הטכניון – מכון טכנולוגי לישראל
הפקולטה להנדסת תעשיה וניהול
הנדסת תוכנה חורף תשפ"ב

פתרון מבחן מסכם מועד ב – הנדסת תוכנה 094219

מרצה: ד"ר אחמד ג'בארה

מתרגל אחראי: מר דביר טויטו

תאריך: 09.03.2022

תעודת זהות: _____

חדר בחינה: _____

הנחיות לנבחנת:

- ❑ מומלץ לקרוא את כל המבחן לפני תחילת הפתרון.
- ❑ חלק מן השאלות מכילות בסופן הערות והכוונות. מומלץ לא להתחיל לפתור שאלה לפני שקראת את כל סעיפיה.
- ❑ משך המבחן: 3 שעות.
- ❑ במבחן יש 7 שאלות. עלייך לענות על כולן.
- ❑ משקל כל שאלה רשום לידה.
- ❑ אין חומר עזר אשר מותר לשימוש.
- ❑ יש לפתור את המבחן בעט כחול או שחור. ניתן להשתמש בעפרון לצורך סרטוט התרשימים בלבד.
- ❑ יש לענות כל השאלות במחברת הבחינה פרט לשאלת ה-UML עליה יש לענות על גבי טופס הבחינה.
- ❑ יש להקפיד על כתב יד ברור.
- ❑ בשאלות בהן נדרש לסרטט תרשים, יש להקפיד על בהירותו ועל כיווני החיצים.
- ❑ חלק מקטעי הקוד עושים שימוש במחלקות סטנדרטיות של Java. ניתן להניח כי בוצע ייבוא כנדרש.
- ❑ ההנחיות במבחן מנוסחות בלשון נקבה מטעמי נוחות בלבד, אך מתייחסות לנבחנים ולנבחנות כאחד.

בהצלחה!

שאלה 1 (20 נקודות)

לפנייך מספר סעיפים אשר אינם קשורים אחד לשני. יש לענות על כולם.

סעיף א (10 נקודות)

לפנייך הגדרה של המחלקות Foo ו-Bar:

```
class Foo {
    public Foo() {
        f();
        g();
        h();
    }

    private void f() {
        System.out.println("Method f from Foo");
        g();
    }

    public void g() {
        System.out.println("Method g from Foo");
        h();
    }

    public static void h() {
        System.out.println("Method h from Foo");
    }
}
```

```
class Bar extends Foo {
    public Bar() {
        f();
    }

    public void f() {
        System.out.println("Method f from Bar");
    }

    public void g() {
        super.g();
        System.out.println("Method g from Bar");
    }

    public static void h() {
        System.out.println("Method h from Bar");
    }
}
```

מהו הפלט של הפעולה הראשית הבאה? הסבירי את תשובתך.

```
class FooBar {
    public static void main(String[] args) {
        Foo b = new Bar();
    }
}
```

פתרון

בבנאי של המחלקה Bar מתבצעת קריאה מרומזת לבנאי של המחלקה Foo. לכן בעת יצירת העצם b יתבצע הבנאי של מהמחלקה Foo. בשורה הראשונה מתבצעת קריאה לפעולה f. מאחר והיא פעולה פרטית, היא לא נדרסת ולכן תתבצע הפעולה אשר מוגדרת במחלקה Foo, והשורה הראשונה שתודפס היא Method f from Foo. הפעולה f משתמש בפעולה g אשר נדרסה, ולכן תתבצע הפעולה של המחלקה Bar. בפעולה זו מתבצעת קריאה לפעולה המוגדרת במחלקת האב, ולכן יודפס Method g from Foo ולאחר מכן תיקרא הפעולה h. מאחר שזו פעולה סטטית תתבצע הפעולה h אשר מוגדרת במחלקה Foo ויודפס Method h from Foo. נחזור לפעולה g של המחלקה Bar ויודפס Method g from Bar. לאחר מכן נחזור לבנאי של המחלקה Foo ותתבצע הקריאה לפעולה g. מאותן סיבות כמו הקריאה הקודמת לפעולה g תתבצע הפעולה של המחלקה Bar. לבסוף, תתבצע הקריאה לפעולה h של המחלקה Foo אשר תדפיס Method h from Foo, ובבנאי של המחלקה Bar תתבצע קריאה לפעולה f של המחלקה Bar ויודפס Method f from Bar.

הפלט הכולל אשר מתקבל:

Method f from Foo

Method g from Foo

Method h from Foo

Method g from Bar

Method g from Foo

Method h from Foo

Method g from Bar

Method h from Foo

Method f from Bar

<u>סעיף ב (4 נקודות)</u>

לפנייך המחלקה `SomeClass` בה מוגדרות הפעולות `doSomething` ו-`makeSomething`:

```
class SomeClass {
    public void doSomething() { /* Some code... */ }

    public final void makeSomething() { /* Some code... */ }
}
```

הסבירי את ההבדל בין הכותרות של שתי הפעולות ואת ההשלכות של האופן בו הן מוגדרות על המחלקות אשר יירשו מן המחלקה `SomeClass`. בתשובתך ציייני בעבור כל פעולה האם מתבצע עבורה קישור סטטי (Static Binding) או קישור דינמי (Dynamic Binding).

