משימה לתרגול מחלקות וירושות

הערה:

- בונוס: ניתן לממש את המשימה בtypescript (למי שמכיר) !
- נא לקרוא את כל המשימה, ואז לבצע תכנון לפני שמתחילים לממש בקוד!
- יש להקפיד על פונקציונאליות נכונה ובדיקות unit tests בעזרת jest
- נא לכתוב טסטים מתאימים לממשק לפני שמתחילים לכתוב קוד!
(נא להתמקד בכיסוי טסטים מלא, ופחות בכמויות מופרזות של טסטים זהים)
- הקפידו על קוד קצר קריא ונקי ולא על מימוש רב שורות מיותרות שניתן לחסוך בכתיב קצר.

משימה א: תרגיל המחלקה Time

בנאי:

שמקבל בפרמטרים את הדברים הבאים: ({ hours, minutes, seconds })

אם לא סופקו אף אחד מהפרמטרים, יש להזין את הזמן הנוכחי .

אם סופק לפחות אחד מהפרמטרים, יש להזין את הפרמטרים ומה שלא הוזן - לאפס אותו ל-0

- טווח הזמן התקין הוא מ: 99:59:59- (זמן שלילי) עד 99:59:59 (זמן חיובי, גם מעל 24 שעות)

למשל:

```
const time1 = new Time({hours: 14});    console.log(time1.toString())    // 14:00:00
const time2 = new Time();               console.log(time2.toString())    // 13:48:25
```

פונקציות ווירטואליות:

set seconds(seconds){ /* do something */ }	// update the seconds
set minutes(minutes){ /* do something */ }	// update the minutes
set hours(hours){ /* do something */ }	// update the hours
get seconds(){ /* do something */ }	// return the seconds
get s(){ /* do something */ }	// return the seconds
get minutes (){ /* do something */ }	// return the minutes
get m (){ /* do something */ }	// return the minutes
get h (){ /* do something */ }	// return the hours
get hours (){ /* do something */ }	// return the hours
get totalSeconds() { /* do something */}	// return the current time in seconds, // like: 00:01:30 => 90 seconds

פונקציות נוספות:

addSeconds(seconds) { /* do something */}	// add seconds to current seconds
removeSeconds(seconds) { /* do something */}	// remove seconds from current seconds
resetSeconds() { /* do something */}	// reset the current seconds to zero
addMinutes(minutes) { /* do something */}	// add minutes to current minutes
removeMinutes(minutes) { /* do something */}	// remove minutes from current minutes
resetMinutes() { /* do something */}	// reset the current minutes to zero
addHours(hours) { /* do something */}	// add hours to current hours
removeHours(hours) { /* do something */}	// remove hours from current hours
resetHours() { /* do something */}	// reset the current hours to zero
reset() { /* do something */}	// reset the time to: 00:00:00

addTime(time) {/* do something */}
Extra: implement '+' operator
	// add time to your current time,
	// for example: time1.addTime(time2);
	// 14:00:00 + 13:48:25 = 27:48:25
	// time after 99:59:59 will be stay 99:59:59

subTime(time) {/* do something */}
Extra: implement '-' operator
	// subtract time to your current time,
	// for example: time1.subTime(time2);
	// 14:00:00 - 13:48:25 = 00:11:35
	// time below 00:00:00 will be negative
	// like: 13:48:25 – 14:00:00 = - 00:11:35
	// time below -99:59:59 will be stay -99:59:59

toString(format = 'HH:MM:SS') { /* do something*/}

// return formatted time string.

// According the format parameter, default is HH:MM:SS for example 13:48:25 .

// The format can change like: 'HHh MMm SSs' for example '13h 48m 25s'

הערות עזר לתכנון המחלקה Time :

כדאי לממש ממשק אחיד, שידע לתקשר בין כל יחידות המידה השונות: שעות, דקות, שניות, (שהוא השניות) ממליץ בחום כדי לשמור על פשטות המחלקה, לשמור במופע רק את כמות השניות סהכ של כל הזמן, ואז את החישובים לבצע מול השניות האלה, למשל:

```
const t = new Time({hours: 0, minutes: 2, seconds: 5}); // total 125 seconds
t.addMinutes(5); // total 125seconds + 5minutes(5*60=300seconds) = 125+300=425 seconds
t.minutes; // parseInt(425seconds / 60) = 7minutes
```

מומלץ ונכון לממש פונקציות עזר מתאימות כמו למשל converters לדוגמה:

```
hoursToSeconds(hours);                    totalSecondsToSeconds(totalSeconds);
minutesToSeconds(minutes);                totalSecondsToMinutes(totalSeconds);
                                          totalSecondsToHours(totalSeconds);
```

השתמשו במתודה **padStart(2, '0')**.'string' כדי לבצע הוספת 0 במידה והמספר חד ספרתי.

