מפה

מבוא כללי

טיפוס הנתונים מפה (טיפוס נתונים מופשט)

כפי שצוין בפרק, המערך הוא למעשה מפה המקשרת בין מפתחות שהם האינדקסים במערך לבין הערכים השמורים בתאי המערך. מגבלות הגודל של המערך נסקרו בעבר. אנו מעוניינים להרחיב את מושג המפה, לשמור ולטפל בכל אוסף גנרי של מפתחות וערכים כך שיישמר המיפוי שבין מפתח (כלשהו) לבין ערך, בלי הגבלת כמות. לאחר הצגת הבעיה באופן כללי בפרק, אנו מתמקדים במפתחות מטיפוס מחרוזתי בלבד.

לאחר שנגדיר טיפוס נתונים מופשט שכזה, השומר על אוספים גנריים - דינמיים ומאפשר טיפול נוח ויעיל בהם, נוכל להשתמש בטיפוס הנתונים מפה לצורך פתרון בעיות נוספות (ראו את התרגילים שבסוף הפרק).

מטרת הפרק

הפרק מדגים את הרעיון של טיפול באוסף שהוא ממטרות היחידה, ומהווה תרגול של כל מה שנלמד ביחידה עד לכאן: הגדרת טיפוס נתונים מופשט, הפרדה מוחלטת בין ממשק למימוש, שימוש באוסף אחד לצורך הייצוג והטיפול באוספים קונקרטיים, הגדרת טיפוסי נתונים (כתיבת מחלקות), שיקולי יעילות ועוד. חשוב לבצע את התרגיל במלואו, באופן מבוקר, בנוכחות המורה ובהנחייתו. כך יתאפשר לכם לוודא שהחומר מובן לתלמידים ולחזור ולהסביר קטעים בעיתיים תוך כדי חשיפתם במהלך התרגול.

הפרק מציג רובד ראשוני של טיפוס האוסף החדש ועולה בשלב התרגילים לרמות הפשטה והבנה גבוהות יותר. כל מורה יחליט עד כמה הוא מעוניין להרחיב ולהתעמק בשלבים אלו (בעיקר השלבים המתקדמים של תרגיל המילון). בכל זאת כדאי לציין שתרגיל המפה מסמן את כיוון החשיבה והניתוח הנדרשים ביחידת הלימוד ומהווה משום כך סמן ומציין לרמת ההבנה הנדרשת בתום לימוד היחידה, ולכן לא כדאי לחסוך מדי בזמן המוקדש לתרגולו.

הצעה למהלך הוראה

1. הגדרת טיפוס הנתונים מפה כטיפוס המאפשר טיפול באוסף של קשרי מפתח-ערך.
2. הגדרת הפעולות הדרושות לטיפוס מפה. כדאי להגיע באופן מובנה ומונחה לאוסף הפעולות המוצגות בממשק ולא להציג את הממשק הקיים, כדי שהתלמידים יבינו את הנחיצות והמשמעות של כל אחת מהפעולות.

   הפעולה **getValue** מניחה שהמפתח קיים במפה. נזכיר כי כאשר אנו מציינים **הנחה** בממשק אנו אומרים למעשה לכותב המחלקה שהוא אינו צריך לבדוק מצב זה בתוך מימוש הפעולה, ומספיק שציין ב"חוזה השימוש" (תיעוד הפעולה) את ההנחה הזו. מאידך על המשתמש

במחלקה לוודא שהוא אכן לא מזמן את הפעולה עבור מפתח שאינו קיים ולכן עליו להיעזר בפעולה exists ולוודא שהמפתח קיים קודם הפנייה אליו.

נוסיף לגבי הפעולה **getValue** שיכול להיות שערך החזרה של מפתח מסוים יהיה **null** אך אין בכך שום הבדל משמעותי מכל ערך החזרה אחר שהוא.

