**Threads:**

ישנן 2 אופציות שבעזרתן אפשר להשתמש בטרדים אם נרצה להתשמש בריצה שלהם בצורה שונה מהרגיל (דיפולט) :

1) ליצור מחלקה בשם כולשהו שיורשת את טרד למשל:

class Runner extends Thread {

ואז במיין ליצור ישויות שלו

2) ליצור מחלקה שתממש את הממשק ראנבל, למשל:

class Runner implements Runnable {

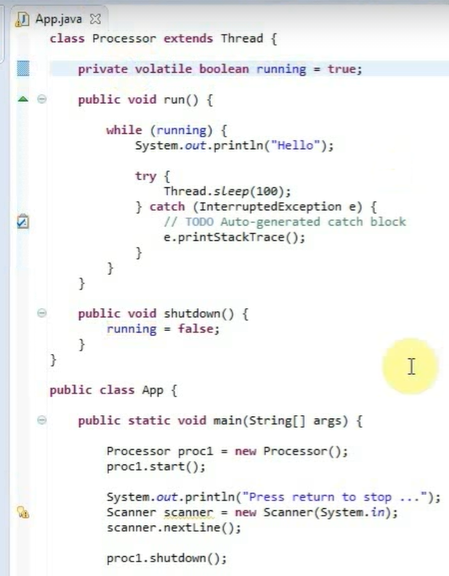
Runner runner1 = new Runner();יוצר טרד מהמחלקה ראנר (יורשת טרד) בשם ראנן1

Thread a = new Thread();יוצר טרד מהמחלקה טרד (מובנת בגווא'ה) בשם א'

runner1.*sleep*(1000);גורם לטרד ללכת לישון לשניה

**Volatile** – מסדר את הבעיה הבאה : (מוסיפים למשנה המדובר)

נגדיר משתנה שאנו רוצים לשנות במהלך ריצה, אך ישנו טרד שרץ לפי ערכו של משנה זה, בנוסף החילוף אינו מתרחש בריצתו של הטרד המדובר אלא בריצתו של טרד אחר שתפקידו להחליף את המשתנה



למשל :

MAIN – משנה את ערכו של המשתנה

Processor – מחלקה שהיא הורשה של טרד שעליה אנו רצים

ואם אנו נחליף את המשנה אחרי שה (וויל (ראנינג ) ) החל יש סיכוי שהתוכנה לא תעצר, לכן אם נשים את הוויטיל הזה בהגדרת המשתנה נפתור את הבעיה ותמיד התוכנה תעצר כאשר נלחץ על משהו (ההגדרה של התוכנה הזו )

* t1.join(); - גויין תפקידו לגרום להמשך של הפעולות שלאחר הגויין לחכות עד אשר טרד טי1 יסיים את חלקו , ( מחייב טרי אנד קאץ )
* public synchronized void increment() {} –
* סנקרונייז תפקידו לאפשר לטרד אחד כל פעם לקחת את המפתח והטרד הבא שמנסה לקחת את המפתח יוכל לקחת רק כאשר מי שלקח את המפתח החזיר כלומר סיים את הפונקציה .

ExecutorService executor = Executors.*newFixedThreadPool*(2);

מחלקה בגוואה שקשורה לטרדים,

טרד פול זה כמו מפעל ליצור עובדים כלומר, מייצר מספר טרדים במקרה הזה (2) , וכל עובד הוא טרד וברגע שהוא מסיים את העבודה הוא עובד למשימה הבאה.

executor.awaitTermination(10, *TimeUnit*.***DAYS***);

מגדיר כמה זמן לחכות עד אשר ימשיך את הפעולות שנמצאות אחרי כמו JOIN בסוגרים מוגדר כמה לחכות עד אשר ימשיך, האיבר הראשון בסוגריים כמה זמן במספר 10 במקרה שלנו והערך השני איזה סוג שניות/ דקות / ימים כלומר הוא יחכה 10 ימים עד אשר ימשיך לפעולות.

