תרגיל 6

תכנות מונחה עצמים

תאריך הגשה: אין.

גם אין קבצי בדיקה. גם כי התרגיל לא להגשה, אבל יותר מזה, כי קשה מאוד להריץ בדיקות אוטומטיות על תוכניות רב-חוטיות.

מצד שני, הנה הפתרונות כך שתוכלו לבדוק אם מה שעשיתם נכון.

מאוד מומלץ לנסות ולפתור את השאלות. גם תדעו משהו על ריבוי-חוטים וגם זו הכנה מצוינת למבחן.

שאלה 1

בשאלה זו נכתוב תוכנית קטנה המריצה כמה חוטים הפועלים במעגל: חוט אחד ער, כותב משהו, מעיר את הבא בתור והולך לישון. אז תור השני, וכך הלאה.

נשתמש פה במנגנון של interrupt כדי להעיר חוטים ישנים. זו לא הדרך "הנכונה" לעשות זאת. אבל זוהי דרך טובה לתרגל הרצת חוטים רבים וסנכרון ביניהם.

שם המחלקה שלנו הוא Cascade. בשיטה היחידה שלה runExample(int n), היא תריץ n חוטים (וכמובן תצטרכו לכתוב מחלקה פנימית שתהיה Runnable לשם כך).

כל חוט חוזר בלולאה אינסופית על המהלך הבא:

1. ישן.
2. מחכה שמישהו יעיר אותו בעזרת interrupt.
3. כותב את שמו.
4. מחכה שנייה אחת.
5. מעיר את החוט הבא בתור (חשבו איך הוא ידע איזה חוט להעיר..)

השיטה runExample מייצרת את כל החוטים ומריצה אותם, ואז מעירה את החוט הראשון כך שיתחיל את הסבב.

שאלה 2

כתבו את המחלקה Exchange<E>. יש לה שתי שיטות:

```
public E give1(E e)
public E give2(E e)
```

השימוש לאובייקט מסוג Exchange, הוא כדרך תקשורת בין שני חוטים. אם לשניהם יש מופע שלו exchange, ואחד קורא לשיטה give1, אז הוא יחכה עד שהשני יקרה ל-give2. קריאה זו תשחרר את שני החוטים וכל אחד יקבל את הקלט שהשני נתן. אותו דבר יקרה אם השיטה give2 נקראה קודם - החוט שקרה לה יחכה, עד שהחוט השני יקרא ל-give1.

למשל, אם נריץ שני חוטים ביחד, וה-run של אחד החוטים הוא:

```
public void run() {
        System.out.println("Runner1");
        System.out.println(exchange.give1("Hi1"));
}
```

ושל החוט השני הוא:

```
public void run() {
        System.out.println("Runner2");
        System.out.println(exchange.give2("Hi2"));
}
```

אז יודפס למשל:

```
Runner1
Runner2
Runner2 got: Hi1
Runner1 got: Hi2
```

הסדר בין שתי השורות הראשונות יכול להיות הפוך, וגם הסדר בין שתי האחרונות, אבל אף פעם לא יתערבב משהו משתי הראשונות עם משהו משתי האחרונות.

בתור תרגול וגם לבדיקה, שווה לכתוב הדגמה של שימוש בשיטה זו ע"י שני חוטים שמעבירים אחד לשני משהו.. שווה להוסיף קצת sleep כדי לראות מה קורה יותר טוב.

כמובן, השיטה לסנכרון פה תהיה דרך monitor - כלומר דרך פקודות wait ו-notify.

שאלה 3

המטרה בתרגיל זה היא ליצור מחלקה שתיצור ותריץ חוט אבל לא עם Runnable שבו אי אפשר לשלוח ל-run כלום, ואי אפשר לקבל ממנו תוצאה, אלא שתריץ:

```
public interface Callable<U, V> {
    public U call(V val);
}
```

שיכול לקבל ערך val ולהחזיר תוצאה מסוג U.

יש פה בעיה כמובן: מי יקבל את התוצאה? הרי ברגע שנריץ את החוט הוא רץ בנפרד, והקוד שלנו ממשיך, אז לאיפה ערך החזרה יגיע? בתרגיל זה נציע שיטה אחת לפתרון בעיה זו.

כתבו את המחלקה RunCallable<U,V>, לה יש את השיטות הבאות:

public void start(Callable<U,V> c, V val)	מתחילה חוט חדש המריץ את השיטה call של c עם val בתור פרמטר.
public U waitForResult()	מחכה עד שהחוט שה-call שהורץ ע"י start מסיימת, ואז מחזיר את הערך שהיא החזירה.
public U getResult()	מחזיר את ערך החזרה כמו בשיטה הקודמת אם החוט כבר סיים, ואחרת מחזיר null (כלומר לא מחכה כלל)

כדאי גם להוסיף בדיקה ל-start שהוא לא נקרא כבר - כי המחלקה הזאת לא בנוייה לתמיכה בכמה הרצות (בפתרון בלינק למעלה זה לא מופיע).

בנוסף, כתבו main קטן שמדגים שימוש במחלקה זו - למשל על איזשהו Callable<Integer, String> שמקבל מספר ומייצר איזושהי מחרוזת מהמספר הנתון.

בקיצור, כתבו טסטים לעצמכם :)