3. הגנריות של המפתח: במסגרת חומר הלימוד של היחידה, K לא יכול להיות גנרי. אך יש לדעת שמנגנון הממשקים הקיים בשפות, יכול היה לאפשר ל-K להיות גנרי ובר-השוואה, אם המחלקה Map הייתה מממשת את הממשק Comparable.

public class Map<K **extends** Compareble, V>

4. העלאת רעיונות שונים לייצוג המפה (שני מערכים, שתי רשימות. תוכניות המדגימות את הפתרונות הללו מצורפות לנוחיותכם אך חבל להתעמק בהן מעבר להעלאת הרעיון וחשיבה קצרה עליו). המטרה היא להגיע לחידוש של מפה כ**רשימה של זוגות**!
5. החלטה על ייצוג המפה כרשימה של **זוגות**: מפתח-ערך. הבנה שיש צורך להגדיר את טיפוס הנתונים זוג, ולשלבו במימוש המפה מבלי שהממשק של המפה ישתנה.

ממשק טיפוס הנתונים זוג מצורף. שימו לב שאין מניעה להגדיר את Pair כגנרי בשני ערכיו מכיוון שאף פעולה אינה נעשית על ערכי המפתח והערך. בזמן השימוש של זוג בתוך מפה יוגדר המפתח כמחרוזת.

ניתן לעורר דיון בכיתה שבעקבותיו ינוסח ממשק המחלקה או לחלק אותו לתלמידים. לאחר הגדרת הממשק יש לבקש מהתלמידים לבצע את חלק א של המשימה: כתיבת המחלקה Pair, הצעת ייצוג של האוסף כרשימה של זוגות והתחלת מימוש הפעולות של Map כבסיס לדיון שיתקיים בסעיף ו על הפעולה הפרטית search.

שימו לב כי מרגע שעלה רעיון הזוגות Pair ממשיך הפרק ומתייחס לייצוג המפה בעזרת רשימה של זוגות, וכי למרות שהמחלקה Pair מוגדרת כגנרית בשני ערכיה, הרי שמסעיף ו אנו מדברים על זוגות שהמפתח שלהם מחרוזתי.

ממשק המחלקה Pair <K, V>

המחלקה מגדירה זוגות של נתונים: מפתח-ערך.

Pair (K key, V value)	הפעולה בונה זוג על פי הפרמטרים
K getKey()	הפעולה מחזירה את המפתח של הזוג הנוכחי
void setKey (K key)	הפעולה משנה את ערכו של מפתח הזוג הנוכחי על פי הפרמטר
V getValue()	הפעולה מחזירה את הערך השמור בזוג הנוכחי
void setValue (V value)	הפעולה משנה את הערך השמור בזוג הנוכחי על פי הפרמטר
String toString()	הפעולה מחזירה מחרוזת המתארת את הזוג

כתיבת המחלקה Pair על ידי התלמידים. המימוש מצורף.

```
public class Pair<K,V>
{
	private K key;
	private V value;

	public Pair (K key, V value)
	{
		this.key = key;
		this.value = value;
	}

	public K getKey()
	{
		return this.key;
	}

	public void setKey (K key)
	{
		this.key = key;
	}

	public V getValue()
	{
		return this.value;
```

```
    }

    public void setValue (V value)
    {
        this.value = value;
    }
}
```

