

## שיטות מתקדמות בתכנות 214.3601 בחינה סופית מועד א'

1. כתוב Iterator המקבל ב constructor שלו שני איטרטורים והעובר על כל האיברים של שניהם.  
א' – כאשר סדר המעבר הוא קודם האיברים של האיטרטור הראשון ואח"כ כל האיברים של האיטרטור השני. לדוגמא אם סדר המניה של האיטרטור הראשון היה 1,2,3,4,5 ושל האיטרטור השני 6,7,8 אז סדר המניה של האיטרטור שלך צריך להיות 1,2,3,4,5,6,7,8  
ב' – כאשר סדר המעבר הוא לסירוגין (פעם מזה ופעם מזה). לדוגמא אם סדר המניה של האיטרטור הראשון היה 1,2,3,4,5 ושל האיטרטור השני 6,7,8 אז סדר המניה של האיטרטור שלך צריך להיות 1,6,2,7,3,8,4,5  
ג' – מה יעשו האיטרטורים שכתבת בסעיפים 1 ו 2 אם ניתן להם ב constructor פעמיים את אותו האיטרטור.  
ד' אם תרצה למנוע מצב זה ששני הפרמטרים יהיו אותו איטרטור עצמו כיצד תעשה זאת?  
ה' מה יעשו האיטרטורים הנ"ל את ניתן להם ב constructor שני עותקים של איטרטור על אותה הרשימה.  
ו' הגדר מה הם התנאים בהם מתודת ה remove שכתבת ב א' ו ב' תצליח ולא תזרוק Exception כלשהו.



2. ממש Panel היודע לבחור אחת מ 25 אופציות ידועות מראש. כאשר הפנל מופיע לראשונה, אף אחד מ 25 הכפתורים איננו בחור (selected) וכפתור ה load במצב disabled. המשתמש יכול לבחור כל אחת מ 25 האופציות ע"י לחיצה על הכפתור המתאים (למשל בכדי לבחור "6" עליך לחוץ על הכפתור המסומן ב "6"). אם אופציה כלשהי נבחרה, כפתור ה load הופך enabled ונשאר כך, כל עוד יש אופציה בחורה (לחיצה על כפתור שהוא selected הופכת אותו ל unselected). אם כאשר כפתור כלשהו selected והמשתמש בוחר אופציה אחרת (למשל "14" היה selected והמשתמש בחר "3") האופציה החדשה נשארת selected והאופציה הקודמת הופכת ל unselected (עכשיו "3" selected ו "14" הופך unselected). שים לב שבכל שלב יש לכל היותר אופציה אחת בחורה. כאשר המשתמש בוחר Load האופציה שנבחרה נטענת למערכת.



הנחות ורמזים:

א' קיים { void load(int what); } interface Loader

אתה רשאי להניח שב constructor של ה panel שלך תקבל את ה Loader שבו עליך להשתמש.

ב' את כל הכפתורים ניתן לממש ע"י JButton לדוגמא:

```
JButton b = new JButton("1");
```

ל JButton יש מתודה setSelected(boolean) בעזרתה ניתן לבחור או לבטל את הבחירה של הכפתור.

הנח שקיים class CheckerBoardPanel extends JPanel

שמקבל ב constructor שלו שני מספרים h,w ויודע לסדר את הרכיבים הבאים שיתווספו לו ע"י add() בצורת ריבוע יפה.

ד' בנוסף לכתיבת הקוד, מומלץ להסביר בכמה מילים את ה design שלך. במקרה של טעויות במימוש עדין תקבל את מרבית הנקודות אם הבודק הצליח להבין את ה design שלך וחושב שהוא נכון.

