

בחינה בתוכנה 1

סמסטר א תשפ"ב, מועד א', 10 בינואר 2022
לנה דנקין, אמיר הרץ, אלה גולדשמידט

משך הבחינה שלוש שעות.

סך הניקוד על השאלות בבחינה הוא 105, אך הציון המקסימלי אותו ניתן לקבל הוא 100.

יש להניח, אלא אם צויין אחרת, כי:

- הקוד שמופיע במבחן מתאים לגרסה Java8.
- כל החבילות הדרושות יובאו, ואין צורך לכתוב שורות import בגוף הקוד.
- כל מחלקה שהיא public מופיעה בקובץ Java משלה.
- בכל שאלה, כל המחלקות מופיעות באותה חבילה (package).
- בזמן הבחינה, אתם נדרשים לזהות שגיאות קומפילציה שנוצרות כתוצאה מהפרת עקרונות Java-יים ושימוש לא נכון במחלקות/פונקציות. במידה וישנה טעות הקלדה (סוגר חסר, שימוש באות גדולה שלא לצורך וכו') אין לראות בסיבות אלה גורמים לשגיאות קומפילציה.
- בסוף הבחינה מופיע נספח עם תיעוד של מחלקות שאתם עשויים לעשות בהן שימוש בחלק הפתוח של הבחינה.
- הקוד שאתם נדרשים לספק צריך להיות יעיל ולהימנע ממחזור קוד. חלק מהציון ניתן גם על היבטים אלה, ולא רק על נכונות הפתרון.

בבחינה זו מופיע קוד שבחלקו אינו מתקמפל, אינו רץ או שנוגד את הסטנדרטים של Java כפי שנלמדו בקורס, וזאת מתוך מטרה לבחון ידע והבנה של נושאים מסוימים. אין לראות בקטעי קוד אלה דוגמא לכתיבה נכונה ב Java.

מבנה הבחינה:

הבחינה מורכבת משני חלקים: חלק פתוח (שתי שאלות על סך 50 נקודות) ושאלות אמריקאיות (11 שאלות, כל אחת שווה 5 נק'). עליכם לענות על הבחינה באופן הבא:

- בשאלות הפתוחות להשלים את הקוד החסר במקומות המסומנים ע"י מסגרת. שימו לב שלא חייבים למלא את כל המסגרות.
- בשאלות האמריקאיות:
 - לסמן את התשובות הנכונות על גבי טופס סימון התשובות שתקבלו בנפרד.
 - לנמק את תשובתכם על גבי טופס הבחינה. הנימוק הוא לא חובה, אך יכול לעזור לכם במקרים של ערעורים או קבלת יותר מתשובה אחת נכונה.

בסוף שאלה 2 ניתן למצוא מסגרת חירום לשימוש במקרה שהמסגרות שמופיעות בגוף השאלות הפתוחות לא מספיקות לכם.

© כל הזכויות שמורות למחברים. מבלי לפגוע באמור לעיל, אין להעתיק, לצלם, להקליט, לשדר, לאחסן במאגר מידע, בכל דרך שהיא, בין מכנית ובין אלקטרונית או בכל דרך אחרת כל חלק שהוא מטופס הבחינה. בהצלחה!

שאלה 1 (33 נק'):

בשאלה זו נעזור לסטודנטית שי לממש מערכת המתווכת בין אנשים שקנו כרטיסים לתערוכה של יאיוי קוסמה במוזיאון תל אביב ולא יכולים להשתמש בהם, לבין אנשים המחפשים לקנות כרטיסים. הקוד שתממשו בבחינה יעזור לשי לנהל את מלאי הכרטיסים המוצעים למכירה.

בשלב הראשון, שי מימשה את המחלקה Reservation המתארת הזמנה יחידה של כרטיסים על שם לקוחה מסויימת. ההזמנה כוללת מזהה יחודי של בעלת הכרטיסים, מועד הביקור (בעת קניית כרטיס לתערוכה המשתמשים צריכים לבחור את מועד ההגעה) ומספר הכרטיסים. הקוד של Reservation מובע חלקית מטעמי חיסכון במקום והוא כולל בנוי ופונקציות get לשלושת השדות.

```

/* @inv getTicketsNum() > 0 */
public class Reservation{
    private String sellerID;
    private Date date;
    private int ticketsNum;
    public Reservation(String sellerID, Date date, int ticketsNum){
        //code here
    }
    /* getSellerID(), getDate(), getTicketsNum() implemented here */
}

```

הערה – לצורך התרגיל אין צורך להכיר את המחלקה Date המייצגת את מועד הביקור בתערוכה. הניחו כי ניתן לייצר Date מתוך מחרוזת. בנוסף ניתן להניח כי Date ממשת את equals/hashCode את המנשק Comparable.

