ΠΕΡΙΕΧΟΜΕΝΑ:

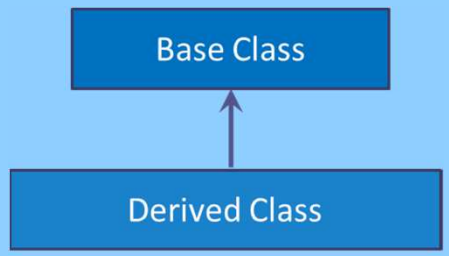
1. Κληρονομικότητα
    1. Αναπαράσταση Κληρονομικότητας
    2. Υπέρβαση Μεθόδων
2. Στοιχεία OOP που δεν υποστηρίζει η JS
3. Ασκήσεις

Εμμανουήλ Α.
Σμαραγδένιος Χορηγός Μαθήματος

Ανδρέας Γ.
Σμαραγδένιος Χορηγός Μαθήματος

Η **κληρονομικότητα (inheritance):**

- Παίρνει μία κλάση, και την επαυξάνει σε μία καινούργια, προσθέτοντας σε αυτήν χαρακτηριστικά (μέλη – μεθόδους)
- Σχηματικά απεικονίζεται ως εξής:



  - Η **παραγόμενη κλάση (derived class)** περιέχει όλα τα μέλη και τις μεθόδους της **βασικής κλάσης (base class)**
  - και λέμε ότι η παραγόμενη κλάση (derived class) **κληρονομεί (inherits)** τη βασική κλάση (base class)
- Εναλλακτικά θα μπορούσε να αναπαρασταθεί και:

  - Για να φαίνεται ότι η παραγόμενη κλάση **επαυξάνει (επεκτείνει, extends)** τη βασική κλάση.

Ορίζουμε ότι η κλάση μας θα κληρονομήσει μία άλλη κλάση:

- με τη λέξη κλειδί extends ακολουθούμενη από το όνομα της βασικής κλάσης (βλ. inheritance_constructors.html):

```
class Derived extends Base {
  …
}
```

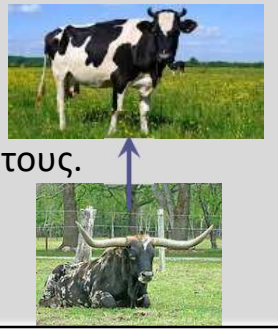
- ενώ ο κατασκευαστής της παραγόμενης κλάσης, μπορεί να καλέσει τον κατασκευαστή της βασικής κλάσης, με την :

```
class Base {
   constructor(baseProperty) {
      this.baseProperty = baseProperty;
   }
}
class Derived extends Base {
   constructor(baseProperty, derivedProperty) {
     super(baseProperty);
     this.derivedProperty = derivedProperty;
   }
}
```

Παράδειγμα 1: inheritance_example

Στο παράδειγμα αυτό:

- Η κλάση «αγελάδα» ορίζει αγελάδες με βάρος και διάθεση, και μία μέθοδο έκφρασής τους.
- Η κλάση «μακρυκέρατη αγελάδα του Τέξας» κληρονομεί την αγελάδα και ορίζει επιπλέον το μήκος των κεράτων της.

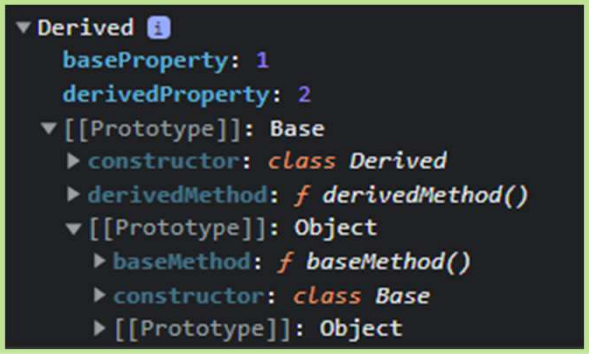


Βλέπουμε την αναπαράσταση της κληρονομικότητας:

- Και η κληρονομικότητα είναι συντακτική ζάχαρη.
- Στην πραγματικότητα κατασκευάζεται μηχανισμός πρωτοτύπων σαν αυτόν που είδαμε στο μάθημα 8.3.
- Βλέπουμε το παράδειγμα inheritance_template:

```
class Base {
  constructor(baseProperty) {
    this.baseProperty = baseProperty;
  }
  baseMethod() {}
}
class Derived extends Base {
  constructor(baseProperty, derivedProperty) {
    super(baseProperty);
    this.derivedProperty = derivedProperty;
  }
  derivedMethod() {};
}
```