ה' אין צורך לממש את הפעולה המתבצעת כאשר לוחצים על cancel



3. בחברת ההשקעות "קרן הצבי" מימשו את הקוד של המסלקה באופן הבא:

```
class AccountsHandler {
    ...
    private void depositMoney(int accountNumber, int sumToDeposit) {
        int money = accounts.get(accountNumber);
        money += sumToDeposit;
        accounts.set(accountNumber, money);
    }

    private void withdrawMoney(int accountNumber, int sumToWithdraw) {
        int money = accounts.get(accountNumber);
        money -= sumToWithdraw;
        accounts.set(accountNumber, money);
    }
    private int getBalance(int accountNumber) { accounts.get(accountNumber); }
}
```

בודק תוכנה זריז, שם לב שהקוד הנ"ל איננו thread safe ושבסביבה מרובת Threads יתכנו שגיאות.  
3א' תן דוגמה לשגיאה אפשרית ולתרחיש שיגרום לה לקרות.  
3ב' תקן את הקוד הנ"ל באופן שיהיה Thread safe : אבל לא יסבול מהשהיות מיותרות (מצב בו הפקדה לחשבון אחד מתעכבת בגלל שהפקדה לחשבון אחר מתרחשת בו זמנית איננו קביל).

3ג' ממש במחלקה הנ"ל מתודה נוספת

```
void transfer(int senderAccount, int receiverAccount, int sum);
```

המעבירה סך של sum מחשבון ה sender לחשבון ה receiver

3ד' מה יקרה אם בדיוק בזמן ש thread אחד קורא ל transfer(123, 321, 100) אז thread שני קורא ל transfer(321, 123, 200)?



4. חברת BuyAndSellOnLine מציעה שירותי קניה ומכירה ברשת. השירות כולל קטלוג של המוצרים הנמכרים ו"סוכנים" שעוסקים במכירה של מוצרים. התוכנה ממומשת בעזרת טכנולוגיית RMI וכוללת את הממשקים הבאים:

```
interface Agent extends Remote {
    void buy(int buyerCreditAccount, int productId) throws RemoteException;
}
class ProductDetails {
    int id;
    String name;
    String description;
    int price;
    Agent agent;
}
interface Catalog extends Remote {
    ProductDetails[] searchByName(String name) throws RemoteException;
    ProductDetails[] searchByDescription(String name) throws RemoteException;
    ProductDetails[] searchById(int Id) throws RemoteException;
    int addToCatalog(ProductDetails[] products) throws RemoteException;
    void removeFromCatalog(ProductDetails[] products, int password)
        throws RemoteException;
}
```

שים לב ששתי המתודות האחרונות מיועדות ללקוחות שרוצים להציע מוצרים משלהם למכירה. לקוח שרוצה להציע מוצרים למכירה קורה למתודה addToCatalog המוסיפה את המוצרים שהוא מציע למכירה לקטלוג. במידה וההוספה הצליחה (היא תכשל אם קיים כבר במערכת מוצר עם אותו id אבל שם או תיאור שונים) המתודה תחזיר מספר המשמש כ password למתודה removeFromCatalog אליה יש לקרוא כאשר לקוח מעוניין להסיר מן הקטלוג הצעות מכירה שנתן בעבר.

השרת של חברת BuyAndSellOnLine נקרא buyAndSellOnLineServer ובכדי לקבל גישה לקטלוג יש לבקש את השירות "Catalog Service" לדוגמא קוד של לקוח יכול להכיל את השורה:

```
Catalog cat = Naming.lookup("buyAndSellOnlineServer", "Catalog Service");
```

א' כתוב קוד של לקוח המחפש מוצר הנקרא "Johnny's hair spray" והקונה אותו מן הסוכן שמוכר אותו הכי בזול.

ב' חוני גאוני המציא אלגוריתם חדש וגאוני לנבא את עליית המחירים של מוצרים שונים

```
int id = getIdOfNextHottestItem();
```

חוני רוצה לבדוק את האלגוריתם שלו על קטלוג המוצרים של חברת BuyAndSellOnLine. כתוב מתודה checkMyAlgorithm שתעשה זאת באופן הבא:

המתודה תזכה "הזדמנות" (בעזרת האלגוריתם getIdOfNextHottestItem), תיצור קשר עם הקטלוג, תקנה את המוצר מהסוכן שמוכר אותו במחיר הזול ביותר (ניתן להשתמש בשגרה מסעיף א') ואח"כ תציע אותו מחדש למכירה במחיר הגבוה ב 20% ממחיר הקניה. כאשר מישוהו יקנה את המוצר במחיר הנ"ל יש להדפיס "Success" ולהוריד את הצעת המכירה מן הקטלוג.

ג' הרחב את הקוד מסעיף ב' כך שאם במשך פרק זמן השקול ל Timeout לא התבצעה מכירה יש להדפיס "Failure".