CountDownLatch latch = new CountDownLatch(3);

מחלקה בגוואה שקשורה לטרדים,

נותן למספר טרדים שנבחר (במקרה שלנו 3 ) , לעשות פעולות ויחכה עד אשר יגיע ל0 כלומר יסיימו את הפעולות שלהם (הם יחכו עד אשר יגיע ל0 ואז ימשיכו )

latch.await();

await מחכה שהcountdown יגיע ל-0 (כלומר כל הטרדים יגמרו),

דורש try & catch.

private BlockingQueue <Integer> queue =

new ArrayBlockingQueue <Integer>(10);

blocking queue הוא מבנה מידע שיכול להכיל data items מסוג שאבחר ב ArrayList , התור עובד בצורה שאתה יכול להוסיף ולהורידitems בשיטת FIFO ( הראשון שנכנס הוא הראשון שנוציא )

wait();

wait היא פונקציה שיש לכל object (מכוון שהיא במחלקה שלו) כדי להשתמש בwait נצטרך להיות בתוך {} של synchronized.

תפקידו של הwait הוא לוקח פיקוד על המפתח של synchronized שבו הוא נמצא וישן עד שישחררו אותו עם .Notify

אפשר להכניס בתוך הסוגרים של ה wait מספר ואז הוא יחכה את המספר ששרשמנו במילי'

lock.wait(); - ככה עושים wait על אויבקט שיצרתי לפני ששמו lock

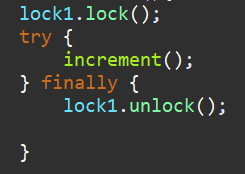
notify();

Notify היא פונקציה שיש לכל object (מכוון שהיא במחלקה שלו) כדי להשתמש Notify נצטרך להיות בתוך {} של synchronized.

תפקידו של Notify הוא משחרר/ מעיר את הטרד שנמצא ב wait עם **אותו** מפתח שעליו יש את ה synchronized.

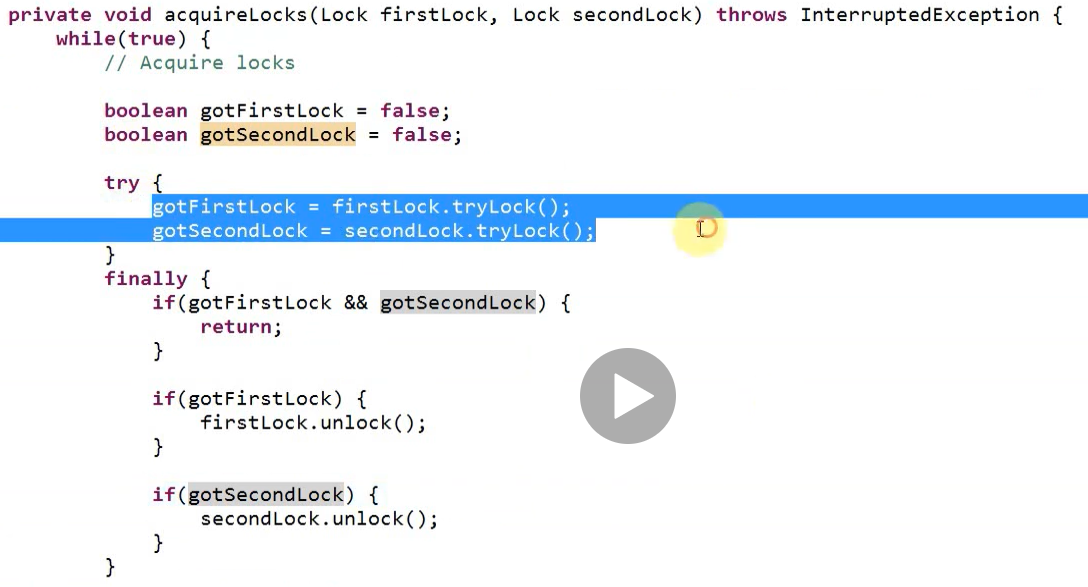
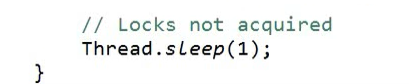
private Lock lock1 = new ReentrantLock();

דרך ReentrantLock נוכל לבצע לקיחת מפתח lock1.lock(); והחזרתו lock1.unlock(); בלי synchronized . יבצע את אותו דבר כמו synchronized רק בלעדיו .