לאחר מכן, שי הגדירה את המנשק שאמורה לממש מערכת ניהול ההזמנות שלה:

```

public interface ITicketsMaster{

    /* @pre !sellerExist(rsv.getSellerID())*/
    public void addTickets(Reservation rsv);

    /* @pre ticketsNum > 0
    * @post: for sellerID, num in $ret.items():
    *     sellerExists(sellerID) &&
    *     $num <= getTicketsNumForSeller(sellerID)
    * @post: if !$ret.isEmpty(), sum($ret.value()) == ticketsNum
    */
    public Map<String, Integer> findTickets(Date date, int ticketsNum);

    /* @pre sellerExists(sellerID)
    * @post !sellerExists(sellerID) */
    public void removeSeller(String sellerID);

    /* @pre: sellerExists(sellerID)
    * @post: $ret > 0 */
    public int getTicketsNumForSeller(String sellerID);

    public boolean sellerExists(String sellerID);
}

```

השירות findTickets מאפשר למשתמשות לחפש כרטיסים במערכת. במידה שלא נמצאו כרטיסים מתאימים, השירות יחזיר Map ריק. אם נמצאו כרטיסים, השירות יחזיר Map המכיל מופיו בין מזהה המוכרת (sellerID) לבין כמות הכרטיסים שיש לקנות ממנה. השירות לא משנה עושה שום שינוי במלאי כרטיסים הקיימים במערכת (לא מוסיף\מוחק).

מספר זהות: _____ מספר מחברת: _____ עמוד 3 מתוך 20

בשלב הראשון נטפל בניהול מלאי הכרטיסים, ולאחר מכן (סעיפים ב-ד) נממש כמה אסטרטגיות למימוש של `findTickets`.

```
Date d1 = /* initialized with "10-01-2021:09:00" */
Date d2 = /* initialized with "11-01-2021:13:00" */
Reservation r1 = new Reservation("r1", d1, 2);
Reservation r2 = new Reservation("r2", d2, 4);
Reservation r3 = new Reservation("r3", d1, 2);
ITicketsMaster tm = new SimpleTM();
tm.addTickets(r1);
tm.addTickets(r2);
tm.addTickets(r3);
System.out.println(tm.sellerExists("r1")); // true
System.out.println(tm.getTicketsNumForSeller("r3")); //2
tm.removeSeller("r1");
System.out.println(tm.sellerExists("r1")); // false
```

סעיף א' (6 נק'י):

השלימו את מימוש המחלקה האבסטרקטית `AbstractTM` אשר מכילה את הקוד המשותף לניהול מלאי הכרטיסים המוצעים למכירה. המחלקה לא תממש את השירות `findTickets`.

```
public abstract class AbstractTM implements ITicketsMaster{
    protected Map<String, Reservation> sellerToRsv = new HashMap<>();
```

```
    public void addTickets(Reservation rsv){
```

```
        this.sellerToRsv.put(rsv.getSellerID(), rsv);
```

```
    }
```

```
    public void removeSeller(String sellerID){
```

```
        sellerToRsv.remove(sellerID);
```

```
    }
```

```
    public int getTicketsNumForSeller(String sellerID){
```

```
        return sellerToRsv.get(sellerID).getTicketsNum();
```

```
    }
```

```
    public boolean sellerExists(String sellerID){
```

```
        return sellerToRsv.containsKey(sellerID);
```

```
    }
```

```
}
```

סעיף ב' (14 נק'):

בשלב זה שי רוצה לממש מספר אסטרטגיות של חיפוש כרטיסים (ובהתאם, גם מכירה) עבור משתמשת שמעוניינת לקנות k כרטיסים במועד d (המיוצג ע"י Date וכולל יום ושעה). האסטרטגיות הן:

1. התאמה מדוייקת – המערכת תמצא כרטיסים אם קיימת מוכרת אשר מציעה בדיוק k כרטיסים במועד d.
2. שילוב כרטיסים – המערכת תמצא כרטיסים אם קיימות שתי מוכרות אשר מציעות יחד בדיוק k כרטיסים למועד d (כלומר, קיים m כלשהו כך שמוכרת אחת מציעה בדיוק m כרטיסים, והשניה מציעה בדיוק k-m כרטיסים).
3. פיצול כרטיסים. המערכת תמצא כרטיסים למכירה אם קיימת מוכרת אחת שמציעה יותר מ k כרטיסים, או שקיימות 2 מוכרות שמציעות ביחד יותר מ k כרטיסים. לעומת שתי האופציות האחרות, אופציה זו מאפשרת קניה של חלק מהכרטיסים של מוכרת מסויימת.

שי החליטה לממש שלוש מחלקות אשר עושות שימוש באסטרטגיות אלה.

המחלקה SimpleTM (סעיף ב') – המחלקה מחפשת כרטיסים רק בשיטת ההתאמה המדוייקת (1) המחלקה ExtendedTM – מחפשת כרטיסים בשיטת ההתאמה המדוייקת (1), ואם לא נמצאו כרטיסים, מחפשת בשיטת "שילוב כרטיסים" (2). מחלקה זו תמומש בסעיף ג'.

המחלקה AdvancedTM – חיפוש בשיטה (1), אם לא נמצאו כרטיסים, חיפוש בשיטה (2), ואם עדין לא נמצאו כרטיסים היא תבצע חיפוש בשיטה (3). מחלקה זו תמומש חלקית בסעיף ד'.

לדוגמא:

עבור מערכת שמופיעים בה הכרטיסים הבאים:

מועד d1 - המשתמשת u1 מציעה 2 כרטיסים, והמשתמשת u2 מציעה 3 כרטיסים.

מועד d2 – המשתמשת u3 מציעה 4 כרטיסים.

