OOP

Object Oriented Programming

תכנות מונחה אובייקטים. תכנות מונחה עצמים. עצם = אובייקט.

רוב הזמן אנחנו נעבוד עם דאטה (נתונים) ופעולות על הדאטה. שיטת ה-OOP מקלה עלינו בזאת.

בבסיס השיטה, הרעיון לייצר "תבניות" של אובייקטים, ואז ליצור את האובייקטים מתוך התבניות.

היחידה הבסיסית, כלומר ה"תבנית" נקראת Class.

Class לא במשמעות של כיתה....

אלא קיצור של המילה Classification שמשמעה "סיווג".

לפעמים מתרגמים את המילה Class בעברית למילה "מחלקה".

שם של קלאס ייכתב תמיד בסגנון UpperCamelCase

ולא upperCamelCase

ולא uppercamelcase

כמעט תמיד ניצור קלאס בקובץ נפרד, ולא מספר קלאסים באותו קובץ.

**Data Members**

אלו הם מאפייני הקלאס,

**Methods**

אלו הן פונקציות, המתארות את הפעולות שאפשר לעשות על האובייקטים. שימו לב שלא צריך לכתוב את המילה השמורה function מכיוון שפונקציות אלה הן מתודות. מתודה היא פעולה שקלאס יכול לעשות. בפועל, מייצגים את זה באמצעות פונקציות.

**Constructor**

הקונסטרקטור היא פונקציה (לא מתודה) מיוחדת המאפשרת לאתחל את האובייקט בנתונים.

הקונסטרקטור היא פונקציה שמתבצעת אך ורק פעם אחת מיד באיתחול אובייקט מתוך הקלאס, אפשר להגדיר לה חתימת פרמטרים באמצעותם ניתן לאתחל את הקלאס.

לא חובה לאתחל את כל הדטה ממברז דרך הקונסטרקטור, אבל נהוג שכן כי זה בדכ מה שצריך.

יש רק קונסטרקטור אחד לקלאס

**Readonly**

ניתן לסמן דטה ממבר כ-readonly ואז לא ניתן לשנות אותו מבחוץ. ניתן לאתחל משתנה readonly או בזמן ההגדרה או מתוך הקונסטרקטור. אך לא דרך שום שיטה אחרת

Access Modifiers

כמפתח הקלאס, אני מקבל יכולת להגדיר עבור כל אחד ואחד מהממברים ומהמתודות, מה תהיה רמת הנגישות אליהם.

ישנן 3 רמות נגישות:

1. Public – גישה פתוחה לכל אחד. ניתן לדעת על קיום של דטה ממברס ופונקציות וניתן גם לגשת אליהם (ואפילו לשנות אם הם לא מוגדרים readonly)
2. Protected
3. Private – הגישה לנתונים ולמתודות תהיה פתוחה אך ורק לאובייקטים שנוצרו מהקלאס עצמו.

**Static**

ממבר סטטי הוא ממבר שערכו זהה בכל אחד מהאובייקטים ולכן כדי לא לשכפל אותו עבור כל אובייקט, שומרים אותו ברמת הקלאס עצמו.

הגישה אליו היא דרך שם הקלאס. לדוגמה : Product.vat

כנ"ל גם מתודות יכולות להיות סטטיות, קרי ברמת הקלאס, ולא ברמת האובייקט. כלל אצבע – אם בקוד של המתודה, אין שימוש ב-this הרי שכנראה שאין למתודה צורך בנתון שספציפי לאובייקט, ולכן כדאי להפוך אותה לסטטית.

מתודות סטטיות, נקראות דרך הקלאס ולא דרך האובייקט, למשל:

Product.whereWereYouMade()

**Inheritence**

בהמון מקרים נגלה שלקלאסים שלנו יש אחד או יותר מאפיינים משותפים.

למשל: קלאסים של כלב, חתול ודג.