נבצע כך :

lock1.tryLock();

שיטה של ReentrantLock שתפקידה לנסות לקחת את המפתח ולהחזיר אם הצליחה או לא (בוליאן) אם הצליחה אז היא לוקחת באמת את המפתח. טוב לשימוש אם נרצה ליצור פונקציה ללקיחת מספר מפתחות בו זמנית בלי להגיע לדד לוק אם יש כמה טרדים שמנסים לקחת את המפתחות בו זמנית )



Semaphore sem = new Semaphore(1);

אוביקט שתפקידו לשמש כמו ספירה לתרדים בסוגרים נרשום כמו מפתחות פנויים

sem.acquire();

תפקידו לקחת את המפתח ולכן אם התחלנו עם מספר מפתחות 1 עכשיו מספר המפתחות ירד ל-0.

sem.release();

תפקידו לשחרר מפתח לכן אם מספר המפתחות היה 0 לאחר ה sem.acquire(); עכשיו מספר המפתחות יעלה ל-1.

**Strings :**

s.substring (2, 5)– יוצא קלט מאות מס 2 עד אות מס' 4 ,התו הראשון נמצא במיקום ה-0

s.substring(1) - יוצא קלט מאות מס 2 עד אות האחרונה

s.equals(s1) – בודק אם אס1 שווה לאס2 ומחזיר נכון לא נכון

s.compareTo(s1) - בדוק את המחרוזות לפי סדר לקסיקוגרפי - מחזיר 0 אם שוות , מספר חיובי אם המילה אס ארוכה יותר מהמילה אס1 ושלילי אם המילה אס קצרה יותר מאס1 .

(המספר זה לפי מספר האותיות מעבר , המספר יכול הלשתנות גם לפי הסדר הלקסיקוגרפי )

str.substring(str.length()-1) – מחזיר את התו האחרון של המחזרות

**Array :**

int or [] = new int (6); – אתחול מערך מסוג אינט בגודל 6

int or [] = {1,3,45,6,3}; – אתחול מערך עם המספרים שהכנסנו

int or [][] = { {1,3,45,6,3} , {1}; – אתחול מערך עם מספר מימדים

**For Loop :**

For (int x: or ) – עובר על כל הערכים במערך אור

**Random number Generator :**

תחילה נצטרך לייבא את החבילה import java.util.Random;

Random RN = new random(); – יוצר אובייקט רדומנלי שנקרא רן

Number1 = RN.nextInt(6); – מגריל לתוך נמבר1 מספר מ-0 עד 5 !

Number2 = 1+RN.nextInt(6); – מגריל לתוך נמבר2 מספר מ-1 עד 6 !

**math class methods :**

Math.abs (-26.7)– יוצא קלט של המספר בערך מוחלט כלומר 26.7 (ולא מינוס )

Math.ceil (26.7)– יוצא קלט של המספר מעוגל כלפי מעלה כלומר 27

Math.floor (26.7)– יוצא קלט של המספר מעוגל כלפי מטה כלומר 26

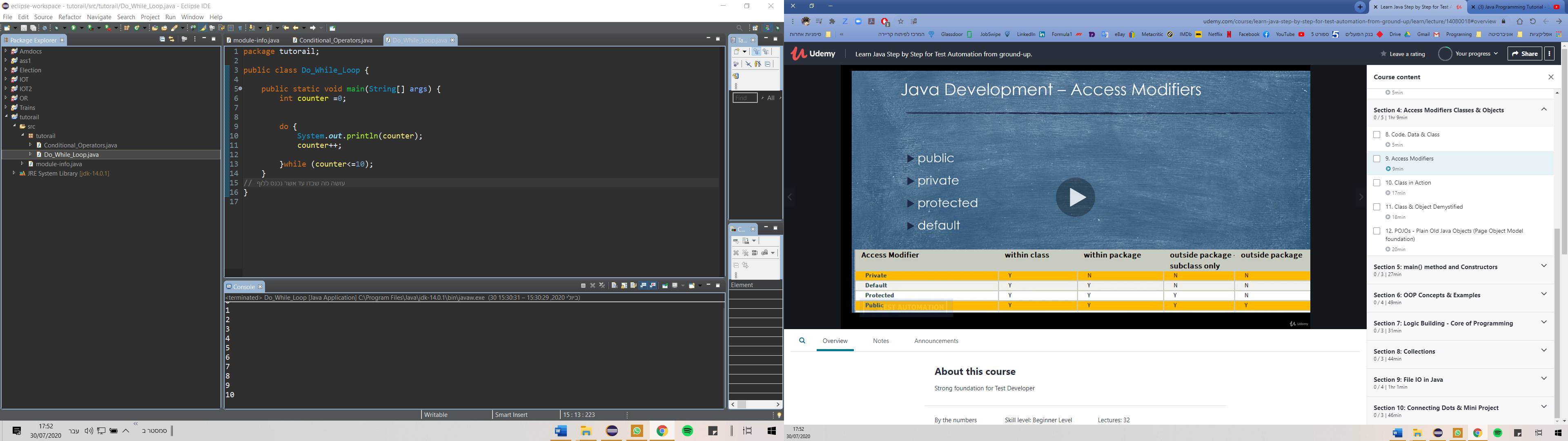
Math.max (2, 5)– מחזיר את המקסימום מבין המספרים כלומר 5