6. בשלב זה כדאי להעלות את השאלה הבאה: האם כדאי לשמור את הנתונים במפה ממוינים או לא? כפי שמצויין בממשק, הפעולה **getAllKeys** מחזירה מערך ממוין של המפתחות אך הדבר אינו מצביע בהכרח על כך שקיים סדר כלשהו בשמירת הנתונים במפה עצמה!
נפרט את מערכת שיקולי היעילות שצריכה להידון במסגרת הניסיון לענות על שאלה זו, ונעלה את הנקודות המשמעותיות למהלך הדיון:
אם השמירה של האוסף במפה תהיה בסדר ממוין כל הזמן, תהיה הפעולה **getAllKeys** "זולה" ותתבצע ב-$O(n)$, בעוד שאם המפה לא תשמר ממוינת, עלות הפעולה תהיה $O(n^2)$.
האם יש סיבה כלשהי שלא להוזיל את הפעולה getAllKeys? סיבה שכזו תהיה רק אם נמצא שסיבוכיות זמן הריצה של פעולות אחרות *עולה* בעקבות שמירת המיון התמידי.
נבדוק זאת:
מהי עלות הפעולות המצריכות איתור מפתח ברשימה (insert, remove, exists, getValue) אם הנתונים במפה *אינם מוחזקים באופן ממוין*? כל הפעולות הללו מחפשות את המפתח, והחיפוש מסתיים כאשר נמצאת החוליה המכילה את המפתח או מסתבר שהמפתח לא קיים באוסף. בשלב זה מבצעת כל אחת מהפעולות את מה שנותר לה לעשות. מספר החוליות שסורק חיפוש המסתיים בהצלחה באוסף לא ממוין היא n במקרה הרע ביותר, ובמקרה הממוצע n/2. חיפוש המסתיים בכישלון חייב לסרוק את כל הרשימה.
ומהי היעילות כאשר *האוסף ממוין*? מחיר של חיפוש המסתיים בהצלחה הוא עדיין סריקת n חוליות במקרה הרע ביותר, כי יתכן שהמפתח נמצא בסוף הרשימה, ובממוצע n/2 חוליות. חיפוש המסתיים בכישלון יכול להסתיים מיד כשרואים מפתח שערכו גדול מהמבוקש, ולכן המחיר הממוצע של חיפושים אלו יורד לסריקת n/2 חוליות, במקום n.
הערה: מבחינת סדרי גודל, מדובר ב- $O(n)$ בכל מקרה, כיון ש-n/2 שונה מ- n רק בקבוע.
נסכם ונאמר: סיבוכיות הפעולות המושפעות מאופן החזקת הנתונים (לעומת הפעולות שאינן קשורות ומושפעות מכך: toString והפעולה הבונה), אינה משתנה באופן משמעותי אם שומרים על מיון קבוע של הנתונים במפה, ומאידך המיון החד פעמי של הנתונים עבור הפעולה getAllKeys יקר.

נציין עוד שכאשר נגיע לפרק הדן בעץ חיפוש בינרי נוכל להדגים ייצוג אחר של המפה, שבו השמירה של הערכים באופן ממוין, *משפרת* את יעילות הפעולות.

7. הפעולה הפרטית **search**. הצגת הצורך בפעולה מתייחסת כבר לייצוג המפה כרשימה של זוגות ולכן הפעולה מחפשת חוליה ברשימה ומחזירה הפנייה אליה.
כותרת הפעולה צריכה להיות:

private Pair<K, V> search (K key)

מכיוון שהפעולה פרטית, אין בעיה בכך שערך החזרה שלה הוא Pair ש"חושף" את אופן הייצוג של המחלקה. החשיפה הזו נשארת בתוך מימוש המחלקה עצמה.

לגבי ערך החזרה של הפעולה, הפרק מזכיר את הדרישה שהוא יהיה הפניה הקודמת לזו שבה המפתח, כדי לממש את remove באופן יעיל. דרישה דומה קיימת בגלל הפעולה insert. אם כאשר המפתח אינו במפה מוחזר **null**, אזי הפעולה תצטרך לחזור על החיפוש. אם במקום זאת תוחזר הפניה לחוליה האחרונה שבה יש ערך מפתח הקטן מהמפתח הנתון (עם טיפול מתאים למפתח חדש הקטן מכל המפתחות שבמפה), אזי insert תתבצע ב- O(1) צעדים.

מכיוון שהפעולה מוגדרת כפעולה פנימית המשרתת את שאר הפעולות להגדירה באופן היעיל ביותר שיקצר את משך ביצוע פעולות הממשק.

8. לאחר הגדרת Pair נחזור למחלקה Map ונכתוב את התכונה היחידה שלה, המשמשת לאחסון אוסף הזוגות:

```
public class Map<V>
{
        private List <Pair<String ,V>> map;
}
```

יש להקצות לתלמידים זמן מספיק לכתיבת כל המחלקה Map על כל פעולותיה, כולל תיעוד מספק ממנו יוכלו לייצר קבצי API.

9. שימוש במחלקה Map. שאלת הבחירות (כוכב נולד) מדגימה שימוש קלסי וברור במחלקה Map. יש לדון עם התלמידים בבעיה ולקרוא איתם ביחד את הפתרון המופיע בפרק באופן מונחה ומוסבר. לאחר הצגת הבעיה יש לקיים דיון ולהבטיח שגם אם עלה הרעיון לפתור את הבעיה בעזרת שני מערכים (שמות וקולות) לא זה יהיה הפתרון המועדף. לפנינו שאלה המציגה מפתח (שם הזמר) וערך הקשור למפתח זה (מניין הקולות) ולכן הפתרון הנדרש הוא בעזרת מפה. כאשר יעברו התלמידים על הפתרון שבפרק יראו בפעם הראשונה שימוש מעשי במפה. אם ברצונכם להוסיף לתרגול הטיפוס גם את השימוש בפעולה remove ניתן לבקש מהתלמידים שיסירו מהמפה את כל הזמרים שקבלו 0 קולות.
10. מומלץ מאד לפתור את כל השאלות המצורפות לפרק. הנחיות מיוחדות ראו בהמשך.

תרגילים

1. מימוש הפעולה ספור-מפתחות:

הוספת הפעולה לממשק פירושה כמובן כפעולה פנימית. כדי שיעילותה תהיה O(1) ברור שצריכה להיות תכונה של מפה שתחזיק את מספר המפתחות הקיימים במפה.

private int numOfKeys;

ומימוש הפעולה יראה כך:

```
public int numOfKeys()
{
    return this.numOfKeys;
}
```

אם לא זה היה הייצוג אזי ברור שיש לשנות את מימוש הפעולות:

Map(); insert(…); remove (…)

כך שהתכונה תחזיק כל הזמן מספר מעודכן של הזוגות. במקרה כזה לא תשתפר יעילות הפעולות האחרות אך במובן ידוע הפעולות שיצטרכו לעדכן את מספר הזוגות יהפכו מעט יותר "כבדות".

2. דף עבודה 1 – **ספר טלפונים**:

פעולות ה-get המחזירות ערכים שונים הקשורים לאיש הקשר מניחות את קיומו בספר הטלפונים. כפי שציינו לגבי הנחות ("חוזה") אין צורך שכותב המחלקה יממש בדיקה לקיום איש הקשר בתוך מימוש הפעולה, אך על המשתמש החיצוני המזמן פעולות אלו, לבדוק שאיש הקשר אכן קיים קודם זימון הפעולות. לשם כך קיימת הפעולה exists. הפעולה המוחקת איש קשר מהספר לא מכילה הנחה בדבר קיומו של איש הקשר מכיוון שאינה מחזירה ערך. אם איש הקשר לא קיים הפעולה תסתיים באופן תקין מבלי לבצע דבר.

שימו לב שפעולת המחיקה של איש קשר מספר הטלפונים, אינה מחזירה כל ערך ולכן לא צריך להגדיר מה יקרה אם איש הקשר לא מופיע בספר. הפעולה תסתיים באופן תקין מבלי שתבצע דבר.

לנוחיותכם מצורפת תוכנית בדיקה המאפשרת בדיקה מהירה ועניינית של ספר הטלפונים שכתבו התלמידים.

3. דף עבודה 2 – **המילון**:

הערה: בכל מה שקשור לשאלה זו מילים המתחילות באות גדולה זהות לחלוטין למילים המתחילות באותה אות אך קטנה. לצורך זה הזכירו לתלמידים שהמחלקה String מכילה פעולות מתאימות בהן הם יכולים להשתמש.

רקע חשוב:

המילון מדגים יישום המטפל באוסף של קשרי מפתח-ערך. המילון משמש אותנו בחיי היום יום, ולכן יש לשער שדוגמה שכזו תעניין את התלמידים. שימו לב שהגדרת מילון בו לכל מילה יש תרגום אחד בלבד, היא בדיוק שימוש במפה שהגדרנו. כל פעולה של המילון היא בדיוק פעולה של מפה ולכן יישום כזה הוא "משעמם" לכאורה.

