**סיכום כללי – oop typescript**

**הגדרות מונחים:**

**מחלקה** – **תבנית** ליצירת מופעים, המגדירה **טיפוס** שמכיל משתנים ופעולות.

ממחלקה אפשר ליצור מופעים על ידי המילה new והמופעים הנ"ל יווצרו בheap (יהיה להם מצביע מהstack)

**אובייקט** – מופע בודד של מחלקה (נוצר על ידי הבנאי)

**Class members** – פונקציות (פעולות) (בנאי הוא גם פונקציה) שמוגדרות בתוך המחלקה, ומשתנים (מאפיינים) שמוגדרים בתוך המחלקה

**הרשאת גישה** – הרשאות גישה יכולות להיות public / private , הגדרת המקומות מהם נוכל לגשת לclass member

**private** – הרשאת גישה , המגדירה את ההרשאה לגשת אל ה class member רק מתוך המחלקה הספציפית (לא דרך האובייקט, ולא דרך הנגזרת) – אם ניגש מחוץ למחלקה נקבל שגיאת קומפילציה

**public** - הרשאת גישה , המגדירה את ההרשאה לגשת אל ה class member מכל מקום (גם דרך המחלקה עצמה, גם דרך האובייקט, וגם דרך הנגזרת)

**static** – הגדרה ברמת המחלקה ולא ברמת האובייקט (המופע) הבודד, מאפשר לגשת ישירות לclass member הסטטי, ללא צורך ביצירת מופע

* מתוך המחלקה ניגש לclass member הסטטי דרך class name ולא דרך this
* ממחלקה יורשת ניגש לclass member הסטטי של הבסיס דרך class name ולא דרך this או super (רק במקרה של פונקציה super יעבוד)
* מחוץ המחלקה ניגש לclass member הסטטי דרך class name ולא דרך האובייקט

**readonly** – משתנה קבוע

* אם המשתנה לא סטטי - אפשר לתת לו ערך רק בשורת ההגדרה או בבנאי
* אם המשתנה סטטי - אפשר לתת לו ערך רק בשורת ההגדרה

(ראה דוגמא 01\_overview - oop ts )

**Ctor –** בנאי – פונקציה דיפולטיבית הנמצאת בצורה אוטומטית בכל מחלקה שניצור.

על ידי האופרטור new נוכל לגשת לבנאי וליצור בheap אובייקט חדש מטיפוס המחלקה.

הבנאי יוצר בפועל מופע בזיכרון ומחזיר את הכתובת של המופע שיצר.

יש אפשרות ליצור בצורה מפורשת בנאי בתוך המחלקה על ידי המילה השמורה constructor, אם נגדיר בנאי שמקבל ערכים נהיה חייבים לשלוח את הפרמטרים בקריאה של הnew

**new –** אופרטור המשמש לקריאה לבנאי של מחלקה, לשם יצירת אובייקט מהמחלקה

**Composition** – הכלה של אובייקט ממחלקה A בתוך מחלקה B (מחלקה B תמיד תצביע לאותו אובייקט – ללא שינוי המצביע)

**Aggregation** - הכלה של אובייקט ממחלקה A בתוך מחלקה B (מחלקה B יכולה לשנות את ההצבעה לאובייקטים שונים מסוג B)

**Inheritance** הורשה של מחלקה A למחלקה B (מחלקה B תמיד תכיל בתוכה את כל הclass members של מחלקת הבסיס)

* מחלקה A - מחלקת בסיס
* מחלקה - B מחלקת נגזרת
* יצירת אובייקט ממחלקה - B תקרא לבנאי של B ומשם בצורה מרומזת / מפורשת (ע"י super()) לבנאי של המחלקה A
* מהמחלקה הנגזרת אי אפשר לגשת לclass member המוגדרים כprivate בבסיס

**Multi level inheritance** - הורשה של מחלקה B שיורשת ממחלקה A למחלקה C (מחלקה C תמיד תכיל בתוכה את כל הclass members של A + B)