Math.min (2, 5)– מחזיר את מינימום מבין המספרים כלומר 2

Math.pow (5, 3)– מחזיר 5 בחזקת 3 כלומר 125

Math.sqrt (9)– מחזיר שורש המספר כלומר 3

**הרשאות גישה:**



**Question for interview :**

**לעשות רוורס / לסדר הפוך – מערך של תווים :**

Q1: reverse array of chars:

class Solution {

public void helper(char[] s, int left, int right) {

if (left >= right) return;

char tmp = s[left];

s[left++] = s[right];

s[right--] = tmp;

helper(s, left, right);

}

public void reverseString(char[] s) {

helper(s, 0, s.length - 1);

}

}

**לעשות רוורס / לסדר הפוך – מספר אינט :**

Q2: reverse int :

class Solution {

public int reverse(int x) {

int rev = 0;

while (x != 0) {

int pop = x % 10;

x /= 10;

if (rev > Integer.MAX\_VALUE/10 || (rev == Integer.MAX\_VALUE / 10 && pop > 7)) return 0;

if (rev < Integer.MIN\_VALUE/10 || (rev == Integer.MIN\_VALUE / 10 && pop < -8)) return 0;

rev = rev \* 10 + pop;

}

return rev;

}

}

**למצוא את האות שחוזרת במילה פעם אחת ולהחזיר את האינדקס שלה**

**(אם יש כמה את האינדקס הנמוך, אם אין להחזיר -1):**

Q3: Given a string, find the first non-repeating character in it and return its index. If it doesn't exist, return -1. : HashMap

class Solution {

public static int firstUniqChar(String s) {

HashMap <Character , Integer> ans = new HashMap <Character , Integer> ();

for (int i=0 ; i<s.length() ; i ++){

if (ans.containsKey(s.charAt(i))){

ans.put(s.charAt(i), ans.get(s.charAt(i))+1);

}else{

ans.put(s.charAt(i), +1);

}

}

if (ans.containsValue(1)){

for (int i=0 ; i<s.length() ; i ++){

if (ans.get(s.charAt(i))==1){

return i;

}

}

}

return -1;

}//firstUniqChar

}

**למצוא אם המילים הן אנגרם:**

Q4: is anagram?

class Solution {

public boolean isAnagram(String s, String t) {

int value=0;

int[] str\_1=new int[256];

int[] str\_2=new int[256];

for (int i = 0; i < s.length(); i++) {

value=s.charAt(i);

str\_1[value]++;

}

for (int i = 0; i < t.length(); i++) {

value=t.charAt(i);

str\_2[value]++;

}

if (Arrays.equals(str\_1,str\_2)){

return true;

}

return false;

}

}

**למצוא אם המילה היא פולינדרום :**

Q5: is Palindrome?

class Solution {

public boolean isPalindrome(String s) {

if(s==null){

return false;

}

s = s.toLowerCase();

int i=0;

int j=s.length()-1;

while(i<j){

while(i<j && !((s.charAt(i)>='a' && s.charAt(i)<='z')

|| (s.charAt(i)>='0'&&s.charAt(i)<='9'))){

i++;

}

while(i<j && !((s.charAt(j)>='a' && s.charAt(j)<='z')

|| (s.charAt(j)>='0'&&s.charAt(j)<='9'))){

j--;

}

if(s.charAt(i) != s.charAt(j)){

return false;

}

i++;

j--;

}

return true;

}

}