- παράγει το αντικείμενο:



- [Παρατηρήστε ότι το πρωτότυπο του αντ/νου Derived είναι αντ/νο Base]

Σημαντική Σημείωση:

- Ο τρόπος που αναπαρίσταται είναι:
  - Τα μέλη δεδομένων της βασικής κλάσης μεταφέρονται στο αντικείμενο παραγόμενης κλάσης
  - Τα μέλη - μέθοδοι της παραγόμενης κλάσης είναι στο πρωτότυπό του αντικειμένου της παραγόμενης κλάσης
  - Τα μέλη - μέθοδοι της βασικής κλάσης είναι στο πρωτότυπό του αντικειμένου της βασικής κλάσης
- Και ισχύει ότι το πρωτότυπο - βασικής κλάση είναι κοινό για όλα τα αντικείμενα της παραγόμενης κλάσης.

Άσκηση 1:

- Επεκτείνοντας το παράδειγμα 1, κατασκευάστε δύο μακρυκέρατες αγελάδες του Τέξας, και διαπιστώστε ότι οι τιμές που έχουν στα properties της βασικής κλάσης είναι ανεξάρτητες.
- Έπειτα διαπιστώστε ότι το πρωτότυπο των δύο μακρυκέρατων αγελάδων είναι όντως κοινό.

Υπέρβαση Μεθόδων (Method overriding):

- Η υποκλάση μπορεί να επαναορίσει κάποια μέθοδο της υπερκλάσης (“Υπερβαίνει” τον προηγούμενο ορισμό της μεθόδου)
- Τα αντικείμενα της υποκλάσης θα καλούν την εκδοχή της μεθόδου που έχουν οριστεί στην υποκλάση.

Η αναφορά super:

- Αν κάνουμε override κάποια μέθοδο και θέλουμε για κάποιο λόγο, **στην παραγόμενη κλάση να κάνουμε κλήση σε μέθοδο της βασικής κλάσης που κάναμε override:**
  - Χρησιμοποιούμε τη λέξη-κλειδί super, που είναι μια αναφορά στο πρωτότυπο της βασικής κλάσης

Παράδειγμα 2: method_override

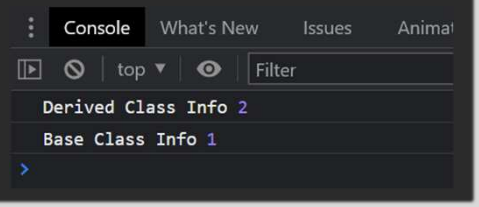
```
class Cow {
  ...
  express() {
    if (this.hunger > 5)
      console.log("Mooooooowwwwwww");
    else
      console.log("Mowww");
  }
}
class TexasLonghorn extends Cow {
  ...
  express() {
    if (this.hunger > 5)
      console.log("Meeeeeeewwwwwww");
    else
      console.log("Mewww");
  }
}
let molly = new Cow(500, 10);
molly.express();
let bob = new TexasLonghorn(400,20,0.50)
bob.express()
```

Παράδειγμα 3: method_override_super

```
class Base {
  constructor(baseProperty) {
    this.baseProperty = baseProperty;
  }
  info() {
    console.log("Base Class Info", this.baseProperty)
  }
}

class Derived extends Base {
  constructor(baseProperty, derivedProperty) {
    super(baseProperty);
    this.derivedProperty = derivedProperty;
  }
  info() {
    console.log("Derived Class Info", this.derivedProperty);
    super.info();
  }
}

let derived = new Derived(1,2);
derived.info();
```



ΜΑΘΗΜΑ 8.5: OOP: Κληρονομικότητα 2. Στοιχεία OOP που δεν υποστηρίζει η JS

JavaScript psounis

Στοιχεία OOP που ΔΕΝ υποστηρίζει η JS:

- **Αφηρημένες Κλάσεις** (=κλάσεις των οποίων δεν ορίζονται αντικείμενα και ο μόνος ρόλος τους είναι να κληρονομηθούν)
- **Τελικές Κλάσεις** (=Κλάσεις που δεν μπορούν να κληρονομηθούν)
- **Πολλαπλή Κληρονομικότητα** (=Η κλάση μας να μπορεί να κληρονόμήσει περισσότερες από μία κλάσεις)
- **Προστατευμένα Μέλη** (=Μέλη που είναι ιδιωτικά στην κλάση, αλλά ορατά στις υποκλάσεις)
- **Διεπαφές (interfaces)** (=Σύνολα Μεθόδων που πρέπει να υποστηρίζει μία κλάση)
- **Στοιχεία Πολυμορφισμού** (Αναφορές Βασικής Κλάσης κ.λπ.)
- **Υπερφόρτωση Τελεστών** (=Τελεστές να δουλεύουν επί αντικειμένων κλάσης)

Παρατήρηση:

- Όλα τα παραπάνω δεν υποστηρίζονται συντακτικά, χωρίς όμως αυτό να σημαίνει ότι οι προγραμματιστές JS δεν αναζητούν workarounds ώστε να προσομοιωθεί λειτουργικότητα σαν την παραπάνω.
- Στο stackoverflow μπορούμε να βρούμε σχετικά άρθρα (που όμως ξεφεύγουν από τους σκοπούς αυτής της σειράς)
- Θα δούμε π.χ. πως μπορούμε με «πλάγιο» τρόπο να ενσωματώσουμε mixins στο πρόγραμμά μας.

- **Τα mixins** προέρχονται από γλώσσες που επιτρέπουν πολλαπλή κληρονομικότητα(π.χ. Ruby). Μέσω αυτών γίνεται προσπάθεια να προσομοιωθεί η πολλαπλή κληρονομικότητα, αποφεύγοντας όμως τα προβλήματα της.
  - **είναι κλάσεις που περιέχουν μεθόδους για να χρησιμοποιηθούν από άλλες κλάσεις**
  - **Χωρίς όμως να πρέπει να κληρονομηθούν από άλλες κλάσεις.** Λέμε ότι ένα mixin ενσωματώνεται στην κλάση (αντί να κληρονομείται)
- Συχνά, είναι λειτουργικότητα που είναι κοινή σε πολλές κλάσεις (όπως π.χ. η αποθήκευση σε JSON αρχεία)

Παράδειγμα 4: mixin

```
class Superhero { … }

function mixinFlying(superhero) {
   Object.assign(superhero,{
      fly() { console.log("I am flying"); },
      airAttack() { console.log("I am attacking"); }
   });
}
let ironMan = new Superhero("Iron Man", 50, 1000);
let batman = new Superhero("Batman", 150, 1200);
mixinFlying(ironMan);
console.log(ironMan);
```

Άσκηση 2.1:

Η κλάση Customer έχει:

- Ονοματεπώνυμο
- Διεύθυνση
- Παραγγελίες (πίνακας από αντικείμενα τύπου Order)
- Μέθοδος: placeOrder() τοποθετεί ένα αντικείμενο τύπου Order στο τέλος του πίνακα.
- toString(): Τυπώνει τα στοιχεία του, κάθε παραγγελία και το συνολικό ποσό των παραγγελιών του.

Η κλάση Order έχει μέλη:

- Ημερομηνία (συμβολοσειρά σε format “YYYYMMDD” π.χ. “20201105”)
- Πληρωμή: Αντικείμενο τύπου Payment

Η κλάση Payment έχει μέλη:

- Ποσό

Στο κυρίως πρόγραμμα:

- Ορίστε έναν πελάτη
- Κατασκευάστε τρεις παραγγελίες και θέστε τις στον πελάτη.
- Τυπώστε τον πελάτη (μέσω της toString())

[Κατασκευάστε όποιαδήποτε επιπλέον μέθοδο κρίνετε σκόπιμη]

Άσκηση 2.2:

Η κλάση Payment κληρονομείται από την Credit η οποία έχει μέλη:

- number (αριθμός κάρτας)
- expDate (ημ/νία λήξης κάρτας - συμβολοσειρά)

Η κλάση Payment κληρονομείται από την Check η οποία έχει μέλη:

- number (αριθμός check_book)
- bankCode (συμβολοσειρά)

Στο κυρίως πρόγραμμα:

- Ορίστε έναν πελάτη
- Κατασκευάστε τρεις παραγγελίες (μία κανονική, μία με Credit και μία με Check) στον πελάτη
- Εκτυπώστε τον πελάτη (μέσω της toString())

[Ορίστε οτιδήποτε επιπλέον κρίνετε σκόπιμο]