**Multi inheritance**—קבלת ירושה במחלקה C ממחלקה A וממחלקה B – דבר זה לא אפשרי ב typescript (וברוב שפות התכנות האחרות)

**Override** – דריסה של פונקציה ממחלקת הבסיס במחלקה הנגזרת. לשם כך יש לענות על הדרישות הבאות:

* הפונקציה בבסיס חייבת להיות עם אותו שם כמו הפונקציה בנגזרת
* הפונקציה בבסיס חייבת לקבל את אותו מספר פרמטרים שבפונקציה הדורסת (והפרמטרים חייבים להיות בעלי טיפוס זהה לפונקציה הדורסת)
* הפונקציה בבסיס חייבת להחזיר את אותו טיפוס שמוחזר מהפונקציה הדורסת

- **Abstract** מחלקה מופשטת (מחלקה בסיסית וראשונית ממנה נירש למחלקות ממשיות יותר), ממחלקה אבסרקטית אי אפשר ליצור מופעים, אבל אפשר להגדיר משתנה שיהיה מטיפוס המחלקה האבסרקטית ויוכל להצביע לכל אובייקט נגזרת (פולימורפיזם).

**Interface** – ממשק – מגדיר מעין "חוזה" שיכול להיות ממומש על ידי מחלקות

**InstanceOf**- אופרטור המשמש לבדיקה האם אובייקט מסויים הוא מופע של מחלקה מסויימת, או נגזרת של אותה מחלקה.

האופרטור יקבל בצד שמאל את שם האובייקט, ובצד ימין את שם המחלקה שנרצה לבדוק.

בתוך בלוק התנאי – התבצעה המרה מרומזת מהאובייקט אל טיפוס המחלקה שבדקנו (לא עובד על תא במערך – שם נצטרך לבצע במרה מפורשת)

**as** – down casting – המרה של אובייקט למחלקה אחרת

**<>** down casting- – המרה של אובייקט למחלקה אחרת

**toString** – פונקציה שמוגדרת למחלקה object

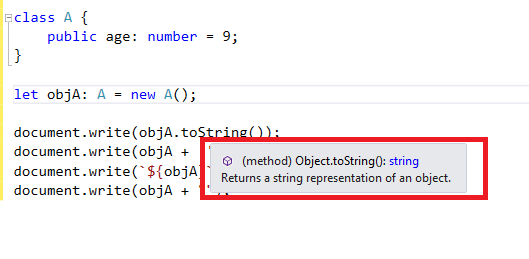
כל טיפוס בtypescript יורש בצורה אוטומטית מobject

ולכן אוטומטית לכל משתנה בtypescipt יש את הפונקציה toString

* עבור משתנה מטיפוס מספרי: יחזיר את המספר במחרוזת
* עבור משתנה מטיפוס מחרוזת: יחזיר את המחרוזת
* עבור משתנה מטיפוס בוליאני: יחזיר את הערך הבוליאני במחרוזת
* עבור משתנה מטיפוס מערך: יחזיר את תאי המערך מופרדים בפסיק
* עבור כל שאר האובייקטים: יחזיר את [object, Object]

יש אפשרות בתוך מחלקה לדרוס את הפונקציה toString כך שתחזיר מחרוזת (המחרוזת תייצג את האובייקט)

לדוגמה:



**סיכום הרשאות והורשות:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | מחלקה רגילה | מחלקה אבסרקטית | ממשק |
| **הגדרת משתנה private** | כן | כן | לא |
| **הגדרת משתנה public** | כן | כן | לא |
| **הגדרת משתנה readonly private** | כן – אפשר לאתחל רק בשורת ההגדרה או בבנאי | כן – אפשר לאתחל רק בשורת ההגדרה או בבנאי | לא |
| **הגדרת משתנה readonly public** | כן – אפשר לאתחל רק בשורת ההגדרה או בבנאי | כן – אפשר לאתחל רק בשורת ההגדרה או בבנאי | לא |
| **הגדרת משתנה ללא הרשאת גישה** | כן – יהיה אוטומטית public | כן – יהיה אוטומטית public | כן – רק ככה אפשר להגדיר משתנים |
| **הגדרת משתנה readonly private static** | כן – אפשר לאתחל רק בשורת ההגדרה | כן – אפשר לאתחל רק בשורת ההגדרה | לא |
| **הגדרת משתנה readonly public static** | כן – אפשר לאתחל רק בשורת ההגדרה | כן – אפשר לאתחל רק בשורת ההגדרה | לא |
| **הגדרת משתנהabstract private** | לא | לא (שגיאת קומפילציה) | לא |
| **הגדרת משתנהabstract public** | לא | כן | לא |
| **הגדרת בנאי** | כן – קריאה על ידי הבנאי (new())  אפשר לפנות מבנאי נגזרת לבנאי בסיס ע"י super() | כן –אפשר לפנות אליו רק מבנאי נגזרת ע"י super() | לא |
| **הגדרת פונקציה public כולל מימוש** | כן | כן | לא |
| **הגדרת פונקציה private כולל מימוש** | כן | כן | לא |
| **הגדרת פונקציהabstract private** | לא | לא | לא |
| **הגדרת פונקציהabstract public** | לא | כן | לא |
| **הגדרת פונקציה ללא מימוש** | לא | לא | כן |
| **ירושה של מחלקה רגילה (extends)** | כן   * נוכל לגשת למשתנים לא סטטיים public של הבסיס על ידי המילה this * נוכל לגשת למשתנים סטטיים public של הבסיס על ידי שם מחלקת הבסיס * נוכל לגשת לפונקציות סטטיות public של הבסיס על שם מחלקת הבסיס * נוכל לגשת לפונקציות לא סטטיות public של הבסיס על ידי המילה this או super – אם דרסנו את פונקציית הבסיס נוכל לפנות רק על ידי super | כן (לא שימושי אבל אפשרי) | לא |
| **ירושה של מחלקה אבסרקטית (extends)** | כן   * חובה לממש את כל המשתנים האבסטרקטיים או הפונקציות האבסרקטיות * נוכל לגשת למשתנים לא סטטיים public של הבסיס על ידי המילה this * נוכל לגשת למשתנים סטטיים public של הבסיס על ידי שם מחלקת הבסיס * נוכל לגשת לפונקציות סטטיות public של הבסיס על שם מחלקת הבסיס * נוכל לגשת לפונקציות לא סטטיות public של הבסיס על ידי המילה this או super – אם דרסנו את פונקציית הבסיס נוכל לפנות רק על ידי super | כן   * לא חובה לממש את כל המשתנים האבסטרקטיים או הפונקציות האבסרקטיות – מה שלא מומש נהיה חייבים לממש בנגזרת * נוכל לגשת למשתנים לא סטטיים public של הבסיס על ידי המילה this * נוכל לגשת למשתנים סטטיים public של הבסיס על ידי שם מחלקת הבסיס * נוכל לגשת לפונקציות סטטיות public של הבסיס על שם מחלקת הבסיס * נוכל לגשת לפונקציות לא סטטיות public של הבסיס על ידי המילה this או super – אם דרסנו את פונקציית הבסיס נוכל לפנות רק על ידי super | לא |
| **מימוש של ממשק (implements)** | כן - חובה לממש את כל המשתנים או הפונקציות מהממשק  אפשר לממש כמה ממשקים במחלקה אחת | כן - חובה לממש את כל המשתנים או הפונקציות מהממשק (אפשר להגדיר אותם כאבסטרקטיים ואז לממש אותם בנגזרת)  אפשר לממש כמה ממשקים במחלקה אחת | לא |
| **ירושה של ממשק (extends)** | לא | לא | כן – אפשר לרשת כמה ממשקים - וכאשר נממש במחלקה את הממשק הזה, נהיה חייבים לממש גם את כל מה שבממשק הבסיס |
| **יצירת משתנה (מצביע)** | כן – יכול להצביע על אובייקטים מהסוג שלו או על אובייקטי נגזרת | כן – יכול להצביע רק על אובייקטי נגזרת | כן – יכול להצביע רק על אובייקטי מחלקה שמממשת את הממשק |
| **יצירת אובייקט** | כן | לא | לא |
| **ביצוע casting לאובייקט על ידי instanceOf** | כן | כן | לא |
| **גישה על ידי מצביע בסיס למשתני נגזרת (שלא קיימים בבסיס)** | לא – חייב המרה מקומית לטיפוס הנגזרת | לא – חייב המרה מקומית לטיפוס הנגזרת | לא – חייב המרה מקומית לטיפוס הנגזרת |