לכולם יש לב

לכולם יש עיניים

לא לכולם יש קשקשים

ניתן ליצור הגדרות של אובייקטים בעלי תכונות משותפות ולאגד את הנתונים והתכונות בקלאסים בסיסיים, שאותם ניתן אחר כך לירוש ולהרחיב (הרחבה משמעותה יציקת תוכן חדש לקלאס, כמו למשל דטה ממברס חדשים או מתודות חדשות)

כדי לרשת מקלאס נתון נשתמש במילה extends למשל:

class Car extends Vehicle {}

כשמייצרים קונסטרקטור בקלאס שיורש, צריך לקלוט את כל הנתונים האפשריים עבור האובייקט, גם עבור הקלאס של האובייקט עצמו וגם עבור הקלאס ממנו ירש, ובתוך הקונסטרקטור לקרוא למתודה super המפעילה את הקונסטרקטור בקלאס הבסיס.

**Override**

קלאס שירש מקלאס אחר, יכול להרחיב את המתודות שירש, על ידי קריאה למתודה דרך super. זה בעצם יפעיל את המתודה בקלאס הבסיס, ואז הקוד יכול להמשיך ולבצע את מה שנדרש ממנו בקלאס הירוש.

**Abstract**

אם אנו יוצרים קלאס לצרכי ירושה בלבד, ואיננו רוצים שמתענת אי פעם ייצור ממנו אובייקט (לדוגמא, vehicle) נוכל להוסיף את המילה abstract לפני המילה class וכך נודיע לטייפסקריפט שאיננו מרשים ליצור אובייקטים מהקלאס הזה. הקלאס הזה שימושו יהיה רק לירושה.

ניתן לייצר גם מתודה אבסטרקטית. המשמעות היא שמייצרים חתימה שלה בקלאס הבסיס, ומכיוון שהיא מתוייגת abstract אנו בעצם אומרים לטייפסקריפט להכריח את כל מי שיורש מהקלאס הזה, ליישם את המתודה אצלו

**Constructor Assignment**

במקום לכתוב את כל רשימת הדטה ממברז ואז להעביר אותה לקונטרקטור ואז לאתחל את כל הדטה ממברז בפרמטרים, אפשר פשוט להעביר את ה-access modifers (public, protected, private) אל בין הסוגריים של הקונסטרקטור ואז טייפסקריפט מבין שזה קונסטרקטור לאתחול האובייקט

לדוגמה

public constructor(

        public manufacturer: string,

        public model: string,

        public color: string,

        public yearMade: number,

        public price: number

    ) { }

**Enum**

קיצור של המילה enumerator או בעברית : ספיר. מלשון לספור. כלומר ניתן לספור אותו. משתמשים ב-enum כאשר יש לי סט סגור וסופי של פריטי מידע. למשל: מצב משפחתי (נשוי, רווק, אלמן, גרוש), למשל מין (זכר, נקבה, אחר) למשל גזעי כלבים.

**Interface**

אינטרפייס הוא לאו דווקא נושא oop-י במהותו, מכיוון שלאינטרפייס יש גם שימוש מחוץ לעולם ה-oop. אנחנו נסתכל על שני השימושים.

בגדול, שני השימושים דומים במהותם, ומהות האינטרפייס היא אכיפת סט של נתונים ופעולות על אובייקטים.

ניתן להשתמש באינטרפייס כדי לאכוף התנהגות (נתונים ומתודות) על כל קלאס שנייצר, באמצעות המילה implements. למשל:

Class Kitten implements Animal

קלאס יכול לרשת מקלאס אחד בלבד, אך ליישם אינספור אינטרפייסים. לדוגמה:

{}Class IPhone extends Device implements Electric, MusicPlayer

שימוש יעיל ביותר לאינטרפייסים נמצא כשנקבל מידע מגורמים חיצוניים (למשל API כלשהו) ואז נרצה לאכוף את מבנה המידע שקיבלנו.