כדי להגדיל את העניין ולהעמיק את התרגול של טיפוסי נתונים, אנו מגדירים את המילון כך שלכל מילה במילון יש כמה תרגומים. התלמיד יצטרך להבין שהערך הגנרי V שהוגדר במפה מצריך בזמן הגדרת המילון, הגדרת טיפוס חדש שהוא אוסף התרגומים של מילה. על נתונים אלו צריכות להיות מוגדרות הפעולות בנייה של האוסף, חיפוש איבר, הכנסה של איבר אם הוא לא קיים עדיין באוסף והחזרה של איבר, כמפורט בהדרכה שבשאלה.

מכיוון שנדרש סדר ידוע בין התרגומים השונים (סדר הוספתם למילון), נראה סביר להשתמש ברשימה השומרת על סדר זה, או להגדיר טיפוס חדש בשם Translations (כדי לשמור את מספר התרגומים לכל מילה יש אולי הגיון בהוספת פעולה פרטית מתאימה או בהגדרת טיפוס חדש).

הערה: אם הנוסח המתייחס לאוסף התרגומים, נראה לכם כמרמז על שימוש בתור, כדאי לשים לב לבעייתיות מסוימת. אמנם ההוספה של תרגומים נעשית בדיוק כפי שנעשית הוספה לתור (בסוף), אך כאשר נצטרך להחזיר את אוסף התרגומים נשלוף אותם בפועל מהתור והוא ייהרס. כדי להמשיך ולשמור על התרגומים נצטרך להכניס לסוף התור כל תרגום שנשלוף מראשו. ניתן אם כן לחשוב על שימוש ברשימה.

ועוד נקודה למחשבה לפני שתיגשו לפתרון: בדקו האם ברור לכם ההבדל היסודי בין מפה למילון? (המפה היא טיפוס נתונים המטפל באוסף כללי. לכן נוכל להיעזר בו לצורך ייצוג אוספים שונים המטפלים בקשרי מפתח-ערך. המילון לעומתו הוא אוסף ספציפי לחלוטין ולא סביר שנוכל להשתמש בו לצורך ייצוג אוספים אחרים).

חלק א: כאמור, כדי להוסיף עניין לשאלה נקבע שלכל מילה יש מספר תרגומים. לכן צריך להגדיר טיפוס נתונים נוסף לשמירת אוסף התרגומים.

ניתן להדגים פתרון לבעיית המילון תוך ייצוגו כ- List<Pair>, כדי שהתלמידים יבינו עד כמה מסורבל יכול להיות המימוש (למעשה הם מגדירים וממשים מחדש את כל הפעולות שמימשו במפה!).

תוספת: פעולת ההוספה של מילה למילון מבצעת אחד משני דברים: אם המילה לא קיימת במילון, היא תוסף יחד עם תרגומה. אם המילה קיימת במילון ותרגומה עדיין לא קיים, יתווסף התרגום לאוסף התרגומים המוחזקים עבור מילה זו.

תרגום המשפט. ניתן להשתמש במחלקה StringTokenizer על פי ה-API שלה או לקלוט את המילים אחת אחר השנייה בעזרת זקיף כלשהו, אך אז האפקט של הכנסת משפט שלם וקבלת תרגומו המיידי, הולכת לאיבוד.

חלק ב של המילון –

סעיף 2: הפעולה, שהיא כמובן פעולה פנימית, אמורה לבדוק קיום תרגום לכל מילה במילון.

סעיף 3: נציין כי הפעולה חייבת להיות מוגדרת כפעולה סטטית ("מקבלת מילון") ולפיכך הכותרת שלה תהיה:

public static List<String>

getSortedWordsFromDictionary (Dictionary dictionary)

מכיוון שלא סביר להגדיר מחלקת שירות שבה פעולה אחת על מילון, ניתן להגדיר את הפעולה בתוך המחלקה הראשית.

לשם הגיוון מחזירה הפעולה רשימה של מחרוזות ולא מערך כפי שהחזירה הפעולה getAllKeys().

שימו לב שבעוד שהמילים במילון מופיעות בסדר ממוין הרי שעל התרגומים יש לבצע מיון לצורך מימוש פעולה זו.