Mapty Project - OOP & Geolocation

> 🎓 **CS Perspective:** This project demonstrates **object-oriented architecture** with ES6 classes implementing **inheritance** (Running/Cycling extend Workout). The use of **private class fields** (`#map`, `#workouts`) shows true encapsulation—these are inaccessible outside the class, unlike the `_` convention which is just naming. The app implements the **Publisher-Subscriber pattern** through event handlers and uses **delegation** for workout clicks. **Geolocation API** shows asynchronous browser APIs with callbacks. **LocalStorage** demonstrates client-side **persistence** with JSON serialization/deserialization. The `bind(this)` calls in the constructor address the classic **context binding problem** when methods are used as callbacks. The architecture follows **separation of concerns**: data classes (Workout, Running, Cycling) vs. application logic (App).

📌 Class Architecture

```
              ┌─────────────┐
              │   Workout   │   (Parent class)
              ├─────────────┤
              │ coords      │
              │ distance    │
              │ duration    │
              │ date        │
              │ id          │
              └──────┬──────┘
                     │
            ┌────────┴────────┐
            │                 │
            ▼                 ▼
      ┌────────────┐    ┌────────────┐
      │  Running   │    │  Cycling   │
      ├────────────┤    ├────────────┤
      │ cadence    │    │ elevation  │
      │ pace       │    │ speed      │
      └────────────┘    └────────────┘
```

📌 ES6 Class Fields

Public and private fields declared outside constructor:

```javascript
class Workout {
  // Public fields (on each instance)
  date = new Date();
  id = (Date.now() + '').slice(-10);
  clicks = 0;

  constructor(coords, distance, duration) {
    this.coords = coords;
    this.distance = distance;
    this.duration = duration;
  }
}

class App {
  // Private fields (truly encapsulated)
  #map;
  #mapZoomLevel = 13;
  #mapEvent;
  #workouts = [];

  constructor() {
    // ...
  }
}
```

Syntax	Type	Accessible Outside Class
`field = value`	Public field	✅ Yes
`#field = value`	Private field	❌ No
`_field`	Convention only	✅ Yes (just naming)

📌 Inheritance with `extends` and `super`

```
  class Running extends Workout {
    type = 'running';

    constructor(coords, distance, duration, cadence) {
      super(coords, distance, duration);  // Call parent constructor
      this.cadence = cadence;
      this.calcPace();
      this._setDescription();
    }

    calcPace() {
      this.pace = this.duration / this.distance;
      return this.pace;
    }
  }
```

`super` Keyword

- `super()` calls parent constructor (must be first in child constructor)
- `super.method()` calls parent method

📌 Geolocation API

```
_getPosition() {
  if (navigator.geolocation)
    navigator.geolocation.getCurrentPosition(
      this._loadMap.bind(this),  // Success callback
      function () {              // Error callback
        alert('Could not get your position');
      }
    );
}
```

Position Object

```
_loadMap(position) {
  const { latitude } = position.coords;
  const { longitude } = position.coords;
  // position.coords also has: accuracy, altitude, heading, speed
}
```

📌 The `bind(this)` Problem

When methods are used as callbacks, `this` is lost:

```
class App {
  constructor() {
    // ❌ Wrong: `this` will be undefined in callback
    navigator.geolocation.getCurrentPosition(this._loadMap);

    // ✅ Correct: bind `this` to the method
    navigator.geolocation.getCurrentPosition(this._loadMap.bind(this));
  }
}
```

Common Places Needing `bind(this)`

```
// Event listeners
form.addEventListener('submit', this._newWorkout.bind(this));

// Geolocation
navigator.geolocation.getCurrentPosition(this._loadMap.bind(this));

// Map events (Leaflet)
this.#map.on('click', this._showForm.bind(this));
```

📌 Working with Leaflet.js

Initialize Map

```
this.#map = L.map('map').setView(coords, this.#mapZoomLevel);

L.tileLayer('https://{s}.tile.openstreetmap.fr/hot/{z}/{x}/{y}.png', {
  attribution: '&copy; OpenStreetMap contributors',
}).addTo(this.#map);
```

Add Marker with Popup

```
L.marker(workout.coords)
  .addTo(this.#map)
  .bindPopup(
    L.popup({
      maxWidth: 250,
      minWidth: 100,
      autoClose: false,
      closeOnClick: false,
      className: `${workout.type}-popup`,
    })
  )
  .setPopupContent('🏃 Running on April 15')
  .openPopup();
```

Handle Map Clicks

```
this.#map.on('click', this._showForm.bind(this));

_showForm(mapE) {
  this.#mapEvent = mapE;
  const { lat, lng } = this.#mapEvent.latlng;
}
```

📌 Data Validation with Helper Functions

```
_newWorkout(e) {
  // Validation helpers using rest parameters and .every()
  const validInputs = (...inputs) =>
    inputs.every(inp => Number.isFinite(inp));
  const allPositive = (...inputs) =>
    inputs.every(inp => inp > 0);

  // Usage
  if (!validInputs(distance, duration, cadence) ||
      !allPositive(distance, duration, cadence))
    return alert('Inputs have to be positive numbers!');
}
```

📌 Event Delegation

Handle clicks on dynamically added elements:

```
// Listen on parent container
containerWorkouts.addEventListener('click', this._moveToPopup.bind(this));

_moveToPopup(e) {
  // Find clicked workout element
  const workoutEl = e.target.closest('.workout');
  if (!workoutEl) return;

  // Find workout data using data attribute
  const workout = this.#workouts.find(
    work => work.id === workoutEl.dataset.id
  );

  // Navigate to location
  this.#map.setView(workout.coords, this.#mapZoomLevel, {
    animate: true,
    pan: { duration: 1 }
  });
}
```

📌 LocalStorage API

Persist data across browser sessions:

```
// Save (serialize to JSON)
_setLocalStorage() {
  localStorage.setItem('workouts', JSON.stringify(this.#workouts));
}

// Load (parse from JSON)
_getLocalStorage() {
  const data = JSON.parse(localStorage.getItem('workouts'));
  if (!data) return;
```

```
    this.#workouts = data;
    this.#workouts.forEach(work => this._renderWorkout(work));
  }

  // Clear
  reset() {
    localStorage.removeItem('workouts');
    location.reload();
  }
```

⚠ LocalStorage Limitation

Objects lose their prototype chain when serialized:

```
// After JSON.parse, objects are plain objects, NOT instances
// workout.click() won't work - no longer a Workout instance
```

📌 Unique ID Generation

```
id = (Date.now() + '').slice(-10);

// Date.now() → 1703145600000 (timestamp)
// + ''        → "1703145600000" (string)
// .slice(-10) → "3145600000" (last 10 chars)
```

📌 App Initialization Flow

```
class App {
  constructor() {
    // 1. Get user's position → loads map
    this._getPosition();

    // 2. Load saved workouts from localStorage
    this._getLocalStorage();

    // 3. Attach event handlers
    form.addEventListener('submit', this._newWorkout.bind(this));
    inputType.addEventListener('change', this._toggleElevationField);
    containerWorkouts.addEventListener('click',
this._moveToPopup.bind(this));
  }
}
```

```
// Instantiate app
const app = new App();
```