מתי נשתמש בתרדים? אני רוצה להרוג תהליך מהמלשינה הבאת מבחינת השרת
ולתעדכן בינהן או שיריבר תוך כדי. אני לא רוצה לחכות שהוא יתחבר.
להסבר תמיד בפורום של stop? אי אפשר לבטל את ה stop? רק Thread
אחר שלא מתחיל.

Thread implements Runnable

start → מתודה אסינכרונית שמה משימה בתור
run → מריץ את הפונקציה

ה שיטות שלנו:

← extends Thread ומממש את run - להריץ בתוכו את הקוד
הא אסינכרוני (אפשר לרשת רק מחלקה אחת ב Java)
← implement Runnable
← להשתמש בהלפטר. Thread t = new Thread (new
TaskRunnable (new MyTask());
או בעצם מחלקה נוספת שמממש את המחלקה של t.start
← מנפק אובייקט מ Thread בתוך הפונקציה שלנו

מתודות שהן סינכרוניות בתרדים: אם טרד נכנס למתודה. אף טרד אחר
לא יכול להכנס לאותו האובייקט לכל מתודה שהיא סינכרונית.

Active Object - הנוסף ל מתודה באובייקט לתת אסינכרוני
והוא מתלבט אתה להריץ עבורנו

BlockingQueue<Runnable> x = new LinkedBlockingQueue<>();

ואנחנו במחלקה בונה נגדיר אותו ונפעיל תהליך שירות אותו
באופן הבא:

```
Constructor ()
{ new Thread(new Runable() {
        public run() {
            while (true)
                q.take().run();
        }).start();
```

ובמקרה שלנו פשוט נוסיף לתור תמיד. add ← יכניס תמיד
take ← שיכניס לעיבוד מחדש.

כשאלי בתקף מה זה double check lock:

שימוש בסינגלטון - רק מוסיף אתה לאובייקט.
שיטת 1. לשים סינכרון בפונקציה - אבל זה אומר שתמיד יפעלו
2. לשאול אבי אם עוד לאותר מכן לשים סינכרוני
על חלק הקוד הספציפי. דוגמה:

```
if helper == null
```

```
synchorined(this)
    if helper == null
        helper = new helper()
return helper.
```

← אתחול פעם אחת

* פרט חשוב! ליצור משתנה מסוג המחלקה בסינקגיה + קונסטרקטור שהוא private

volitiel - מוסיף יתרון למשתנה

Thread pool - הרבה טרדים לא בודד
מתי נשתמש? אם נרצה כמה לחזיק אותם
ולהשתמש ב reuse

Executer - צד המחלקה
יש כמה אפשרויות:

Fixed (מספר)
single (אחד)
:

executer service מכיל

execute (runnable)

subit (callable)

- Callabale

submit - מחזיר לך Future<worker>

```
<V Future<V> submit(Callable<V> c)
{
    Future<V> f = new Future<>();
    execute(() -> f.set(c.call()));
    return f;
}
```

Atomic var = מאפשר לנו לבדוק מבלי יותר ע"י כך שאי אפשר לבדוק ולעדכן במשתנה ב טרדים במקביל.

trylock = ReentrantLock
unlock

ArrayBlockingQueue
ConcurrentHashMap
ConcurrentLinkedQueue

- Completable future

```
CompletableFuture.supplyAsync(() -> {
    .thenApply
    .thenAccept;
```

נשתמש בזה לקשר דברים אחרכך!

Atomic - רק לאמולטי טרדים. מקבר לצרכ אין סיבה.

לינוי הנלבה