<u>פתרון</u>

הפעולה `makeSomething` מוגדרת כ-`final` ולכן לא ניתן יהיה לדרוס אותה במחלקות אשר יורשות מן המחלקה `SomeClass`. לפעולה זאת מתבצע קישור סטטי, שכן המהדר יכול לדעת בזמן ההידור לאיזו פעולה מתבצעת הקריאה.

לעומתה, את הפעולה `doSomething` ניתן לדרוס במחלקות אשר יורשות מן המחלקה `SomeClass`. על כן בעת ההידור המהדר אינו יכול לקבוע לאיזו לפעולה מתבצעת הקריאה ולכן מתבצע קישור דינמי עבור הפעולה.

סעיף ג (6 נקודות)

האם המחלקות הבאות עוברות הידור? במידה וכן, הסבירי מדוע. במידה ולא, הסבירי מדוע וציייני מהן השגיאות אשר קיימות בקוד (אם יש יותר מאחת, עלייך לציין את כולן).

```
class MyFirstException extends Exception { }
class MySecondException extends MyFirstException { }

class A { }
class B extends A { }
class C extends B { }

interface MyInterface {
    A doSomething();
}

class SuperClass {
    public B doSomething() throws MyFirstException {
        System.out.println("Doing Something in SuperClass...");
        return new C();
    }
}

class DerivedClass extends SuperClass implements MyInterface {
    public C doSomething() throws MySecondException {
        System.out.println("Doing Something in DerivedClass...");
        return new C();
    }
}
```

פתרון

המחלקות אינן עוברות הידור.

ראשית, המחלקה SuperClass תקינה מאחר ועל פי עקרון ההחלפה של ליסקוב ניתן להחזיר עצם מטיפוס C בפעולה doSomething מאחר והמחלקה C יורשת מן המחלקה B.

המחלקה DerivedClass אינה עוברת הידור, מאחר ובפעולה doSomething שבממשק MyInterface לא מוצהרת זריקה של חריגה ומאחר ו-MySecondException הנה חריגה מסומנת לא ניתן להצהיר על זריקתה בעת דריסת הפעולה במחלקה DerivedClass.

שימי לב שניתן לשנות את טיפוס ההחזרה של הפעולה doSomething מן המחלקות A ו-B למחלקה C על ידי שימוש ב-Covariant Return Type, ולכן שינוי זה אינו מהווה שגיאה.

בנוסף, שימי לב כי ניתן לשנות את ההצהרה על החריגה הנזרקת בפעולה doSomething מן החריגה MyFirstException לחריגה MySecondException ולכן גם שינוי זה אינו מהווה שגיאה.

שאלה 2 (11 נקודות)

נתונות המחלקות Node ו-Helper:

```
class Node {
    private double info;
    private Node next;

    public Node(double info, Node next) {
        this.info = info;
        this.next = next;
    }

    public Node(double info) {
        this(info, null);
    }

    public double getInfo() {
        return info;
    }

    public Node getNext() {
        return next;
    }
}
```

```
class Helper {
    private int num = 0;

    public void increase() {
        num++;
    }

    public int getNum() {
        return num;
    }
}
```

בנוסף, נתונה הפעולה הסטטית calculateSomething:

```
public static double calculateSomething(Node list, Helper helper) {
    if (list == null) {
        return 0;
    }

    double value = calculateSomething(list.getNext(), helper);

    int num = helper.getNum();
    helper.increase();

    return (list.getInfo() + value * num) / (num + 1);
}
```

סעיף א (6 נקודות)

מה הפעולה calculateSomething מחזירה בעבור כל אחת משלוש הקריאות הבאות?

```
calculateSomething(new Node(1), new Helper());
calculateSomething(new Node(1, new Node(3)), new Helper());
calculateSomething(new Node(1, new Node(2, new Node(3, new Node(4)))), new Helper());
```

הסבירי את תשובתך על ידי ביצוע מעקב עבור כל אחת מן הקריאות.

סעיף ב (5 נקודות)

בהינתן רשימה מקושרת כלשהי list, הסבירי במילים מה הפעולה calculateSomething מחזירה בעבור הקריאה הבאה:

```
calculateSomething(list, new Helper());
```

פתרון

בעבור הקריאה הנתונה, הפעולה מחשבת את ממוצע האיברים שנמצאים ברשימה המקושרת.

על כן, בעבור הקריאה הראשונה של סעיף א' יוחזר הערך 1.0, בעבור הקריאה השנייה יוחזר הערך 2.0 ובעבור הקריאה השלישית יוחזר הערך 2.5.

שאלה 3 (25 נקודות)

בטכניון קיימים קורסים רבים. לכל קורס שומרים את שמו, מספרו, את מרצה הקורס, את כל הסטודנטים אשר לוקחים את הקורס ומשתנה המציין האם ניתן פקטור לציוני הסטודנטים בקורס.

בעבור כל מרצה שומרים את שמו, גילו, המייל הטכניוני שלו, את המשכורת אותה הוא מקבל ואת הוותק שלו באקדמיה.

בעבור כל סטודנט שומרים את שמו, גילו, המייל הטכניוני שלו, את ממוצע הציונים שלו ואת הציון שלו בקורס.

סעיף א (5 נקודות)

כתבי מחלקות אשר מייצגות מרצה וסטודנט. במחלקות יש לכלול תכונות מתאימות ובנאים אשר מקבלים את ערכי כל התכונות ומאתחלים אותן בהתאם.