משימה ב: תרגיל המחלקה Clock היורשת ממחלקה Time

בנאי:

שמקבל בפרמטרים את הדברים הבאים: ({ hours, minutes, seconds, autoStart = true })

ברגע שהבנאי אותחל עם השדה autoStart = true השעון יתעדכן כל שניה

למשל:

```
const clock1 = new Clock({hours: 14, autoStart = false});
    console.log(clock1.toString())      // 14:00:00
    console.log(clock1.toString())      // 14:00:00
clock1.start();
    console.log(clock1.toString())      // 14:00:01
const clock2 = new Clock();
    console.log(clock2.toString())      // 13:48:25
    console.log(clock2.toString())      // 13:48:26
clock2.pause()
    console.log(clock2.toString())      // 13:48:26
```

פונקציות וירטואליות:

לתמוך בכל הפונקציות הווירטואליות שבמשימה א'

פונקציות נוספות:

toString(format = 'HH:MM:SS') { /* do something*/}	// same from the parent class
start() { /* do something */}	// starting ticker (interval) the clock every seconds
pause() { /* do something */}	// stopping the clock ticker (interval)
reset() { /* do something */}	// same from the parent class

משימה ג: תרגיל המחלקה Stopper היורשת ממחלקה Time

בנאי:

שמקבל בפרמטרים את הדברים הבאים: (autoStart = false)

ברגע שהבנאי אותחל עם השדה autoStart = true הסטופר ירוץ ויתעדכן כל שניה

למשל:

```
const stopper1 = new Stopper();
    console.log(stopper1.toString())    // 00:00:00
    console.log(stopper1.toString())    // 00:00:00
stopper1.start()
    console.log(stopper1.toString())    // 00:00:01
    console.log(stopper1.toString())    // 00:00:02

const stopper2 = new Stopper(true);
    console.log(stopper2.toString())    // 00:00:01
    console.log(stopper2.toString())    // 00:00:02
    console.log(stopper2.toString())    // 00:00:03
stopper2.pause()
    console.log(stopper2.toString())    // 00:00:03
stopper2.start()
    console.log(stopper2.toString())    // 00:00:04
stopper2.stop()
    console.log(stopper2.toString())    // 00:00:04
stopper2.start()
    console.log(stopper2.toString())    // 00:00:01
stopper2.start()
    console.log(stopper2.toString())    // 00:00:02
stopper2.stop()
stopper2.reset()
    console.log(stopper2.toString())    // 00:00:00
    console.log(stopper2.toString())    // 00:00:00
```

פונקציות וירטואליות:

לתמוך בכל הפונקציות הווירטואליות שבמשימה א'

פונקציות נוספות:

toString(format = 'HH:MM:SS') { /* do something*/}	// same from the parent class
start() { /* do something */}	// starting ticker (interval) the clock every seconds

pause() { /* do something */}
reset() { /* do something */}
	// pause the clock ticker (interval)
	// **not** stopping the clock ticker (interval),
	// reset immediately

stop () { /* do something */}	// stopping the clock ticker (interval) // and reset on the next start and not immediately

משימה ד: תרגיל המחלקה Timer/Countdown היורשת ממחלקה Time

בנאי:

שמקבל בפרמטרים את הדברים הבאים: ({hours, minutes, seconds })

הבנאי חייב לקבל זמן כלשהו (אחד מהפרמטרים לפחות)

למשל:

```
const cb = () => { console.log('DONE!'); }
const counter1 = new Countdown({seconds: 3});
    console.log(counter1.toString())    // 00:00:03
counter1.start(cb)
    console.log(counter1.toString())    // 00:00:03
    console.log(counter1.toString())    // 00:00:02
    console.log(counter1.toString())    // 00:00:01
    console.log(counter1.toString())    // 00:00:00
                                        // DONE!
const counter2 = new Countdown ({hours: 3});
counter2.start(cb)
    console.log(counter2.toString())    // 03:00:00
    console.log(counter2.toString())    // 02:59:59
    console.log(counter2.toString())    // 02:59:58
counter2.pause()
    console.log(counter2.toString())    // 02:59:58
counter2.start(cb)
    console.log(counter2.toString())    // 02:59:57
counter2.stop()
    console.log(counter2.toString())    // 00:00:00
counter2.hours = 0;
counter2.seconds = 8;
counter2.start(cb);
    console.log(counter2.toString())    // 00:00:08
counter2.start();
    console.log(counter2.toString())    // 00:00:07
...
```

פונקציות וירטואליות:

לתמוך בכל הפונקציות הווירטואליות שבמשימה א'

פונקציות נוספות:

toString(format = 'HH:MM:SS') { /* do something*/}	// same from the parent class
start(cb) { /* do something */}	// getting callback function that will call when counter will done, starting ticker (interval) the timer every seconds, ignore from last cb when apply twice

pause() { /* do something */}
reset() { /* do something */}
	// pause the timer ticker (interval)
	// **not** stopping the timer ticker (interval),
	// reset immediately

stop () { /* do something */}	// stopping the timer ticker (interval) // and reset on the next start and not immediately