חיפוש \ מחלקה	מועד d3, 3 כרטיסים	מועד d2, 10 כרטיסים	מועד d1, 2 כרטיסים	מועד d1, 5 כרטיסים	מועד d1, 4 כרטיסים
SimpleTM	{}	{}	{u1=2}	{}	{}
ExtendedTM	{}	{}	{u1=2}	{u1=2, u2=3}	{}
AdvancedTM	{}	{}	{u1=2}	{u1=2, u2=3}	{u1=2, u2=2} or {u1=1, u2=3}

הפונקציה findTickets תחזיר אופציה אחת מבין כל אפשרויות הקניה של k כרטיסים ביום d מסויים, ללא חשיבות לאופן בחירת המוכרות. לדוגמא, ב Advanced, עבור 4 כרטיסים יש שתי אופציות שונות, וכל אחת מהן מהווה ערך החזרה חוקי.

*עבור d1 2 כרטיסים ישנה אופציה אחת בלבד, כיוון שחיפוש מדוייק מחזיר תשובה, לכן, המחלקות Extended ו Advanced מחזירות את התוצאה של חיפוש זה ולא ממשיכות בחיפוש.

*עבור מועד d3 מחזירים כולם מפה ריקה, כיוון שאין כרטיסים עבור אותו היום. עבור המועד d2, אין מספיק כרטיסים שמוצעים למכירה ולכן גם כן חוזרת מפה ריקה.

ממשו את המחלקה SimpleTM כך שהמימוש יתאים גם לצרכים שלכם בסעיפים ג' וד'. עליכם להוסיף שדות וכן להשלים את המימוש של findTickets. אם נדרש, הוסיפו פונקציות עזר ודרשו פונקציות שנורשו מ AbstractTM. שמירת הנתונים צריכה להיות יעילה, כך שתאפשר שליפות מהירות של המידע הרלוונטי, והימנעות ממעברים מיותרים על כרטיסים שלא רלוונטיים לחיפוש. בנוסף, עליכם לעצב את הקוד שלכם כך שיתאפשר שימוש חוזר בקוד ומניעת שכפול קוד.

```
public class SimpleTM extends AbstractTM{
```

```
// members
```

```
protected Map<Date, Map<Integer, Set<String>>> rsvMap
    = new HashMap<>();
```

```
public Map<String,Integer> findTickets(Date date, int ticketsNum){
    Map<String, Integer> res = new HashMap<>();
    if (!rsvMap.containsKey(date) ||
        !rsvMap.get(date).containsKey(ticketsNum)){
        return res;
    }
    addKReservations(date, ticketsNum, 1, res);
    return res; //it's either empty or full
}
```

ניקוד מלא ניתן גם לפתרון שבנה מיפוי מזוגות של Date ו ticketsNum לאוסף של הזדמנות או sellerId. מבחינת ניהול הזכרון, יש עלות למבנה שמכיל Map של Map-ים, כך ש Map יחיד הוא עדיף, אבל גם בשימוש בו יש עלות כיוון שבכל שליפה צריך לייצר את המפתח שהוא זוג של date ו ticketsNum. מצד שני,

בנוסף, מבנה של זוגות מאלץ אותנו בסעיף ג' לעבור על כל הצירופים האפשריים של מספרי כרטיסים בשביל השליפה ממבנה הנתונים. מן הסתם, אם מדובר ברכישת 10K כרטיסים, זה פחות יעיל, אבל אם נתרגם את הבעיה לעולם האמיתי – יש הרבה מועדים אפשריים לביקור תערוכה, וגם כמות ההזמנות במערכת יכולה להיות גדולה, אבל אפשר להניח ש ticketsNum יהיה מספר קטן (לא יעלה על כמה עשרות, במקרה הטוב).

כשהשליפה היא תמיד לפי ticketsNum ו date, האופציה של מפתח הבנוי מזוג היא אופציה טובה מאוד. אם כבר הולכים ל map של map – יש הבדל בין זהות המפתח הראשי למשני. במקרה שלנו, ה date הוא המפתח הראשי כי אנחנו שולפים לפעמים רק לפי date. אם כל השליפות היו לפי date+ticketsNum, אולי היה יתרון בכך שהמפתח הראשי יהיה דווקא ticketsNum. מכיוון שיש הרבה תאריכים אבל יחסית מעט ערכים ל ticketsNum, זה היה מקטין את מספר אובייקטי ה map שנוצרים.

```
// other methods
```

```
public void addTickets(Reservation rsv){
    super.addTickets(rsv);
    if (!rsvMap.containsKey(rsv.getDate())){
        rsvMap.put(rsv.getDate(), new HashMap<>());
    }
    Map<Integer, Set<String>> ticketsForDate =
        rsvMap.get(rsv.getDate());
    if (!ticketsForDate.containsKey(rsv.getTicketsNum())){
        ticketsForDate.put(rsv.getTicketsNum(), new HashSet<>());
    }
    Set<String> ticketsForNum =
        ticketsForDate.get(rsv.getTicketsNum());
    ticketsForNum.add(rsv.getSellerID());

    public void removeSeller(String sellerID){
        Reservation rsv = this.sellerToRsv.get(sellerID);
        super.removeSeller(sellerID);
        Map<Integer, Set<String>> mapForDate = rsvMap.get(rsv.getDate());
        mapForDate.get(rsv.getTicketsNum()).remove(sellerID);
        if (mapForDate.get(rsv.getTicketsNum()).isEmpty()){
            mapForDate.remove(rsv.getTicketsNum());
        }
        if (mapForDate.isEmpty()){
            rsvMap.remove(rsv.getDate());
        }
    }
}
```

```

    }
}

protected boolean addKReservations(Date date, int ticketsNum,
    int reservationsNum, Map<String, Integer> mapToFill){
    Set<String> allSellers = rsvMap.get(date).get(ticketsNum);
    if (allSellers.size() >= reservationsNum){
        Iterator<String> allSellersIt = allSellers.iterator();
        for(int i = 0; i< reservationsNum; i++){
            mapToFill.put(allSellersIt.next(), ticketsNum);
        }
        return true;
    }
    return false;
}
}