פתרון

מאחר למרצה וסטודנט יש תכונות משותפות, נאגד אותן במחלקה אחת:

```
class Person {
    protected String name;
    protected int age;
    protected String mail;

    public Person(String name, int age, String mail) {
        this.name = name;
        this.age = age;
        this.mail = mail;
    }
}
```

מחלקת מרצה:

```
class Lecturer extends Person {
    private int salary;
    private int seniority;

    public Lecturer(String name, int age, String mail, int salary, int seniority) {
        super(name, age, mail);
        this.salary = salary;
        this.seniority = seniority;
    }
}
```

מחלקת סטודנט:

```
class Student extends Person {
    private int grade;
    private double average;

    public Student(String name, int age, String mail, int grade, double average) {
        super(name, age, mail);
        this.grade = grade;
        this.average = average;
    }

    public int getGrade() {
        return grade;
    }
}
```

טעות נפוצה בסעיף זה הייתה לא ליצור את מחלקת Person, מה שהוביל לשכפול קוד.

<u>סעיף ב (4 נקודות)</u>

כתבי מחלקה אשר מייצגת קורס. במחלקה יש לכלול תכונות מתאימות וכן יש לכלול בנאי אשר מקבל את שם הקורס, מספרו, המרצה ומשתנה המציין האם ניתן פקטור ומאתחל את התכונות בהתאם.

<u>פתרון</u>

```
class Course {
    private String name;
    private String courseNumber;
    private Lecturer lecturer;
    private List<Student> students;
    private boolean factor;

    public Course(String name, String courseNumber, Lecturer lecturer, boolean factor) {
        this.name = name;
        this.courseNumber = courseNumber;
        this.lecturer = lecturer;
        this.factor = factor;
        students = new ArrayList<>();
    }
}
```

סעיף ג (6 נקודות)

בצעי את השינויים הנדרשים במחלקות כך שניתן יהיה לבצע העתקה עמוקה של קורס על ידי שימוש בבנאי העתקה.

פתרון

על מנת שנוכל לבצע העתקה עמוקה לקורס, יהיה עלינו לבצע העתקה למרצה הקורס ולסטודנטים. לכן, תחילה נוסיף בנאי העתקה למחלקת המרצה ולמחלקת הסטודנט.

```
class Lecturer extends Person {
    /* Code from part A */

    public Lecturer(Lecturer lecturer) {
        this(lecturer.name, lecturer.age, lecturer.mail, lecturer.salary,
             lecturer.seniority);
    }
}


class Student extends Person {
    /* Code from part A */

    public Student(Student student) {
        this(student.name, student.age, student.mail, student.grade, student.average);
    }
}
```

נוסיף בנאי העתקה למחלקת הקורס:

```
class Course {
    /* Code from part B */

    public Course(Course course) {
        this(course.name, course.courseNumber, course.lecturer, course.factor);
        lecturer = new Lecturer(course.lecturer);
        students = new ArrayList<>();

        for (Student student : course.students) {
            students.add(new Student(student));
        }
    }
}
```

טעות נפוצה בסעיף זה הייתה לממש במחלקה את הממשק Cloneable ולדרוס את הפעולה clone במקום להגדיר בנאי העתקה כפי שנדרש בשאלה.

סעיף ד (10 נקודות)

על מנת לאפשר לחשב בקלות סטטיסטיקות על ציוני הסטודנטים בקורסים לצורך הצגתם באתר "מזלג-גבינה" (CheeseFork) נשיא הטכניון רוצה לאפשר מעבר על הסטודנטים אשר רשומים לקורס מסוים באמצעות לולאת foreach.

לצורך חישוב הסטטיסטיקות בנוחות, הנשיא מעוניין בכך שבעת סריקת הסטודנטים בקורס יוצגו רק סטודנטים אשר הציון הסופי שלהם באותו הקורס גדול (או שווה) מציון סף כלשהו. ציון הסף ייקבע לפני הסריקה, ולא ישתנה במהלכה.

לשם כך, תחילה עלייך להגדיר במחלקה Course פעולה בשם setThresholdGrade אשר מקבלת מספר שלם. המספר אשר מועבר לפעולה קובע את ציון הסף הנוכחי עבור הסריקות העתידיות של הסטודנטים בקורס עד לקריאה הבאה לפעולה.

לאחר מכן, עלייך ליצור מחלקה בשם CourseIterator אשר מממשת את הממשק Iterator<Student>. יש לממש במחלקה זו את הפעולות hasNext ו-next בהתאם לציון הסף.

לבסוף, עלייך לממש במחלקה Course את הממשק Iterable<Student> ואת הפעולה iterator אשר מוגדרת בממשק.

הערות לכל סעיפי השאלה:

- יש לתכנן את המחלקות על פי עקרונות תכנות מונחה עצמים נכונים.
- ניתן להשתמש בפעולות get ו-set לתכונות המחלקות מבלי לממש אותן.
- אין צורך להקפיד על ביצוע import.

פתרון

מחלקת האיטרטור:

```
class CourseIterator implements Iterator<Student> {
    private List<Student> students;
    private int thresholdGrade;
    private int nextIndex;

    public CourseIterator(List<Student> students, int thresholdGrade) {
        this.students = students;
        this.thresholdGrade = thresholdGrade;
        nextIndex = calculateNextIndex(0);
    }
```

```
    private int calculateNextIndex(int startIndex) {
        int nextIndex = startIndex;
        boolean found = false;

        for (int i = startIndex; i < students.size() && !found; i++) {
            if (students.get(i).getGrade() >= thresholdGrade) {
                nextIndex = i;
                found = true;
            }
        }
        return nextIndex;
    }

    @Override
    public Student next() {
        Student nextStudent = students.get(nextIndex);
        nextIndex = calculateNextIndex(nextIndex + 1);
        return nextStudent;
    }

    @Override
    public boolean hasNext() {
        return nextIndex < students.size();
    }
}
```

השינויים במחלקת קורס:

```
class Course implements Iterable<Student> {
    /* All fields from part A */
    private int thresholdGrade;

    public Course(String name, String courseNumber, Lecturer lecturer, boolean factor) {
        /* Constructor from part A */
        thresholdGrade = 0;
    }

    public void setThresholdGrade(int thresholdGrade) {
        this.thresholdGrade = thresholdGrade;
    }

    @Override
    public Iterator<Student> iterator() {
        return new CourseIterator(students, thresholdGrade);
    }
}
```

קוד השאלה בשלמותו:

```
class Person {
    protected String name;
    protected int age;
    protected String mail;

    public Person(String name, int age, String mail) {
        this.name = name;
        this.age = age;
        this.mail = mail;
    }
}

class Lecturer extends Person {
    private int salary;
    private int seniority;

    public Lecturer(String name, int age, String mail, int salary, int seniority) {
        super(name, age, mail);
        this.salary = salary;
        this.seniority = seniority;
    }

    public Lecturer(Lecturer lecturer) {
        this(lecturer.name, lecturer.age, lecturer.mail, lecturer.salary,
             lecturer.seniority);
    }
}

class Student extends Person {
    private int grade;
    private double average;

    public Student(String name, int age, String mail, int grade, double average) {
        super(name, age, mail);
        this.grade = grade;
        this.average = average;
    }

    public Student(Student student) {
        this(student.name, student.age, student.mail, student.grade, student.average);
    }

    public int getGrade() {
        return grade;
    }
}
```

```java
class Course implements Iterable<Student> {
    private String name;
    private String courseNumber;
    private Lecturer lecturer;
    private List<Student> students;
    private boolean factor;
    private int thresholdGrade;

    public Course(String name, String courseNumber, Lecturer lecturer, boolean factor) {
        this.name = name;
        this.courseNumber = courseNumber;
        this.lecturer = lecturer;
        this.factor = factor;
        students = new ArrayList<>();
        thresholdGrade = 0;
    }

    public Course(Course course) {
        this(course.name, course.courseNumber, course.lecturer, course.factor);
        lecturer = new Lecturer(course.lecturer);
        students = new ArrayList<>();

        for (Student student : course.students) {
            students.add(new Student(student));
        }
    }

    public void setThresholdGrade(int thresholdGrade) {
        this.thresholdGrade = thresholdGrade;
    }

    @Override
    public Iterator<Student> iterator() {
        return new CourseIterator(students, thresholdGrade);
    }
}

class CourseIterator implements Iterator<Student> {
    private List<Student> students;
    private int thresholdGrade;
    private int nextIndex;

    public CourseIterator(List<Student> students, int thresholdGrade) {
        this.students = students;
        this.thresholdGrade = thresholdGrade;
        nextIndex = calculateNextIndex(0);
    }
```

```
    private int calculateNextIndex(int startIndex) {
        int nextIndex = startIndex;
        boolean found = false;

        for (int i = startIndex; i < students.size() && !found; i++) {
            if (students.get(i).getGrade() >= thresholdGrade) {
                nextIndex = i;
                found = true;
            }
        }
        return nextIndex;
    }

    @Override
    public Student next() {
        Student nextStudent = students.get(nextIndex);
        nextIndex = calculateNextIndex(nextIndex + 1);
        return nextStudent;
    }

    @Override
    public boolean hasNext() {
        return nextIndex < students.size();
    }
}
```

שאלה 4 (15 נקודות)

במסגרת תואר ראשון בטכניון, כל סטודנט חייב לבצע פרויקט גמר כחלק מחובותיו להשלמת התואר. פרויקט יכול להיות מוצע ע"י סטודנט או ע"י מנחה – איש מקצוע ותיק המלווה את הסטודנט בביצוע הפרויקט, מעריך את עבודת הסטודנט ונותן ציונים על ההגשות השונות במהלך חיי הפרויקט.

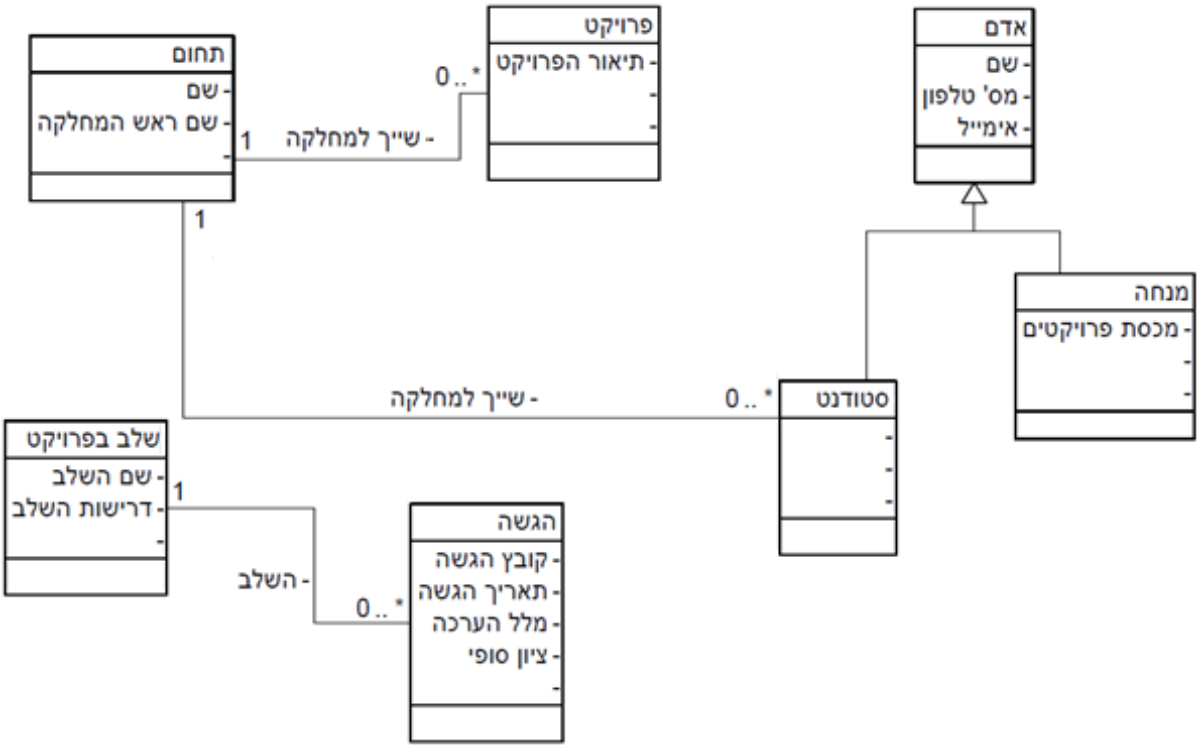
כל פרויקט מבוצע ע"י סטודנט אחד לפחות ולכל היותר ע"י חמישה סטודנטים. לכל פרויקט יש מנחה אחד בדיוק. האדם אשר הציע את הפרויקט לאו דווקא ייקח בו חלק – פרויקט לא בהכרח יבוצע ע"י הסטודנט שהציע אותו או יונחה ע"י המנחה שהציע אותו.

בכל פרויקט ישנם ארבעה שלבים ובסיום כל שלב הסטודנטים אשר מבצעים את הפרויקט נדרשים להגיש מסמך אשר מסכם את הנעשה באותו השלב. המסמכים מוגשים למנחה של הפרויקט אשר בודק ומעריך אותם.

סטודנט אשר ביצע פרויקט ונכשל בביצועו יכול לבצע פרויקט אחר בשנה לאחר מכן כל עוד לא עבר את רף הניסיונות המוגבל אשר נקבע על ידי מנהלת הפרויקטים בטכניון.

לכל סטודנט יש תחום מחקר מסוים וכל פרויקט משויך לתחום מסוים. מנחה יכול להנחות פרויקטים מכל תחום. עם זאת, לכל מנחה קיימת מכסת פרויקטים אשר מגבילה את מספר הפרויקטים אותם הוא יכול להנחות במקביל במהלך שנת לימודים.

להלן מידול התחלתי של המערכת לניהול פרויקטים המוצגת על ידי תרשים UML:



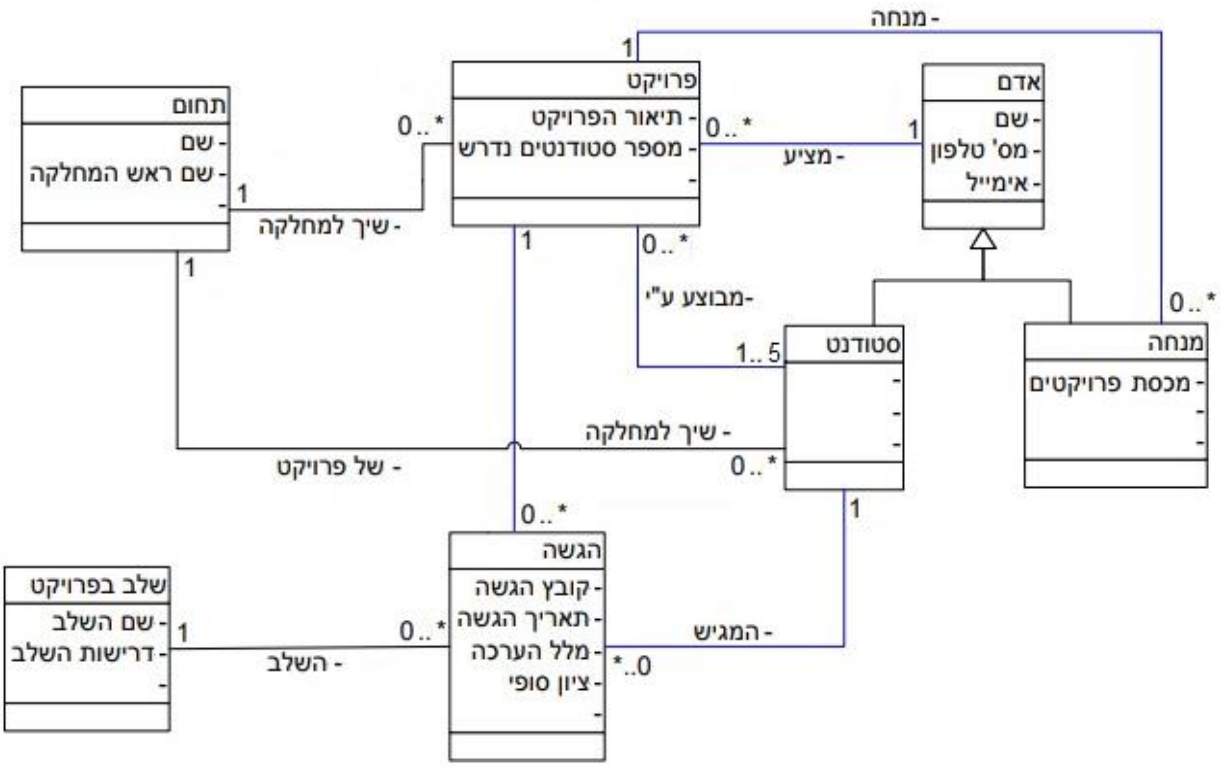
עלייך להוסיף לתרשים **אך ורק** את המידע הבא באמצעות יחסים, ריבויים (Multiplicity), תפקידים (Roles), מחלקות ותכונות:

- מי המציע של כל פרויקט, מי הסטודנטים המבצעים את הפרויקט ומי מנחה אותו.
- מספר הסטודנטים המדויק אשר נדרשים לביצוע פרויקט מסוים.
- אילו הגשות הוגשו בפרויקט ומי העריך ונתן ציון להגשה של הפרויקט.
- אילו פרויקטים ביצע סטודנט ואילו הוא הגשות ביצע.

שימי לב:

- הוספת מחלקה / תכונה / יחס מיותרים או לא נכונים עשויה להוריד נקודות.
- אין חובה לרשום מאפיינים אשר מייצגים יחס אך הוספתם לא תוריד ניקוד.
- אל תמהרי להוסיף מחלקות. עדיף להוסיף יחס או מאפיין במידה והם עונים על הדרישה.
- יחס יכול להיות יחס פשוט (Association), הכלה חזקה (Composition), הכלה חלשה (Aggregation) והורשה (Inheritance).
- לכל יחס שאינו יחס הורשה יש לציין את הריבוי (Multiplicity) בכל צד שלו ולפחות תפקיד (Role) אחד.
- התבליטים הריקים אשר מופיעים בכל מחלקה אינם מעידים על כך שעלייך להוסיף תכונות למחלקה וכן הם אינם מעידים על מספר התכונות של המחלקה.
- יש לענות על שאלה זאת **על גבי טופס הבחינה**.
- לנוחיותך, בדף האחרון של הבחינה מופיע עותק נוסף של התרשים. ניתן להיעזר בעותק זה לצורך ביצוע סקיצות של התרשים. על התשובה הסופית להופיע בשאלה זו ולא על גבי העותק.

פתרון


מנחה
1
פרויקט
תיאור הפרויקט
מספר סטודנטים נדרש
0..*
מציע
1
אדם
שם
מס' טלפון
אימייל
תחום
שם
שם ראש המחלקה
1
שיך למחלקה
0..*
1
0..*
מבוצע ע"י
1..5
סטודנט
מנחה
מכסת פרויקטים
0..*
שיך למחלקה
של פרויקט
0..*
1
0..*
הגשה
קובץ הגשה
תאריך הגשה
מלל הערכה
ציון סופי
שלב בפרויקט
שם השלב
דרישות השלב
1
0..*
השלב
המגיש
0..*

שאלה 5 (10 נקודות)

לפנייך מספר מחלקות אשר מרכיבות מערכת לשליחת וקבלת הודעות בתוכנית מרובת תהליכונים. בתוכנית ישנם משתמשים אשר שולחים הודעות ומשתמשים אשר מקבלים אותן. משתמש אשר שולח הודעות נקרא sender ומשתמש אשר מקבל הודעות נקרא receiver. כל sender שולח מקבץ של הודעות (פקטות) אשר מסתיים בהודעה "End". על כל receiver לדעת שנשלחה אליו הודעה על מנת שיוכל לקרוא אותה, ועל כל sender לדעת כי ההודעה האחרונה ששלח נקראה על ידי ה-receiver על מנת שיוכל לשלוח את ההודעה הבאה בלי חשש מדריסת ההודעה הקודמת.

את תוכנית זו ראית בהרצאה העוסקת בתכנות מקבילי, אך בעת העתקתה לטופס הבחינה נמחקו מן הקוד כל השורות (או חלקים מן השורות) אשר תפקידן להפוך את התוכנית לבטוחה בעבודה עם מספר תהליכונים (Thread Safe).

בשאלה זו עלייך לעזור לנו להשיב את הקוד להיות בטוח על מנת שיעבוד באופן תקין, תוך מניעת מצבים של Busy Waiting – מצבים בהם תהליכון נמצא בלולאה ריקה ולא מבצע דבר עד אשר תנאי הלולאה יחדל להתקיים והתהליכון יצא מן הלולאה וימשיך לבצע את הקוד אשר נותר לו.

```
1.  class Data {
2.      private String packet;
3.
4.      private boolean transfer = true;
5.
6.      public String receive() {
7.          while (transfer) {
8.
9.          }
10.
11.         transfer = true;
12.         String returnPacket = packet;
13.
14.         return returnPacket;
15.     }
16.
17.     public void send(String packet) {
18.         while (!transfer) {
19.
20.         }
21.
22.         transfer = false;
23.         this.packet = packet;
24.     }
25. }
26.
27. class Sender implements Runnable {
28.     private Data data;
29.
30.     public Sender(Data data) {
31.         this.data = data;
32.     }
33.
```

```
34.     @Override
35.     public void run() {
36.         String packets[] = {
37.                 "First packet",
38.                 "Second packet",
39.                 "Third packet",
40.                 "Fourth packet",
41.                 "End"
42.         };
43.
44.         for (String packet : packets) {
45.             data.send(packet);
46.
47.             try {
48.                 Thread.sleep(ThreadLocalRandom.current().nextInt(1000, 5000));
49.             } catch (InterruptedException e) {
50.                 Thread.currentThread().interrupt();
51.                 System.err.println("Thread interrupted");
52.             }
53.         }
54.     }
55. }
56.
57. class Receiver implements Runnable {
58.     private Data load;
59.
60.     public Receiver(Data load) {
61.         this.load = load;
62.     }
63.
64.     @Override
65.     public void run() {
66.         for (String receivedMessage = load.receive(); !receivedMessage.equals("End");
67.              receivedMessage = load.receive()) {
68.             System.out.println(receivedMessage);
69.
70.             try {
71.                 Thread.sleep(ThreadLocalRandom.current().nextInt(1000, 5000));
72.             } catch (InterruptedException e) {
73.                 Thread.currentThread().interrupt();
74.                 System.out.println("Thread interrupted");
75.             }
76.         }
77.     }
78. }
79.
80. class Program {
81.     public static void main(String[] args) {
82.         Data data = new Data();
83.         Thread sender = new Thread(new Sender(data));
84.         Thread receiver = new Thread(new Receiver(data));
85.         sender.start();
86.         receiver.start();
87.     }
88. }
```

עלייך לבצע את התיקונים הנדרשים בקוד כך שהוא יהפוך לבטוח לעבודה עם מספר תהליכונים, וימנע Busy Waiting.

שימי לב כי אין צורך בכתיבת כל הקוד מחדש. מספיק לציין אילו שינויים היית עורכת בשורה מסוימת או איזה קטעי קוד היית מוסיפה בין שתי שורות קוד קיימות.

פתרון

הקוד לאחר ההוספות (ההוספות ממורקרות בצבע צהוב):

```
class Data {
    private String packet;

    private boolean transfer = true;

    public synchronized String receive() {
        while (transfer) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        transfer = true;
        String returnPacket = packet;
        notifyAll();
        return returnPacket;
    }

    public synchronized void send(String packet) {
        while (!transfer) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        transfer = false;
        this.packet = packet;
        notifyAll();
    }
}

class Sender implements Runnable {
    private Data data;

    public Sender(Data data) {
        this.data = data;
    }
```

```
    public void run() {
        String packets[] = {
                "First packet",
                "Second packet",
                "Third packet",
                "Fourth packet",
                "End"
        };

        for (String packet : packets) {
            data.send(packet);

            try {
                Thread.sleep(ThreadLocalRandom.current().nextInt(1000, 5000));
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                System.err.println("Thread interrupted");
            }
        }
    }
}

class Receiver implements Runnable {
    private Data load;

    public Receiver(Data load) {
        this.load = load;
    }

    public void run() {
        for (String receivedMessage = load.receive(); !receivedMessage.equals("End");
             receivedMessage = load.receive()) {
            System.out.println(receivedMessage);

            try {
                Thread.sleep(ThreadLocalRandom.current().nextInt(1000, 5000));
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                System.out.println("Thread interrupted");
            }
        }
    }
}

class Program {
    public static void main(String[] args) {
        Data data = new Data();
        Thread sender = new Thread(new Sender(data));
        Thread receiver = new Thread(new Receiver(data));
        sender.start();
        receiver.start();
    }
}
```

שאלה 6 (12 נקודות)

לפנייך קטע קוד אשר מכיל פעולה ראשית המוגדרת במחלקה Main. מיד לאחריו מופיע חלק מקוד המקור של המחלקה הגנרית Vector.

```
1 public class Main {
2     public static void main(String[] args) {
3         Vector<Integer> v = new Vector<Integer>();
4         for (int i = 1; i <= 12; i++) {
5             v.add(i);
6             System.out.println(v.size() + ": " + v.capacity());
7         }
8     }
9 }
```

קוד המקור:

```
1  public class Vector<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, java.io.Serializable
2  {
3      protected Object[] elementData;
4      protected int elementCount;
5      protected int capacityIncrement;
6      private static final int MAX_ARRAY_SIZE = Integer.MAX_VALUE - 8;
7      public Vector(int initialCapacity, int capacityIncrement) {
8          super();
9          if (initialCapacity < 0)
10             throw new IllegalArgumentException("Illegal Capacity: "+
11                                                initialCapacity);
12         this.elementData = new Object[initialCapacity];
13         this.capacityIncrement = capacityIncrement;
14     }
15     public Vector(int initialCapacity) {
16         this(initialCapacity, 0);
17     }
18     public Vector() {
19         this(10);
20     }
21     public synchronized boolean add(E e) {
22         modCount++;
23         ensureCapacityHelper(elementCount + 1);
24         elementData[elementCount++] = e;
25         return true;
26     }
27     private void ensureCapacityHelper(int minCapacity) {
28         if (minCapacity - elementData.length > 0)
29             grow(minCapacity);
30     }
31     private void grow(int minCapacity) {
32         int oldCapacity = elementData.length;
33         int newCapacity = oldCapacity + ((capacityIncrement > 0) ?
34                                          capacityIncrement : oldCapacity);
35         if (newCapacity - minCapacity < 0)
36             newCapacity = minCapacity;
37         if (newCapacity - MAX_ARRAY_SIZE > 0)
38             newCapacity = hugeCapacity(minCapacity);
39         elementData = Arrays.copyOf(elementData, newCapacity);
40     }
41 }
```

סעיף א (8 נקודות)

מהו הפלט אשר מתקבל בעת הרצת הפעולה הראשית? הסבירי את תשובתך וצייני באופן מפורש את מספרי השורות של המחלקה Vector אשר התבצעו והובילו לפלט זה.

הערות:

- הפעולות `size` ו-`capacity` לא נתונות והינך אמורה לדעת מה הן מבצעות גם ללא קריאת הקוד שלהן.
- בקוד מופיעות שורות אשר אינן מסבירות את הפלט. אינך נדרשת לדעת מה שורות אלו עושות ואינך נדרשת להסבירן.

פתרון

הפלט אשר מתקבל:

1: 10

2: 10

3: 10

4: 10

5: 10

6: 10

7: 10

8: 10

9: 10

10: 10

11: 20

12: 20

הסבר לפלט:

Creating the vector:

The `main` method creates a vector object via constructor chaining: Line 3 of the `main` calls the following lines (methods) in the Vector class: Default constructor (line 18) will be called. This constructor will call another constructor (line 15) which will call a third constructor (line 7).

Adding values:

The `for` loop will call 12 times the `add` method (line 21) of Vector. The `add` method will call the `ensureCapacityHelper` method (line 27). This method will call `grow` in case there is a need for more room. The grow method will set the new capacity of the vector and call the `copyOf` method to copy the old data to the new vector.

<u>סעיף ב (4 נקודות)</u>

האם תשובתך לסעיף א' תשתנה במידה ונשנה את שורה 3 בקוד הנתון לשורה הבאה?

```
Vector<Integer> v = new Vector<>(10, 10);
```

במידה וכן, צייני אילו פלטים ישתנו והסבירי את תשובתך. במידה ולא, הסבירי מדוע.

<u>פתרון</u>

הפלט לא היה משתנה, מאחר ובשני המקרים הקיבולת של הווקטור מאותחלת לערך 10 ובעת הוספת האיבר ה-11 היא גדלה ב-10 והופכת ל-20.

שאלה 7 (7 נקודות)

אחת מתבניות העיצוב עליה דנו בקורס היא סינגלטון (Singleton). להלן חלק ממערכת אשר משתמש בתבנית עיצוב זו:

```
class Parent implements Cloneable {
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}

class Singleton extends Parent {
    private static final Singleton instance = new Singleton();

    private Singleton() { }

    public static Singleton getInstance() {
        return instance;
    }
}
```

סעיף א (3 נקודות)

במערכת קיימת בעיה באופן שבו תבנית העיצוב ממומשת. הסבירי מהי הבעיה וספקי דוגמת קוד אשר מנצלת את בעיה זאת ושוברת את המטרה לשמה תבנית העיצוב נוצרה.

פתרון

מאחר והמחלקה Singleton יורשת מן המחלקה Parent אשר מממשת את הממשק Cloneable ודורסת את הפעולה clone, ניתן יהיה להעתיק מופעים של המחלקה Singleton וזאת בסתירה לתבנית העיצוב, אשר מהותה להגביל את יצירת המופעים מן המחלקה למופע בודד.

דוגמת קוד:

```
public static void main(String[] args) throws CloneNotSupportedException {
    Singleton s1 = Singleton.getInstance();
    Singleton s2 = (Singleton) s1.clone();
}
```

סעיף ב (4 נקודות)

מנהלי המערכת פנו אלייך בבקשה לעזרה בתיקון הבעיה אשר תיארת בסעיף א'. באפשרות המנהלים לשנות **רק** את המחלקה Singleton אך עליהם לשמור על יחסי ההורשה אשר קיימים במערכת.

הציעי דרך לעזור למנהלים לתקן את הבעיה, וממשי את הצעתך.

פתרון

נדרוס את הפעולה clone במחלקה Singleton ונזרוק בה את החריגה CloneNotSupportedException ובכך נמנע שימוש בפעולה.

```
class Singleton extends Parent {
    private static final Singleton instance = new Singleton();

    private Singleton() { }

    public static Singleton getInstance() {
        return instance;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException();
    }
}
```

עותק נוסף של תרשים המחלקות: