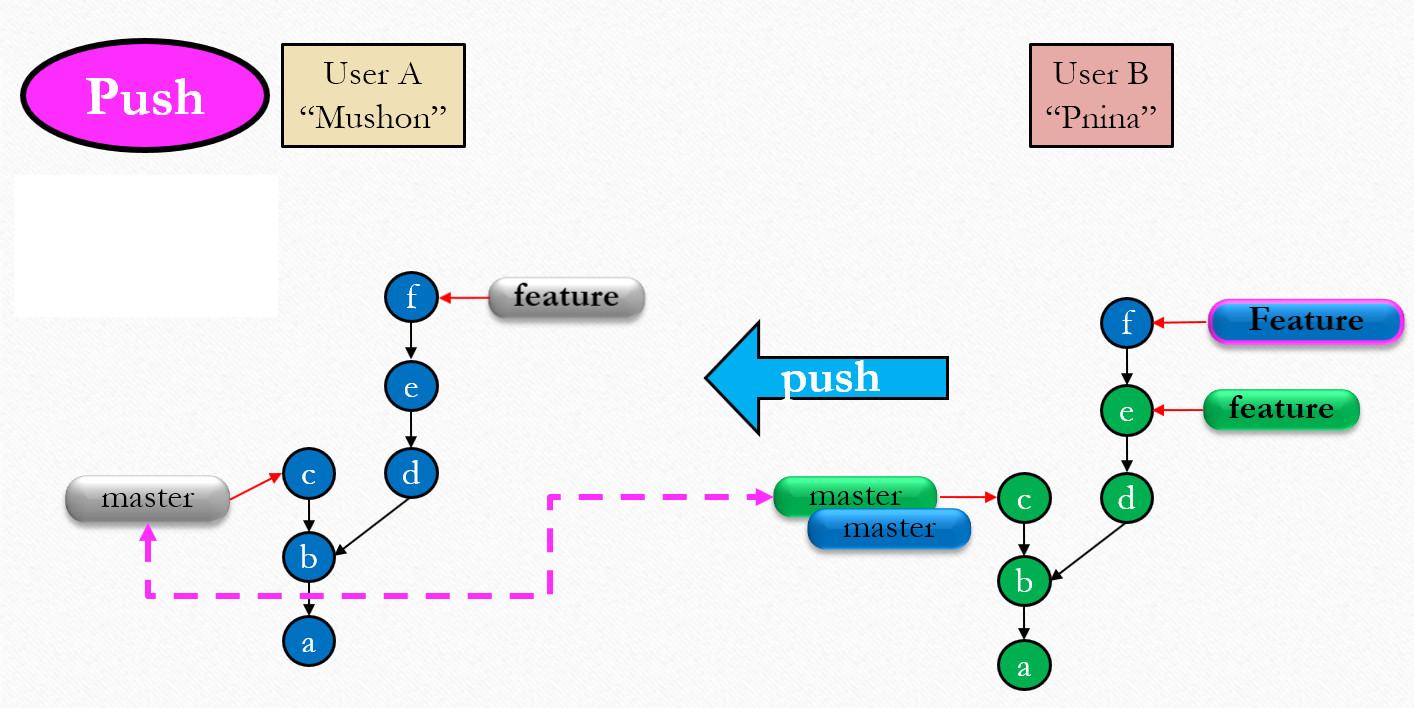


ניתן לראות כי feature הכחול "התרחק" מ feature הירוק (שהיה ונותר עודנו מראה מקום למצב המתאים אצל מושון).