```

סעיף ג' (8 בק'): _____

השלימו את מימוש הפונקציה findTickets של המחלקה ExtendedTM. שימו לב שניתן לממש את המחלקה ExtendedTM ללא הוספת שדות נוספים. הוסיפו מימושים של פונקציות נוספות, אם נדרש (פונקציות עזר, דריסה של פונקציות שנורשו מ SimpleTM וכו').

```
class ExtendedTM extends SimpleTM{
```

```

    public Map<String,Integer> findTickets(Date date, int ticketsNum){
        Map<String, Integer> res = new HashMap<>();
        if (!rsvMap.containsKey(date)){
            return res;
        }
        if (this.addKReservations(date, ticketsNum, 1, res)){
            return res;
        }
        Map<Integer, Set<String>> mapForDate = rsvMap.get(date);
        for (int num : mapForDate.keySet()){
            if (num*2 == ticketsNum){
                // buy same number of tickets from two different sellers.
                if (this.addKReservations(date, num, 2, res)){
                    break;
                }
            }
            else{
                if (mapForDate.containsKey(ticketsNum-num)){
                    this.addKReservations(date, num, 1, res);
                    this.addKReservations(date, ticketsNum-num, 1, res);
                    break;
                }
            }
        }
        return res;
    }
}

```

}

סעיף ד' (5 נק'): _____

בסעיף זה עליכם להשלים חלק מהפונקציונליות של AdvancedTM.

בבואה לממש את המחלקה, שי הבינה שהמנשק שלה לא תומך באופציה של עדכון מספר הכרטיסים המוצע למכירה ע"י כל מוכרת. ב SimpleTM וב ExtendedTM זה לא נדרש – בשתי המחלקות כל הכרטיסים של מוכרת מסויימת נמכרים בשלמותם, ולכן שי יכולה לעשות שימוש בפונקציה removeSeller לאחר שמתבצעת מכירה. מכיוון ש AdvancedTM מאפשרת חיפוש שמחזיר רק חלק מהכרטיסים (ולכן גם מאפשר מכירה חלקית של הכרטיסים), שי החליטה להוסיף פונקציה חדשה – updateTickets. ממשו את הפונקציה updateTickets. אין צורך לממש את findTickets.

```
class AdvancedTM extends ExtendedTM{
```

```

/* @pre sellerExists(sellerID)
 * @pre getTicketsNumForSeller(sellerID) > ticketsNum > 0
 * @post getTicketsNumForSeller(sellerID) = ticketsNum
 */
public void updateTickets(String sellerID, int ticketsNum){
    Reservation oldRsv = sellerToRsv.get(sellerID);
    this.removeSeller(sellerID);
    Reservation newRsv = new Reservation(sellerID,
        oldRsv.getDate(), ticketsNum);
    this.addTickets(newRsv);
}

```

```
}
}
```

שאלה 2 (17 נק'):

סעיף א' (9 נק'):

בסעיף זה עליכם לממש ZipIterator אשר פועל בדומה לפונקציית zip של שפת Python, ומאפשר ריצה בלולאה על שני עצמים שמממשים Iterable. מספר האיטרציות של ZipIterator יהיה למספר האיטרציות המינימלי שמאפשר כל אחד משני מהעצמים.

דוגמא לשימוש ב ZipIterator על שתי רשימות (שהן Iterable):

```
Iterable<String> strings = Arrays.asList("a", "b", "c");
Iterable<Integer> ints = Arrays.asList(1,2);
ZipIterator<String, Integer> iter = new ZipIterator<>(strings,ints);
while(iter.hasNext()){
    Pair<String, Integer> pair = iter.next();
    System.out.println(pair.getFirst() + " " + pair.getSecond());
}
```

הלולאה תרוץ במשך שתי איטרציות ותדפיס 1 a ו-2 b.

להלן המימוש של המחלקה Pair שבה תעזרו לצורך המימוש. המימוש המובא הוא חלקי מטעמי מקום, והוא מכיל מכיל בנאי ופונקציות get.

```
public class Pair<A,B>{
    private A first;
    private B second;

    public Pair(A first, B second){ /* code here */ }

    /* getFirst, getSecond implemented here */
}
```

השלימו את מימוש המחלקה ZipIterator:

```
public class ZipIterator <A,B>
    implements Iterator<Pair<A,B>> {
```

```
    // members
    Iterator<A> firstIterator;
    Iterator<B> secondIterator;
```

```
    public ZipIterator(Iterable<A> firstIterable,
        Iterable<B> secondIterable) {
```

```
        this.firstIterator=firstIterable.iterator();
        this.secondIterator=secondIterable.iterator();
```


}

```
public boolean hasNext() {
```

```
    return firstIterator.hasNext() && secondIterator.hasNext() ;
```

}

```
public Pair<A,B> next() {
```

```
    return new Pair<>(firstIterator.next(), secondIterator.next());
```

}

}

סעיף ב' (8 נק')

ממשו את הפונקציה `getLongestSeq` אשר בהנתן מחרוזת `str` ותו `chr` (character) מחזירה את אורך הרצף הכי ארוך של `chr` בתוך `str`. אם `chr` לא מופיע כלל ב `str` הפונקציה תחזיר 0. לדוגמא, עבור המחרוזת `abxxxxcax` והתו `x`, הפונקציה תחזיר 3 כיוון שאורך הרצף הכי ארוך של `x` הוא 3 (מודגש בגוף המחרוזת). ניתן להניח כי אורך המחרוזת הוא גדול מ 0.

הנחיה: סעיף זה לא קשור לסעיף א', כך שאין צורך לעשות שימוש ב `ZipIterator`. עליכם לממש את הפונקציה כך שהיא תעשה מעבר יחיד על המחרוזת `str`. שימוש במתודות חיפוש (כמו `contains`) מבצע מעבר יחיד על כל המחרוזת, כך שלא ניתן לקרוא לפונקציות אלה יותר מפעם אחת.

```
public static int getLongestSeq(String str, char chr){
```

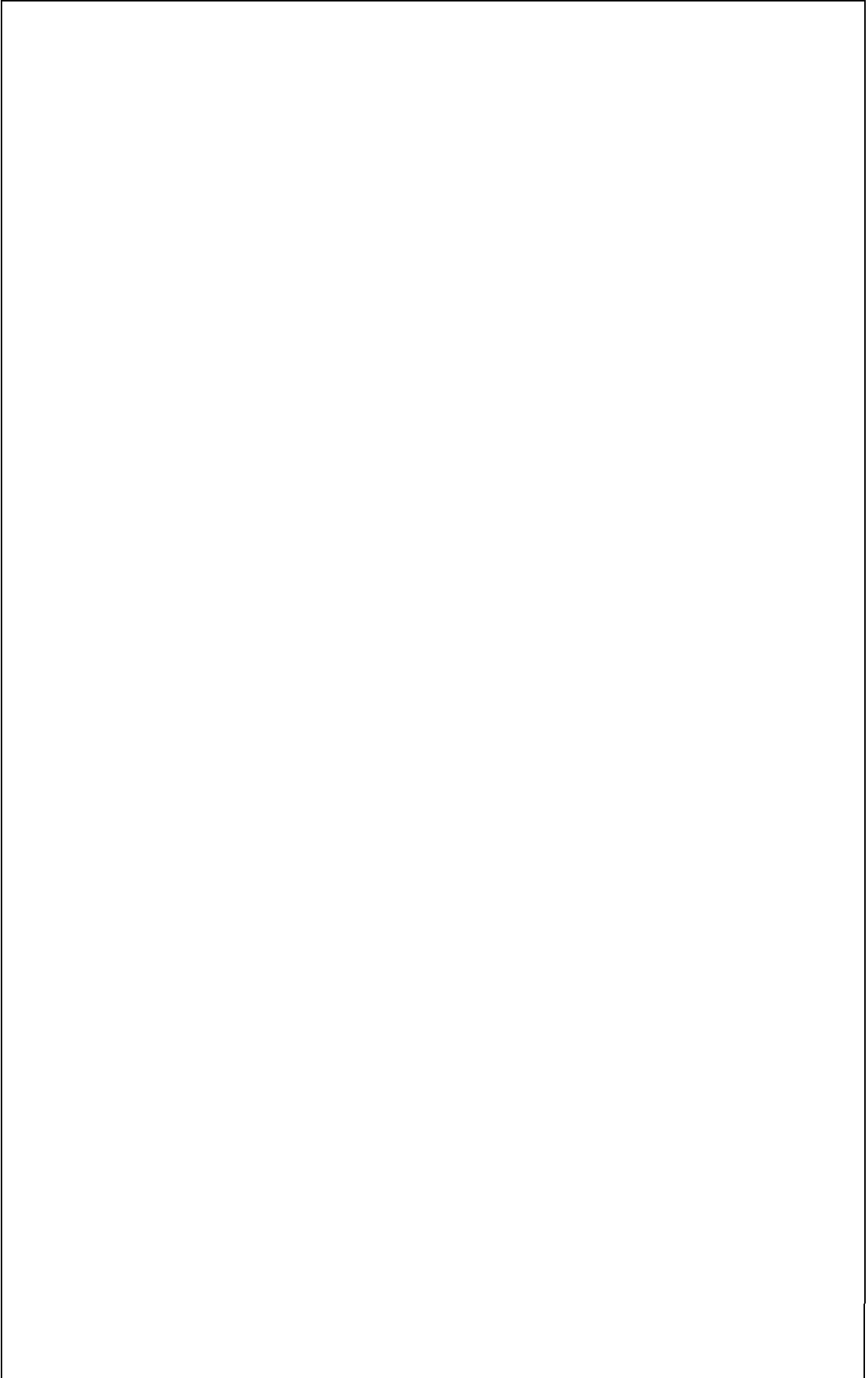
```
    int maxSeqLen = 0;
    int currSeqLen = 0;
    for (int i = 0; i < str.length(); i++){
        if (str.charAt(i) == chr){
            currSeqLen += 1;
            maxSeqLen = currSeqLen > maxSeqLen? currSeqLen: maxSeqLen;
        }
        else{
            currSeqLen = 0;
        }
    }
    return maxSeqLen;
}
```

