בחינה ב"מבוא לתכנות מונחה עצמים"

מספר קורס: 7027910
סמסטר א', מועד א', 10.2.2020
מרצים: בעז בן משה, אליזבת איצקוביץ

משך הבחינה שעתיים וחצי
אין חומר עזר (אסור ספרים, אסור מחשבים, אסור מחברות - או כל חומר אחר)
מבנה הבחינה: 4 שאלות בעלות משקל שווה, יש לענות על כולן.
תשובות מסורבלות או דו משמעיות לא יזכו בניקוד מלא.

מותר בהחלט לענות על סעיף אחד בעזרת סעיף אחר.
לבחינה זו מצורפים החומרים הבאים:

1. ממשק שמייצג גרף מכוון, ממושקל (graph).
2. ממשק שמייצג קודקוד בגרף מכוון (node_data).
3. ממשק שמייצג צלע בגרף מכוון ממושקל (edge_data).

הקפידו על טוהר הבחינה !!

בהצלחה !

שאלה 1 (25 נקודות):

התייחסו לממשק של graph שמייצג גרף מכוון ממושקל (מצורף).
בשאלה זו נתכנן ונממש את המחלקה EdgeIterator שמממשת את הממשק Iterator<edge_data>:

```
public interface Iterator<T> {
        public boolean hasNext();  // return true iff there are still “unvisited” elements of T.
        public T next(); // return the “next” element of type T.
}
```

1.1 (5 נקודות) הסבירו במילים כיצד יש לממש את המחלקה מבחינת שדות מידע, ואופן פעולה.
1.2 (15 נקודות) ממשו באופן פשטני את המחלקה EdgeIterator
1.3 (5 נקודות) הסבירו במילים כיצד ניתן לוודא שמופע של ה EdgeIterator מעודכן לגרף, משמע: אם הגרף משתנה לאחר יצירת האובייקט (של האיטרטור) על אותו אובייקט EdgeIterator לזרוק שגיאה.

שאלה 2 (25 נקודות):

התייחסו לממשק של graph, הניחו שקיימת לכם מחלקה בשם DGraph שמממשת את הממשק (שימו לב בשאלה זו אין צורך לכתוב קוד):
2.1 (12 נקודות) הסבירו באופן כללי איזו בעיה יכולה להיגרם אם מספר תהליכים ייגשו לאותו גרף. הדגימו בעיה ספציפית, אילו שיטות יש צורך לשנות כדי לאפשר לגרף לתמוך במספר תהליכים.
2.2 (13 נקודות) הסבירו באופן כללי (אין צורך לכתוב קוד) כיצד ניתן לאפשר לתהליכים שונים לקבל עדכון כאשר הגרף משתנה, ציינו את תבנית העיצוב הרלוונטית לכך.

שאלה 3 (25 נקודות):

3.1 (8 נקודות) במטלות 3,4 נדרשתם להזיז רובוטים ע"ג גרף. המערכת נדרשה לעדכן את השרת (ולהתעדכן ממנו) וכן להציג את מיקום הרובוטים במשק גרפי. בעוד שההצגה במשק הגרפי צריכה להיות בקצב אחיד (נניח 10 פעמים בשנייה) העדכונים של "השרת" תלויים במיקום ומהירות של הרובוטים (בקצב משתנה ותלוי אירועים). הסבירו כיצד ניתן לפתור בעיה זו בצורה מערכתית מיטבית.
3.2 (9 נקודות) בקורס הצגנו שתי דרכים שונות לשמור מידע בקובץ: דרך ממשק serializable, ודרך json, הסבירו בקצרה כל אחת מהשיטות ועמדו על היתרונות והחסרונות של כל שיטה.
3.3 הניחו שלשני סטודנטים (Alice, Bob) יש פרויקט משותף ב github, הסבירו מהו Conflict, כיצד הוא נגרם וכיצד ניתן לפתור אותו, הדרכה: השתמשו במושגים הרלוונטיים מעולם ה git.

שאלה 4 (25 נקודות):

בשאלה זו נתכנן מערכת לניהול מערך מעליות בבניין. הניחו שמדובר במעליות "חכמות" שמחיבות את המשתמש להקיש את קומת היעד (בקומת המקור), ולאחר מכן המשתמש מקבל חיווי לגבי מספר המעלית שתיקח אותו לקומת היעד, ואז המעלית מגיעה לקומת המקור אוספת את המשתמשים ונעה כלפי היעדים (והמקורות) שהמערכת הגדירה לה.
הניחו שבבניין קיימות K מעליות, ו N+1 קומות (בין קומה 0 לקומה N)
4.1 (9 נקודות) אפיינו אילו מחלקות נדרשות כדי לבנות את המערכת: (לכל מחלקה ציינו: שם, תפקיד ופירוט יכולות)
4.2 (8 נקודות) תכננו באופן כללי את האלגוריתם שמשבץ כל בקשה של משתמש למעלית ספציפית. הדרכה: נסו לצמצם למינימום את משך ההמתנה הכללי של משתמשי המעליות.
4.3 (8 נקודות) תכננו מערכת סימולציה שתאפשר בדיקת ביצועים של אלגוריתם שיבוץ מבחינת משך ההמתנה הכללי עבור תרחיש בקשות של משתמשי המעליות בבניין.

בהצלחה!

חומר עזר מצורף:

```java
/** This interface represents a directional weighted graph. */
public interface graph {
     /**return the node_data by the node_id (null - if none)*/
     public node_data getNode(int key);
     /** return the data of edge(src,dest), null if none */
     public edge_data getEdge(int src, int dest);
     /** add a new node to the graph with the node_data. */
     public void addNode(node_data n);
/** Connects an edge with weight w between src->dest. */
     public void connect(int src, int dest, double w);
     /** Return a reference (shallow copy) for the
      * collection representing all the nodes in the graph. */
     public Collection<node_data> getV();
     /** Return a reference (shallow copy) for the
      * collection representing all the edges getting out of
      * the given node (src) */
     public Collection<edge_data> getE(int src);
     /** Delete the node (with the given ID) from the graph */
     public node_data removeNode(int key);
     /** Delete the edge from the graph */
     public edge_data removeEdge(int src, int dest);
     /** return the number of nodes in the graph. */
     public int nodeSize();
      * return the number of edges (directional graph). */
     public int edgeSize();
/** return the Mode Count for testing changes in the graph. */
     public int getMC();
}

/** This interface represents the set of operations applicable
on a node (vertex) in a (directional) weighted graph. */
public interface node_data {
     /** Return the key associated with this node. */
     public int getKey();
}

/** This interface represents the set of operations applicable
on a directional edge(src,dest) */
public interface edge_data {
     /** @return the id of the source node of this edge. */
     public int getSrc();
     /** @return the id of the destination of this edge. */
     public int getDest();
     /** @return the weight of this edge (positive value). */
     public double getWeight();
}
```