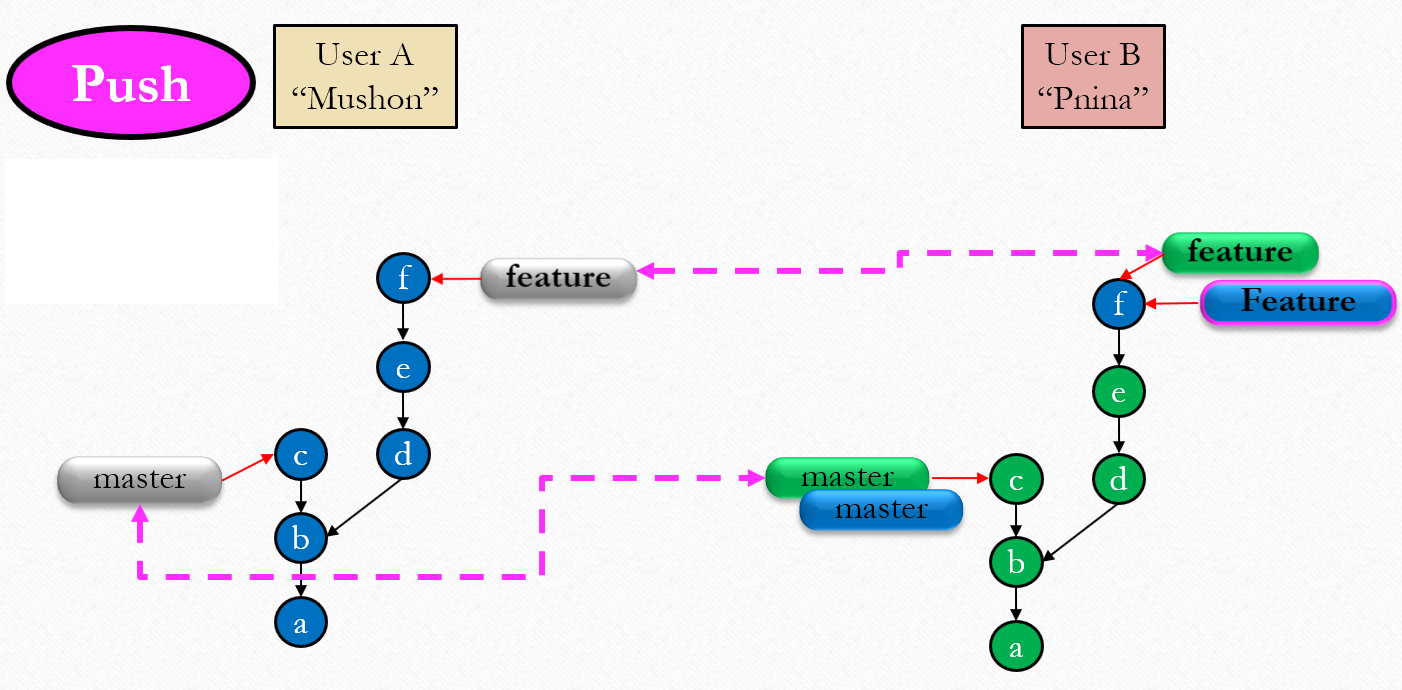
במצב זה כשפנינה תרצה לדחוף את המידע שהצטבר על feature הכחול למושון – היא תפעיל את הפקודה Push.

פקודה זו פועלת על branch אחד בלבד, אשר חייב להיות ה head branch וחייב להיות remote tracking branch

במהלך פקודה זו תחושב הדלתא אצל פנינה לדחיפה: סך הקומיטים והשינויים שבין feature הכחול ל feature הירוק (הקומיט f) והיא תועבר למושון תוך עדכון ה feature אצל מושון:



עכשיו יש לעדכן (לאפס) את feature הירוק אצל פנינה להצביע גם כן על קומיט f שכן זהו מצב הדברים הנוכחי גם אצל מושון, ובכך תמה הפעולה:

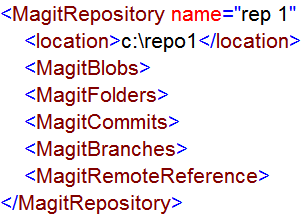


נספח ד' – תיאור מבנה המערכת באמצעות קובץ XML

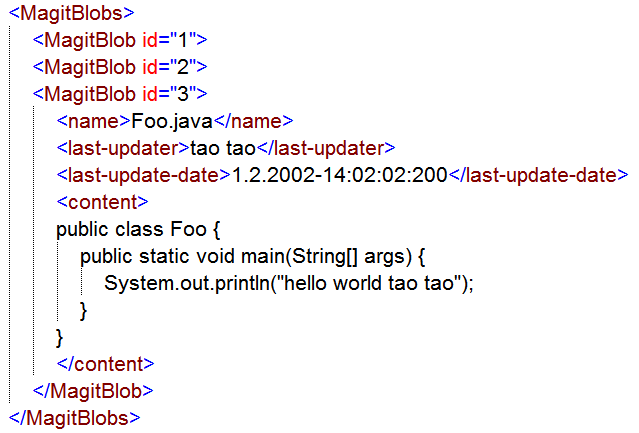
תיאור מבנה המערכת מפורט כחלק מקובץ ה XML

במהלך הבדיקה (של שלל התרגילים), תיבדק המערכת באמצעות מספר קבצים שונים, חלקם חוקיים וחלקם תקולים, במטרה לראות האם וכיצד המערכת מגיבה לשגיאות.

בחנו היטיב את קבצי הדוגמא שהועלו למע' ה Mama וודאו כי אתם מבינים את פרטיהם ומבניהם.

הקובץ מתאר מבנה של repository בודד:

|  |  |  |  |
| --- | --- | --- | --- |
| # | סוג | שם | מהות |
| 1 | Element | MagitRepository | האלמנט הראשי המכיל את כל המידע לגבי תוכן ה repository |
| 2 | Attribute | name | שם של ה repository |
| 3 | Element | location | המיקום במע' הקבצים של הספרייה המתארת את ה repository. בתוך ספרייה זו תשב ספריית .magit |

 **Blobs**

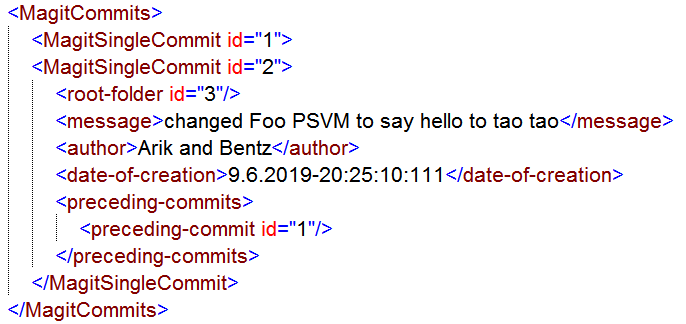
|  |  |  |  |
| --- | --- | --- | --- |
| # | סוג | שם | מהות |
| 1 | Element | MagitBlobs | אלמנט המכיל מידע כל הקבצים במע'. אלמנט זה יכול להיות ריק במקרה של repository ריק. |
| 2 | Element | MagitBlob | אלמנט המתאר קובץ אחד במע' |
| 3 | Attribute | id | מזהה חד חד ערכי של ה blob. מסוג String. |
| 4 | Element | name | שם הקובץ |
| 5 | Element | last-updater | שם המשתמש האחרון שביצע שינוי בקובץ |
| 6 | Element | last-update-date | התאריך האחרון בו בוצע שינוי בקובץ |
| 7 | Element | Content | תוכנו של הקובץ |