עמוד 10 מתוך 20

מספר זהות: _____ מספר מחברת: _____

}

מסגרת חירום



שאלה 3:

```
public class Q3 {
    public void func1(List<? extends IOException> lst1,
                    List<? super Exception> lst2){
        lst1.add(lst2.get(0));
    }

    public void func2(List<? extends IOException> lst1,
                    List<? extends Exception> lst2){
        lst1 = lst2;
    }

    public void func3(List<? extends IOException> lst1,
                    List<Object> lst2){
        lst1 = lst2;
    }
}
```

אילו מבין הפונקציות הבאות מתקמפלת? בחר\י בתשובה הטובה ביותר:

- א. כל הפונקציות מתקמפלות.
- ב. כל הפונקציות לא מתקמפלות.
- ג. רק פונקציה func1 מתקמפלת.
- ד. רק פונקציה func2 מתקמפלת.
- ה. רק פונקציה func3 מתקמפלת.
- ו. רק פונקציות func1+func2 מתקמפלת.
- ז. רק פונקציות func2+func3 מתקמפלת.
- ח. רק פונקציות func1+func3 מתקמפלות.

נימוק:

שאלה 4:

לפניך מספר טענות על המילה השמורה final. בחר\י בתשובה הטובה ביותר:

- א. לא ניתן לרשת ממחלקה שמוגדרת כ final.
- ב. שדה שהוא final חייב להיות מוגדר כ static.
- ג. ניתן לדרוס פונקציה שמוגדרת כ final, אך אסור לעשות שימוש בקוד של הפונקציה הנדרסת.
- ד. שדה שמוגדר כ final חייב להיות מטיפוס immutable.
- ה. מלבד תשובה זו כל התשובות לא נכונות.
- ו. מלבד תשובה זו יש יותר מתשובה נכונה אחת.

נימוק:

שאלה 5:

```

public class Point{
    private int x,y;
    private Random rand = new Random();

    public Point(int x, int y){ this.x = x; this.y = y; }

    @Override
    public boolean equals(Object o){
        Point other = (Point)o;
        return this.x+this.y == other.x+other.y;
    }

    @Override
    public int hashCode(){ return rand.nextInt(3); } //random in [0,2]

    public static void main(String[] args){
        Point p1 = new Point(1,5);
        Point p2 = new Point(2,4);
        Point p3 = new Point(3,6);
        Set<Point> set = new HashSet<>();
        set.add(p1);
        System.out.println(set.contains(p1)); //#
        System.out.println(set.contains(p2)); //$
        System.out.println(set.contains(p3)); //*
    }
}

```

לפניכם 3 טענות הקשורות להרצת התוכנית Point. הטענות מתייחסות לתוצאות אפשריות של ריצה כלשהי, כיוון שיש פה חלק רנדומלי (ב hashCode). ניתן להניח שגודל ה HashSet גדול מ 1.

טענה 1: הקוד בשורה # יכול להדפיס false בריצה מסויימת.

טענה 2: הקוד בשורה \$ יכול להדפיס true בריצה מסויימת.

טענה 3: הקוד בשורה * יכול להדפיס true בריצה מסויימת.

מהן הטענות הנכונות? בחר\י בתשובה הטובה ביותר:

- א. כל הטענות לא נכונות.
- ב. רק טענה 1 נכונה.
- ג. רק טענה 2 נכונה.
- ד. רק טענה 3 נכונה.
- ה. רק טענות 1+2 נכונות.
- ו. רק טענות 1+3 נכונות.
- ז. רק טענות 2+3 נכונות.
- ח. כל הטענות נכונות.

נימוק:

שאלה 6:

```
public void func(A a) {
    B b = (B)a;
}
```

לפניכם 3 טענות על הקוד הבא:

טענה 1: אם $B \mid A$ הן מחלקות קונקרטיות, ב upcasting (A הוא צאצא של B) נוצר אובייקט חדש על ה heap אשר מכיל את כל השדות שקיימים ב B ולא ב A.

טענה 2: ב downcasting (B הוא צאצא של A) הקוד תמיד יתקמפל, אבל עלול לזרוק שגיאת זמן ריצה.

טענה 3: אם $B \mid A$ הם שני מנשקים שלא קשורים אחד לשני (אחד לא יורש מהשני), הקוד לא יתקמפל.

בחר/י בתשובה הטובה ביותר:

א. רק טענה 1 נכונה.

ב. רק טענה 2 נכונה.

ג. רק טענה 3 נכונה.

ד. רק טענות 1+2 נכונות.

ה. רק טענות 1+3 נכונות.

ו. רק טענות 2+3 נכונות.

ז. כל הטענות נכונות.

ח. כל הטענות לא נכונות.

נימוק:

שאלה 7:

```
class Outer<T>{
    public class Inner{ T t; } //#
    public static class Nested{ T t; } //$
    public static void func(List<T> lst){} //*
}
```

בחר/י בתשובה הטובה ביותר:

א. רק שורה # מתקמפלת.

ב. רק שורה \$ מתקמפלת.

ג. רק שורה * מתקמפלת.

ד. רק שורות # ו \$ מתקמפלות.

ה. רק שורות # ו * מתקמפלות.

ו. רק שורות \$ ו * מתקמפלות.

ז. כל השורות מתקמפלות.

ח. כל השורות לא מתקמפלות.

נימוק:

שאלה 8:

```

public class Box<S, T extends Comparable<String>> {
    T item; // #
    //Object item

    List<S> values = new ArrayList<>(); // $
    //List<Object> values = new ArrayList<>();

    public Box(T item){this.item = item;}

    public T getItem(){ return item; }

    public void addValue(S val){ this.values.add(val); }

    public static void main(String[] args){
        Box<Integer, String> b = new Box<>("abc");
        String item = b.getItem(); /*
        // String item = (String)b.getItem();
    }
}

```

שאלה זו מתייחסת לתהליך ה erasure (מחיקת הטיפוסים) שקורה בזמן הקומפילציה של המחלקה. בחר\י בתשובה הטובה ביותר:

- טענה 1: השורה המסומנת ב # מוחלפת בזמן קומפילציה בשורה המופיעה מתחתיה בהערה.
 טענה 2: השורה המסומנת ה \$ מוחלפת בזמן קומפילציה בשורה המופיעה מתחתיה בהערה
 טענה 3: השורה המסומנת ב * מוחלפת בזמן הקומפילציה בשורה המופיעה מתחתיה בהערה

בחר\י בתשובה הטובה ביותר:

- א. רק טענה 1 נכונה.
 ב. רק טענה 2 נכונה.
 ג. רק טענה 3 נכונה.
 ד. רק טענות 1+2 נכונות.
 ה. רק טענות 1+3 נכונות.
 ו. רק טענות 2+3 נכונות.
 ז. כל הטענות נכונות.
 ח. כל הטענות לא נכונות.

נימוק:

שאלה 9:

```
public class Q9 {
    public static void main(String[] args) {
        Mom m = new Child();
        System.out.println(m.func());
    }
}

public class Grandma{
    public int func() { return 1; }
    public int foo() { return 1; }
}

public class Mom extends Grandma{
    public int foo() { return 1+super.foo(); }
}

public class Child extends Mom{
    public int func() {
        int res = 1 + ((Mom)this).foo();
        res += super.func(); // #
        return res;
    }

    public int foo() { return 1+super.foo(); }
}
```

בחר/י בתשובה הטובה ביותר:

- א. ישנה שגיאת קומפילציה בשורה המסומנת ב #.
- ב. הקוד מתקמפל אך נזרקת NullPointerException בעת ריצתו.
- ג. הקוד מתקמפל, רץ ומדפיס 1.
- ד. הקוד מתקמפל, רץ ומדפיס 2.
- ה. הקוד מתקמפל, רץ ומדפיס 3.
- ו. הקוד מתקמפל, רץ ומדפיס 4.
- ז. הקוד מתקמפל, רץ ומדפיס 5.
- ח. הקוד מתקמפל, רץ ומדפיס 6.

נימוק:

שאלה 10:

```
public class Mom{
    /* //op1
    private void foo(Integer num) { } */

    /* //op2
    public Integer foo(Object num) { return 1;} */

    /* //op3
    public Integer foo(Integer num) throws Exception{return 1;} */
}

public class Child extends Mom{
    protected Integer foo(Integer num) { return 1;}
}
```

נרצה להוציא אחת מהפונקציות של Mom מהערה כל שהקוד יתקמפל. בחר\י בתשובה הטובה ביותר:

- א. רק אם נוציא את op1 מהערה, הקוד של Mom ו Child יתקמפל.
- ב. רק אם נוציא את op2 מהערה, הקוד של Mom ו Child יתקמפל.
- ג. רק אם נוציא את op3 מהערה, הקוד של Mom ו Child יתקמפל.
- ד. רק אם נוציא את op1 או op2 מהערה, הקוד של Mom ו Child יתקמפל.**
- ה. רק אם נוציא את op1 או op3 מהערה, הקוד של Mom ו Child יתקמפל.
- ו. רק אם נוציא את op2 או op3 מהערה, הקוד של Mom ו Child יתקמפל.
- ז. אם נוציא כל אחת מהפונקציות מהערה, הקוד יתקמפל.
- ח. אם נוציא כל אחת מהפונקציות מהערה, הקוד לא יתקמפל.

נימוק:

שאלה 11:

בחר\י בתשובה הטובה ביותר:

- א. שימוש ב Git מאפשר לשמור גירסאות שונות של הקוד בצורה נוחה.**
- ב. ה interpreter (מפרש) קורא קבצים המכילים קוד java ומתרגם אותם לשפת מכונה.
- ג. רקורסיה אינסופית ב Java מגדילה בהכרח את צריכת הזכרון על ה heap, אך לא מגדילה בהכרח את צריכת הזכרון על ה stack.
- ד. מלבד תשובה זו, כל התשובות לא נכונות.
- ה. מלבד תשובה זו יש יותר מתשובה נכונה אחת.

נימוק:

שאלה 12:

```
public interface IDecidable {
    public int decide(char a, char b);
}

public class DecideClass implements IDecidable{
    public static int i = 0;
    public int decide(char a, char b) {
        return 1;
    }
}

public class PrintDecidable {

    public static void printDecision(IDecidable item) {
        System.out.println(item.decide('x', 'y'));
    }

    public static void main(String[] args) {
        // printDecision((a,b) -> "abc" + a); // op1
        // printDecision((a,b) -> new DecideClass()); //op2
        // printDecision((a,b) -> DecideClass.i++); //op3
    }
}
```

נרצה לדעת אילו מבין השורות המופיעות ב main מתקמפלות. בחר\י בתשובה הטובה ביותר:

- א. רק op1 מתקמפלת.
- ב. רק op2 מתקמפלת.
- ג. רק op3 מתקמפלת.
- ד. רק op1 ו op2 מתקמפלות.
- ה. רק op1 ו op3 מתקמפלות.
- ו. רק op2 ו op3 מתקמפלות.
- ז. כל השורות מתקמפלות.
- ח. כל השורות לא מתקמפלות.

נימוק:

שאלה 13:

להלן 3 טענות על מחלקות אבסטרקטיות:

טענה 1: במחלקה אבסטרקטית אין בנאים.

טענה 2: מחלקה אבסטרקטית נועדה להכיל קוד משותף בין המחלקות היורשות אותה.

טענה 3: מחלקה אבסטרקטית חייבת להכיל פונקציות אבסטרקטיות.

בחר\י בתשובה הטובה ביותר:

א. רק טענה 1 נכונה.

ב. רק טענה 2 נכונה.

ג. רק טענה 3 נכונה.

ד. רק טענות 1+2 נכונות.

ה. רק טענות 1+3 נכונות.

ו. רק טענות 2+3 נכונות.

ז. כל הטענות נכונות.

ח. כל הטענות לא נכונות.

נימוק:

--

public interface Map<K,V>

Modifier and Type	Method and Description
void	<code>clear()</code> Removes all of the mappings from this map (optional operation).
boolean	<code>containsKey(Object key)</code> Returns true if this map contains a mapping for the specified key.
boolean	<code>containsValue(Object value)</code> Returns true if this map maps one or more keys to the specified value.
<code>Set<Map.Entry<K,V>></code>	<code>entrySet()</code> Returns a Set view of the mappings contained in this map.
V	<code>get(Object key)</code> Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
V	<code>getOrDefault(Object key, V defaultValue)</code> Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
boolean	<code>isEmpty()</code> Returns true if this map contains no key-value mappings.
<code>Set<K></code>	<code>keySet()</code> Returns a Set view of the keys contained in this map.
V	<code>put(K key, V value)</code> Associates the specified value with the specified key in this map. Returns the previous value associated with key, or null if there was no mapping for key.
V	<code>remove(Object key)</code> Removes the mapping for a key from this map if it is present (optional operation). Returns the previous value associated with key, or null if there was no mapping for key.
int	<code>size()</code> Returns the number of key-value mappings in this map.
<code>Collection<V></code>	<code>values()</code> Returns a Collection view of the values contained in this map.

public interface Set<E> extends Collection<E>

boolean	<code>add(E e)</code> Adds the specified element to this set if it is not already present.
void	<code>clear()</code> Removes all of the elements from this set (optional operation).
boolean	<code>contains(Object o)</code> Returns true if this set contains the specified element.
boolean	<code>isEmpty()</code> Returns true if this set contains no elements.
<code>Iterator<E></code>	<code>iterator()</code> Returns an iterator over the elements in this set.
boolean	<code>remove(Object o)</code> Removes the specified element from this set if it is present.
int	<code>size()</code>

	Returns the number of elements in this set (its cardinality).
--	---

public interface List<E> extends Collection<E>

boolean	add(E e) Appends the specified element to the end of this list. Always returns true.
void	clear() Removes all of the elements from this list.
boolean	contains(Object o) Returns true if this list contains the specified element.
E	get(int index) Returns the element at the specified position in this list.
int	indexOf(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
Iterator<E>	iterator() Returns an iterator over the elements in this list in proper sequence.
boolean	remove(Object o) Removes the first occurrence of the specified element from this list, if it is present. Returns true if this list contained the specified element
int	size() Returns the number of elements in this list.
default void	sort(Comparator<? super E> c) Sorts this list according to the order induced by the specified Comparator.

public final class String

char	charAt(int index) Returns the char value at the specified index.
int	compareTo(String anotherString) Compares two strings lexicographically.
static String	join(CharSequence delimiter, CharSequence... elements) Returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
int	length() Returns the length of this string.
String[]	split(String regex) Splits this string around matches of the given regular expression.
String	trim() Returns a string whose value is this string, with any leading and trailing whitespace removed.

public interface Iterator<E>

boolean	hasNext() Returns true if the iteration has more elements.
E	next() Returns the next element in the iterator .

public interface Iterable<T>

Iterator<T>	iterator() Returns an iterator over elements of type T.
-------------	--

עמוד 21 מתוך 20

מספר זהות: _____ מספר מחברת: _____