הערכה חלופית סמסטר ב' תשפ"ה

כזכור ההערכה החלופית מורכבת מתרגילים 1-3 במערכת הבדיקות (פרויקט גילוי חריגות) בהם ניתן לכם גם מוד אימון חשוף.

לעומת זאת בתרגיל הנוכחי (4) הצפייה היא שאתם תכתבו בדיקות לעצמכם בטרם אתם מגישים את הקוד למערכת הבדיקות. לכן בעת הגשה לא תתבצע כל בדיקה, גם לא קומפילציה, רק תקבלו הודעה שהגשתם את הקבצים בהצלחה (ותתעלמו מהציון כמובן). הבדיקה תתבצע offline לאחר מועד ההגשה. תרגיל זה מהווה 50% מהציון הסופי.

בהצלחה!

דרישה 1 – הגדרת טיפוס משתמש (User)

במסגרת הפרויקט עליכם להגדיר טיפוס נתונים המייצג משתמש במערכת.

הדרישות לטיפוס המשתמש הן:

- לכל משתמש יש את השדות הבאים:
  - מזהה ייחודי (id) מסוג Int
  - שם מלא (name) מסוג String
  - כתובת דוא"ל (email) מסוג String
  - תפקיד במערכת (role) מסוג String
- הטיפוס חייב להיות מוגדר כך ש:
  - ניתן יהיה ליצור מופעים חדשים שלו בצורה פשוטה וברורה.
  - השוואה בין מופעים תתבצע לפי הערכים של השדות (ולא לפי מיקום בזיכרון).
  - ניתן יהיה להדפיס מופע לפורמט קריא כברירת מחדל (למשל לצורך דיבוג).
  - יש תמיכה בפונקציית העתקה (copy) ליצירת מופעים חדשים עם שינויים חלקיים בשדות.

רמז:
סקאלה מספקת דרך פשוטה וסטנדרטית להגדרת טיפוס עם התכונות הללו. חשוב להשתמש בכלי המתאים כדי לעמוד בדרישות מבלי לכתוב קוד מיותר.

ממשו זאת בקובץ User.scala

דרישה 2 – טיפוס UserSystem לניהול אוסף משתמשים אימיוטבילי

עליכם להגדיר טיפוס נתונים בשם UserSystem (בקובץ UserSystem.scala) שמייצג אוסף של משתמשים.

הטיפוס חייב להיות **אימיוטבילי** – עליכם להבין בעצמכם מהן ההשלכות לכך בקוד.

פונקציות נדרשות למימוש (ראו את שמות הפונקציות בהמשך)

1. הוספת משתמש חדש
2. מחיקת משתמש לפי מזהה
3. עדכון משתמש
4. חיפוש משתמש לפי מזהה
5. החזרת כל המשתמשים כמפה בין זהות למשתמש

מותר לכם להוסיף פונקציות נוספות כראות עיניכם.

דגשים פונקציונליים

- השתמשו במבני נתונים אימיוטביליים מבוססי Scala סטנדרטיים.
- שמרו על טוהר פונקציונלי – אל תשתמשו ב־ var או בשינויים במצב פנימי.

נקודות בדיקה אוטומטית אפשריות

- בדיקה שהפונקציה addUser לא מוסיפה משתמש עם מזהה שכבר קיים.
- בדיקה ש removeUser מוחקת משתמש לפי מזהה ומחזירה עותק חדש.
- בדיקה ש updateUser משנה רק את המשתמש עם המזהה הקיים, ושאר המערכת נשארת ללא שינוי.
- בדיקה ש findUser מחזירה Some(user) אם המשתמש קיים ו None-אחרת.

מלבד הדרישות לעיל, הטיפוס צריך להיות עקבי עם הדוגמאות הבאות:

```
val sys1 = new UserSystem()
val sys2 = sys1.addUser(User(1, "Alice", "alice@example.com", "admin"))
val sys3 = sys2.addUser(User(2, "Bob", "bob@example.com", "user"))
val sys4 = sys3.updateUser(User(2, "Bobby", "bob@example.com", "user"))
val maybeUser = sys4.findUser(2)  // Some(User(2, "Bobby", ...))
val sys5 = sys4.removeUser(1)
if (sys3==sys2) …
val users=sys4.getAllUsers // Map[Int,User]
```

דרישה 3 – מימוש איחוד אימיוטבילי של אוספי משתמשים באמצעות מונואיד (Monoid)

עליכם לממש איחוד (combine) בין שני מופעי UserSystem ליצירת מופע חדש, המכיל את כל המשתמשים משני האוספים.

- הפעולה חייבת להיות **אימוטבילית.**
- במקרה של התנגשות במזהה משתמש בין שני האוספים a1,a2 , יש לשמור את המשתמש מהאוסף השני a2.
- מימוש פונקציונליות האיחוד יתבצע באמצעות **object נפרד,** בשם UserSystemMonoid, אשר יממש שתי פונקציות:
  - empty - ערך ניטרלי לאיחוד (אוסף ריק).
  - combine - פונקציה שמקבלת שני מופעים של UserSystem ומחזירה מופע מאוחד.
- **האובייקט לעיל צריך לממש ממשק גנרי בשם Monoid שעליכם לכתוב גם כן.**
- עליכם להקפיד שהמימוש יעמוד **בשני החוקים המרכזיים של מונואיד**!

את המימוש יש לבצע בקובץ UserSystemMonoid.scala

שאלה (פתוחה)

א. מדוע עדיף לממש את פונקציונליות האיחוד (המונואיד) באמצעות object נפרד ולא באמצעות ירושה או הרחבה של UserSystem?
ב. מהם שני החוקים של מונואיד?
ג. כיצד חוקים מתמטיים אלו עוזרים לנו בעולם פיתוח התוכנה?

ענו על השאלות הפתוחות באנגלית כהערות בסוף הקוד.

דרישה 4 Pipeline – גנרי לעיבוד נתונים (Processor[A])

ממשו טיפוס כללי ואימיוטבילי בשם Processor[A] המאפשר עיבוד בר־שרשור של אוספים, תוך שמירה על טיפוסיות לאורך השרשרת.

API נדרש:

- filter(p: A => Boolean): Processor[A] מסנן ומחזיר מופע חדש.
- map[B](f: A => B): Processor[B] ממפה לטיפוס חדש ומחזיר מופע חדש.
- reduce(f: (A, A) => A): A מצמצם לערך יחיד מטיפוס A.
- toList(): List[A] מחזיר את הפריטים במצב הנוכחי של השרשרת.

בנוסף, יש לספק נקודת כניסה סטנדרטית לבניית המעבד:

- object Processor { def fromIterable[A](it: Iterable[A]): Processor[A] }

חשוב לבדיקה: הבדיקה תאתחל את המעבד תמיד באמצעות Processor.fromIterable(sys.getAllUsers.values)

דגשים פונקציונליים

- הטיפוס חייב להיות **אימיוטבילי** (אין var, אין שינוי מצב פנימי; כל פעולה מחזירה מופע חדש)
- השרשור חייב לעבוד **בכל סדר פעולות** שהמשתמש יבחר.
- יש לשמור על **טיפוסיות נכונה**: אחרי map(User => X) למשל, המשך השרשור מתנהל על X
- אין לדרוש ייצוג פנימי מסוים, הבדיקה ניגשת רק דרך ה־ API לעיל.

דוגמא

```
val p = Processor.fromIterable(sys.getAllUsers.values)
val adminsUpper = p.filter(_.role == "admin").map(u => u.copy(name =
u.name.toUpperCase)).toList
val totalIdSum  = p.map(_.id).reduce(_ + _)
```

בהצלחה!