**Folders:**

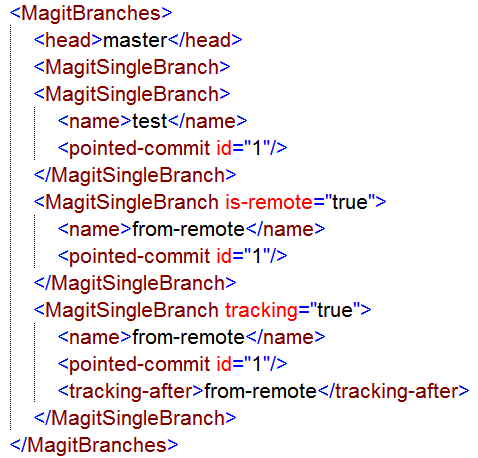


|  |  |  |  |
| --- | --- | --- | --- |
| # | סוג | שם | מהות |
| 1 | Element | MagitFolders | אלמנט ראשי המכיל את כל המידע לגבי כל הספריות במערכת. יכול להיות ריק מתוכן (ב repository ריק) |
| 2 | Element | MagitSingleFolder | אלמנט המתאר ספרייה אחת במערכת |
| 3 | Attribute | id | מזהה חד חד ערכי של ספריה. מסוג string |
| 4 | Attribute | is-root | Boolean – מתאר האם הספריה המתוארת היא הספרייה הראשית של המע' או לא. ברירת מחדל: false |
| 5 | Element | name | אופציונלי. חייב להופיע עבור ספרייה שאיננה root – ואז מתאר את שם הספרייה |
| 6 | Element | last-updater | שם המשתמש האחרון שביצע שינוי בספרייה (במי מהפרטים שבה) |
| 7 | Element | last-update-date | התאריך האחרון בו בוצע שינוי בספרייה (במי מהפרטים שבה) |
| 8 | Element | item | מתאר פריט אחד המוחזק ע"י הספרייה. חייב להיות לפחות אבר אחד כזה בכל ספרייה. |
| 9 | Attribute | type | מתאר את סוג הפריט המוחזק בספרייה.  ערכים אפשריים: blob ; folder |
| 10 | Attribute | id | אם ה type הוא blob מצביע למזהה של blob ; אם ה type הוא folder מצביע למזהה של ספרייה. |

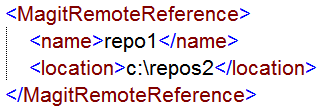
**Commits:**



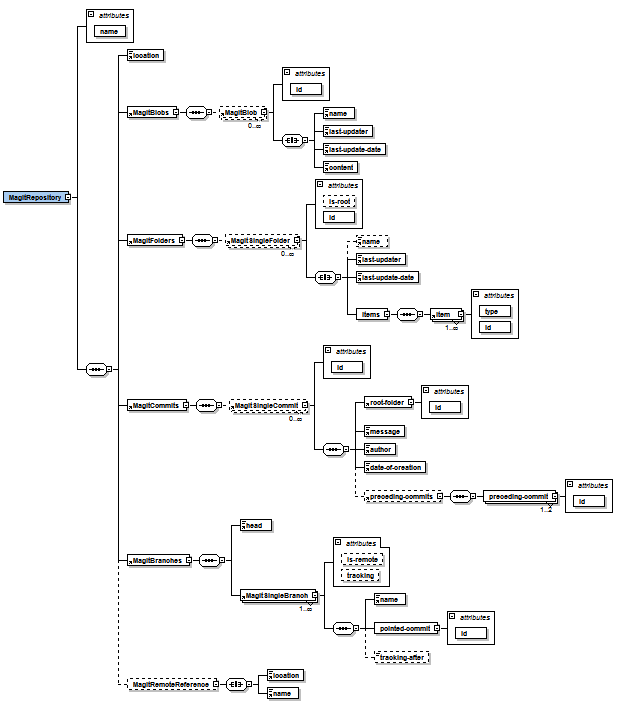
|  |  |  |  |
| --- | --- | --- | --- |
| # | סוג | שם | מהות |
| 1 | Element | MagitCommits | אלמנט ראשי המתאר את כל הקומיטים במע'. יכול להיות ריק מתוכן (ב repository ריק) |
| 2 | Element | MagitSingleCommit | אלמנט המתאר קומיט אחד במערכת |
| 3 | Attribute | id | מזהה חד חד ערכי של הקומיט הנוכחי. מסוג string |
| 4 | Element | root-folder | אלמנט המתאר מיהי הספרייה הראשית המוצבעת ע"י קומיט זה |
| 5 | Attribute | id | מזהה של אלמנט מהספריות שחייב להיות is-root=true |
| 6 | Element | message | הודעת הקומיט |
| 7 | Element | author | מי שיצר את הקומיט |
| 8 | Element | date-of-creation | תאריך יצירת הקומיט |
| 9 | Element | preceding-commits | אלמנט המתאר את כל הקומיטים שקדמו לקומיט זה. אלמנט זה נחשב אופציונלי ויכול לא להופיע רק עבור הקומיט הראשון במע' |
| 10 | Element | preceding-commit | אלמנט המתאר קומיט שקדם לקומיט זה. יכולים להופיע בדיוק אחד או שניים כאלה עבור כל קומיט במע' |
| 11 | Attribute | Id | מזהה חד חד ערכי של קומיט המתואר במע' |

**Branches:**

|  |  |  |  |
| --- | --- | --- | --- |
| # | סוג | שם | מהות |
| 1 | Element | MagitBranches | אלמנט כללי המתאר את כל ה branches הקיימים במע' |
| 2 | Element | head | אלמנט המתאר את השם של ה branch הפעיל. |
| 3 | Element | MagitSingleBranch | מתאר branch בודד במע' |
| 4 | Element | name | מתאר שם של branch. מסוג string |
| 5 | Element | pointed-commit | מתאר את ההפנייה לקומיט המוצבע ע"י branch זה |
| 6 | Attribute | id | מזהה לקומיט שמוגדר ב commits |
| 7 | Attribute | is-remote | Boolean: מתאר האם branch זה מהווה ייצוג של remote branch.  ערך ברירת מחדל: false |
| 8 | Attribute | tracking | Boolean: מתאר האם branch זה "עוקב" אחרי remote branch אחר (RTB)  ערך ברירת מחדל: false |
| 9 | Element | tracking-after | מזהה את שם ה branch אחריו branch זה עוקב. אלמנט זה אופציונלי ויופיע רק ב branch שבו tracking=true |

**Remote:**

|  |  |  |  |
| --- | --- | --- | --- |
| # | סוג | שם | מהות |
| 1 | Element | remote | אלמנט המתאר את הנתונים לגבי ה remote repository ש repository זה עובד איתו. |
| 2 | Element | name | שמה הלוגי של ה repository המרוחק. יופיע בראשית השמות של כל ה remote branches |
| 3 | Element | location | מיקומה של ה remote repository |

נספח ה' – קובץ סכמה לוידוא קובץ ה XML

קובץ ה XML מקושר לסכמה שמגדירה את חלקיו השונים ואת חוקיותם.

מומלץ להשתמש בסכמה כדי לוודא כי הגדרתם את הקובץ נכונה. (הסכמה הועלתה למערכת ה Mama).

להלן תרשים המתאר את מבנה הסכמה:

נספח ו' – מידע על ספריות צד שמושיות לצד ג'

תוכלו להשתמש בתרגיל בספריות צד ג' אשר יעזרו לכם במטלות השונות אותם תצטרכו לפתח.

הרשימה מטה מהווה המלצה בלבד ואינכם חייבים להשתמש בכולה או בחלקה.

כך או אחרת עליכם לדאוג כי כל jar שתשתמשו בו יוגש גם כן כחלק מההגשה שלכם ויישב בצורה שה class loader יוכל למצוא אותו

הספריות הנ"ל הועלו כולם לאתר הקורס תחת חומרי התרגיל וזמינות לשימושכם.

* + 1. **commons-codec**

ספריה זו שימושית בעיקר לעבודה עם sha1 בשורה אחת פשוטה.

היא מכילה מחלקה בשם DigestUtils שבאמצעותה ניתן להפעיל מגוון אלגוריתמי קידוד והצפנה שונים.

הנה דוגמא (input הוא משתנה מסוג String; הפלט הוא 40 תווים אקסהדצימליים...):

DigestUtils.*sha1Hex*(input)

* + 1. **commons-io**

ספריה זו שימושית לכל הקשור בהתעסקות עם מע' הקבצים. פעולות כמו יצירת ספרייה, יצירת קובץ, קריאת קובץ, מחיקת ספרייה/קובץ וכו'.

המחלקה העקרית השימושית בה בהקשר זה היא המחלקה FileUtils.

* + 1. **commons-collections4**

ספרייה זו שימושית בכל הקשור בהתעסקות בפעולות "מתקדמות" יותר על collections, פעולות כגון:

* חיסור collections
* איחוד collections
* חיתוך בין 2 collections

הפעולות הנ"ל מאוד מסייעות בכל המקומות בהם אתם צריכים להשוות בין 2 קומיטים או 2 repositories שאחרי ככלות הכל ימומשו בדמות אסופה של collections...

המחלקה המרכזית בה לשימוש בהקשר זה היא CollectionUtils. (על אף שיש בתוכה עוד פרטים מעניינים רבים...)

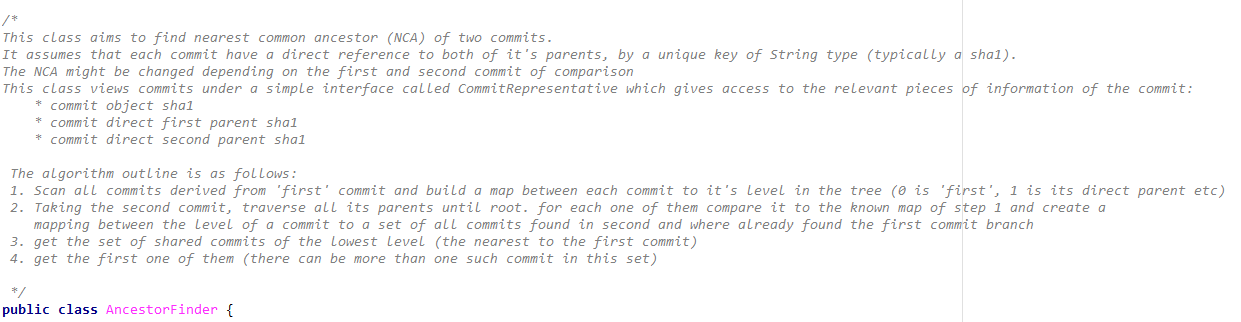
1. **AncestorFinder**

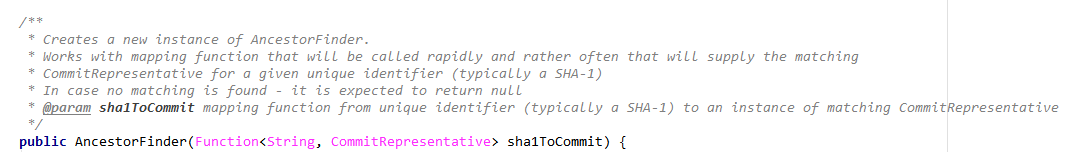
לטובת ביצוע תהליך המיזוג (merge) בין 2 branch'ים יש למצוא אב קדמון משותף ל 2 ה branch'ים שרוצים למזג בינהם. יש מספר אלגוריתמים לזה, אולם לכולם יש מקרי קיצון שונים והתנהגות שונה. זהות האב הקדמון יכולה לעיתים להשפיע בצורה נחרצת על תוצאת ה merge ולכן יש חשיבות רבה לכך שכולכם תזהו את אותו האב הקדמון עבור אותה מפת branch'ים.

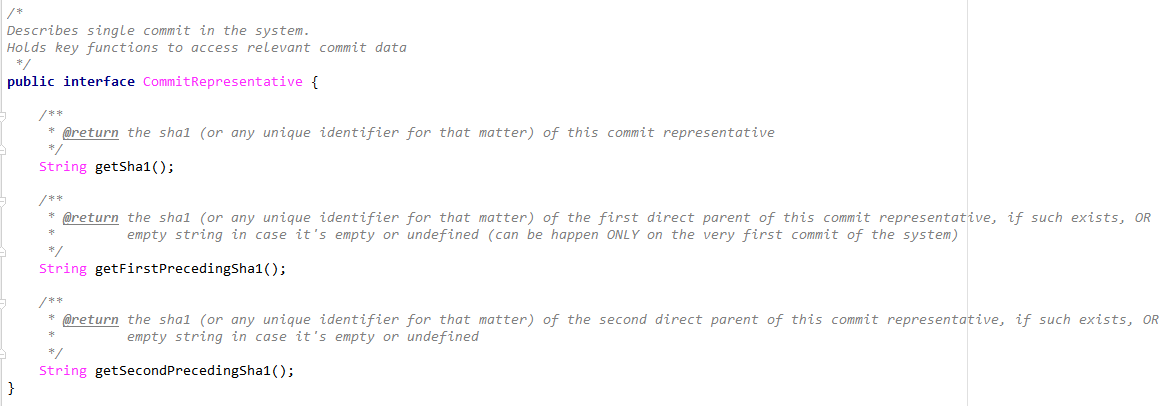
כדי לפשט את התהליך מסופקת לכם ספרייה צד שלישי שנקראת **AncestorFinder** שתפקידה למצוא אב קדמון משותף קרוב ביותר, **ואתם מחוייבים להשתמש בה לטובת תהליך המיזוג**

היות ולכולכם יש מימוש שונה של קומיטים והיחסים בינהם (להלן עץ הקומיטים), בבואכם לעבוד עם הספרייה תצטרכו "ליישר קו" עם דרישותיה הגנריות כדי לקבל תוצאות אחידות. "יישור קו" זה בא לידי ביטוי בדמות ממשק (interface) שהספרייה עובדת איתו (CommitRepresentative) אשר תצטרכו לספק לספריה כחלק מהעבודה איתה.

כדי להשתמש בספרייה תצטרכו לקרוא את התיעוד שלה ולהבין את האופן שבו היא עובדת:



הנה תיעוד ה constructor:

הנה תיעוד הממשק CommitRepresentative איתו עובדת המחלקה:

אחרי שתאתחלו את מופע הספריה (AncestorFinder) תוך מתן פונקציית המיפוי, תקראו למטודה האחת ויחידה traceAncestor:

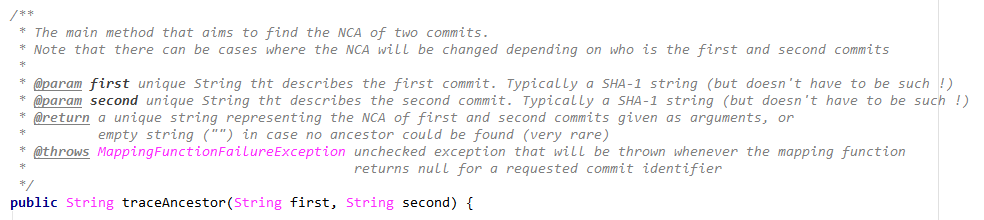


בקריאה הנ"ל המטודה מוצאת את האב הקדמון המשותף הקרוב ביותר בין 2 קומיטים המזוהים כ "A" ו "B". הפלט של המטודה הוא המזהה הייחודי של הקומיט שהוא האב הקדמון ל 2 הקומיטים הנ"ל.

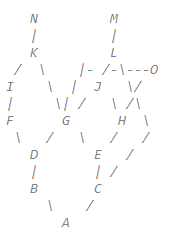
(שימו לב כי כל המזהים הייחודיים יגיעו בסוף מפונקציית המיפוי שתספקו לספרייה ומטבע הדברים הם

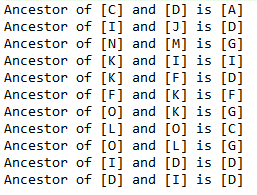
יהיו SHA-1'ים של הקומיטים השונים אצלכם במע'.)

הנה התיעוד של הפונקציה traceAncestor:



כדי להדגים את אופן פעולת הספרייה על עץ קומיטים נרחב תוכלו להסתכל על הדוגמא הבאה:



ועל תוצאות השונות של בדיקת ה NCA עבור מי מהקומיטים הנ"ל: