חלק 2

<u>סעיפי המסמך</u>

1. רקע (המוטיבציה) למטלה
2. מטרת המטלה
3. דרישות המימוש למחלקות: ***Task*** ו-***CustomExecutor***
4. הנחיות נוספות
5. נספח א' - ה-enum ***TaskType***
6. נספח ב' – המתודה ()partialTest לבדיקה חלקית של המימוש שלכם (וכוללת רמזים)

<u>רקע</u>

ב-Java אין אפשרות מובנית לתת עדיפות למשימה אסינכרונית (משימה שתתבצע ב-Thread נפרד).
השפה אמנם מאפשרת לקבוע עדיפות ל-Thread שמריץ את משימה, אך לא למשימה עצמה.
לכן, אנו בבעיה כאשר נרצה לתעדף משימה אסינכרונית תוך שימוש ב:

1. ממשק Callable<V>. מדובר בממשק שמייצג משימה עם ערך החזרה גנרי.
משימה שהיא סוג של Callable<V> לא ניתנת לביצוע במסגרת Thread רגיל, ולכן לא ניתן לקבוע לה עדיפות באופן עקיף.
2. ב-ThreadPool אשר שמים בתור המשימות שלו Runnable או Callable<V>, שכן לא ניתן לקבוע עדיפות ל-Thread ספציפי באוסף ה-Threads של ב-Executor.

<u>מטרת המטלה</u>

ליצור טיפוס חדש שמייצג משימה אסינכרונית עם עדיפות וסוג של ThreadPool חדש אשר תומך במשימות בעלות עדיפות. ראשית, נתאר באופן כללי את **<u>שתי</u>** המחלקות ולאחר מכן נציג את הדרישות בצורה מפורטת:

<u>המחלקה *Task* –</u> מייצגת פעולה שניתן להריץ באופן אסינכרוני ויכולה להחזיר ערך מסוג כלשהו (כלומר, יוגדר. כטיפוס החזרה גנרי). אין הכרח שהפעולה תצליח ובמקרה של כישלון, תיזרק חריגה (Exception).
<u>המחלקה ***CustomExecutor* –**</u> מייצגת טיפוס חדש של ThreadPool שתומך בתור של משימות בעלות עדיפות (כל משימה בתור היא מסוג Task). *CustomExecutor* ייצור Task טרם הכנסתו לתור על-ידי העברה של Callable<V> ו-enum מסוג TaskType. *CustomExecutor* ייבצע את המשימות בהתאם לעדיפות שלהן.
TaskType הוא enum שמתאר את סוג המשימה (חישובית/גישה ל-I/O/לא ידוע) והעדיפות שלה על-סמך הערך המספרי של סוג המשימה. **<u>עליכם להעתיק את *TaskType* מהסיפא של המסמך. אין לשנות את המימוש.</u>**
** הערה: המילים "פעולה" ו-"משימה" הן בעלות אותה משמעות במסמך זה.

<u>דרישות</u>

בחלק הבא מוצג הסבר על המחלקות והדרישות במימוש.
שימו לב! בחלק מהסעיפים החתימות של המתודות, בין היתר וטיפוסי החזרה לא יינתנו לכם באופן מפורש.
<u>מדובר במרכיב מהותי בפתרון המטלה ובהבנת החומר. לרשותכם גם רמזים ב-test בנספח ב'.</u>

<u>המחלקה Task – פעולה שניתן להריץ באופן אסינכרוני עם עדיפות</u>

1. מכילה TaskType שמצייג את סוג המשימה. לכל סוג יש ערך מספרי את אשר קובע את עדיפות המשימה
2. תכיל **<u>בין היתר</u>** מתודה עם ערך החזרה גנרי. במידה ולא ניתן לבצע את הפעולה תיזרק חריגה (Exception)
3. תכלול מתודת Factory שתיצור מופעים של המחלקה
4. לצורך יצירה של Task צריך להעביר:
טיפוס מובנה ב-Java של משימה שניתן לבצע באופן אסינכרוני עם ערך החזרה **או** משימה שניתן לבצע באופן אסינכרוני עם ערך החזרה, בתוספת TaskType . לכן עבור האפשרות הראשונה, אתם תקבעו מהו ה-TaskType כברירת מחדל
5. מופעים של המחלקה Task צריכים להיות ברי-השוואה בהתאם לסוג שלהם
6. עליכם לקבוע **<u>בעצמכם</u>** את חתימת המחלקה, חתימות המתודות ושדות המידע של המחלקה
7. חשבו אלו מתודות צריכות להופיע בכל טיפוס חדש שאנו יוצרים כדי לתמוך בפעולות שמתוארות במסמך ובמקרה שירצו להרחיב את המחלקות שניצור או פשוט להשתמש בהן

<u>המחלקה *CustomExecutor* – סוג של ThreadPool המריץ פעולות מסוג Task אסינכרונית בהתאם לעדיפות</u>

1. תכריז על מתודה להגשה של מופעים מסוג Task לתור משימות עם עדיפות
2. תכריז על מתודה שמטרתה להגיש לתור פעולה שניתן לבצע באופן אסינכרוני בתוספת TaskType
3. תכריז על מתודה שמטרתה להגיש לתור פעולה שניתן לבצע באופן אסינכרוני ללא TaskType כפרמטר
4. המתודות בסעיף 2 ו-3 חייבות לעשות שימוש במתודה מסעיף 1
5. מספר ה-threads המינימלי באוסף ה-threads ב-***CustomExecutor*** יהיה מחצית מספר המעבדים שזמינים לטובת ה-Java Virtual Machine
6. מספר ה-threads המקסימלי באוסף ה-threads ב-***CustomExecutor*** יהיה מספר המעבדים שזמינים לטובת ה-Java Virtual Machine פחות 1
7. ***CustomExecutor*** יתחזק תור משימות אשר מסדר את האלמנטים בתור בהתאם לעדיפות שלהם, מהנמוכה לגבוהה בכל רגע נתון
8. Thread עודף הינו thread שהוסף באופן דינאמי, כך שמספרו באוסף ה-threads ב-***CustomExecutor*** גדול מהמספר המינימלי וקטן (או שווה) למספר המקסימלי שמופיעים בסעיפים 4 ו-5
9. פרק הזמן שמותר ל-thread עודף להיות ללא משימה (מצב idle) צריך להיקבע ל-0.3 שניות (300 מילישניות).
10. ממשו את המתודה ()getCurrentMax אשר תחזיר את העדיפות הגבוהה ביותר של משימה שנמצאת כעת בתור ב-O(1) מבחינת סיבוכיות זמן ריצה וסיבוכיות מקום. **אתם לא רשאים לגשת לתור בעת הקריאה למתודה זו**
11. המחלקה צריכה לאפשר להפסיק את פעילותו של מופע מסוג ***CustomExecutor*** באופן הבא:
    a. לא לאפשר הכנסה של משימות נוספות לתור
    b. ביצוע כל המשימות שנותרו בתור
    c. סיום של כל המשימות שנמצאות כעת בביצוע באוסף ה-threads של ה-***CustomExecutor***

<u>הנחיות נוספות</u>

- אתם רשאים להיעזר בכל המשאבים בדף הקורס ב-Moodle, בפרט קבצי המקור שכתבנו בהרצאות
- חשבו בתהליך הפיתוח על עקרונות ה-Object-Oriented Design שלמדנו כגון SOLID
- חשבו האם יש constructors שצריכים עם הרשאה לא פומבית
- השתמשו בהכלה/ירושה/מימוש ממשק בהתאם לצורך
- עליכם לצרף דיאגרמת מחלקות עבור הפתרון. ניתן לבצע זאת באופן אוטומטי דרך IntelliJ
- עליכם להעלות את הפתרון שלכם לתיבה במודל. בנוסף תבצעו הגשה פרונטלית למטלה, במהלך השבוע לאחר ההגשה בזמן התרגול.

נספח א' – ***TaskType***

לרשותכם ה-enum ***TaskType*** שמתאר את סוג המשימה והעדיפות שלה (על-סמך הערך המספרי של סוג המשימה):

```
public enum TaskType {
  COMPUTATIONAL(1){
    @Override
    public String toString(){return "Computational Task";}

  },
  IO(2){
    @Override
    public String toString(){return "IO-Bound Task";}
  },
  OTHER(3){
    @Override
    public String toString(){return "Unknown Task";}
  };

  private int typePriority;

  private TaskType(int priority){
    if (validatePriority(priority)) typePriority = priority;
    else
      throw new IllegalArgumentException("Priority is not an integer");
  }

  public void setPriority(int priority){
    if(validatePriority(priority)) this.typePriority = priority;
    else
      throw new IllegalArgumentException("Priority is not an integer");
  }

  public int getPriorityValue(){
    return typePriority;
  }

  public TaskType getType(){
    return this;
  }

  /**
   * priority is represented by an integer value, ranging from 1 to 10
   * @param priority
   * @return whether the priority is valid or not
   */
  private static boolean validatePriority(int priority){
    if (priority < 1 || priority >10) return false;
    return true;
  }
}
```

נספח ב' – Test חלקי כדי שתבדקו את המימוש שלכם. כולל רק חלק מהבדיקות שנערוך

```
import org.junit.jupiter.api.Test;
import org.junit.platform.commons.logging.Logger;
import org.junit.platform.commons.logging.LoggerFactory;
import java.util.concurrent.*;

public class Tests {
    public static final Logger logger = LoggerFactory.getLogger(Tests.class);
    @Test
    public void partialTest(){
        CustomExecutor customExecutor = new CustomExecutor();
        var task = Task.createTask(()->{
            int sum = 0;
            for (int i = 1; i <= 10; i++) {
                sum += i;
            }
            return sum;
        }, TaskType.COMPUTATIONAL);

        var sumTask = customExecutor.submit(task);

        final int sum;
        try {
            sum = sumTask.get(1, TimeUnit.MILLISECONDS);
        } catch (InterruptedException | ExecutionException | TimeoutException e) {
            throw new RuntimeException(e);
        }

        logger.info(()-> "Sum of 1 through 10 = " + sum);

        Callable<Double> callable1 = ()-> {
            return 1000 * Math.pow(1.02, 5);
        };

        Callable<String> callable2 = ()-> {
            StringBuilder sb = new StringBuilder("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
            return sb.reverse().toString();
        };

        // var is used to infer the declared type automatically
        var priceTask = customExecutor.submit(()-> {
            return 1000 * Math.pow(1.02, 5);
        }, TaskType.COMPUTATIONAL);

        var reverseTask = customExecutor.submit(callable2, TaskType.IO);

        final Double totalPrice;
        final String reversed;
        try {
            totalPrice = priceTask.get();
            reversed = reverseTask.get();
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
        logger.info(()-> "Reversed String = " + reversed);
        logger.info(()->String.valueOf("Total Price = " + totalPrice));

        logger.info(()-> "Current maximum priority = " +
customExecutor.getCurrentMax());
        customExecutor.gracefullyTerminate();
```

```
    }
}
```

המילה השמורה var מאפשרת להסיק באופן אוטומטי את טיפוס המשתנה בהתאם לטיפוס האובייקט.
ברגע שתממשו את המחלקות והמתודות שלכם, תוכלו להחליף את המילה var בטיפוס אחר ובדקו האם אתם לא מקבלים שגיאות

פלט חלקי:

```
Dec 26, 2022 12:28:56 AM Tests partialTest
INFO: Sum of 1 through 10 = 55

Dec 26, 2022 12:28:56 AM Tests partialTest
INFO: Reversed String = ZYXWVUTSRQPONMLKJIHGFEDCBA

Dec 26, 2022 12:28:56 AM Tests partialTest
INFO: Total Price = 1104.0808032

Dec 26, 2022 12:28:56 AM Tests partialTest
INFO: Current maximum priority = 2
```