1 — Informe de reflexión (README.md)

```
# Proyecto: Prototype Chain en JavaScript + Fútbol

## Entrega
Contenido preparado para entrega: README, código original (phase2_code.js), código optimizado (phase3_code.js) y diagrama (diagram.svg).

## 1) Descripción del problema
Elegí el concepto **Prototype Chain** en JavaScript y lo combiné con el tema **Fútbol**. Objetivo: modelar Person, Player y Team usando prototipos (sin usar `class`) y construir una simulación que maneje eventos asincrónicos. Se siguió además la restricción de evitar `map()` y usar callbacks en la versión inicial.

## 2) Fase 1 — Identificación y pensamiento divergente
**Prompt (transcripción):**
"Quiero practicar Prototype Chain en JS usando un tema creativo: fútbol. Dame ideas de cómo modelar Person, Player y Team con prototipos y proponme un snippet corto que use Object.create en lugar de class."

**Resultado clave:** diseño con `Person`, `Player` (hereda de Person vía `Object.create`), `Team`. Idea añadida: simulación de partido y eventos de gol con `setTimeout`.

## 3) Fase 2 — Desarrollo y desafío del sesgo (código 'prohibido')
Se implementó la idea original de forma intencionada *no estándar*:
- Sin `class`.
- Sin `map()`.
- Uso de callbacks para operaciones asincrónicas.

Archivo: `phase2_code.js`.

## 4) Fase 3 — Revisión, evaluación y optimización
Feedback de la IA:
- **Legibilidad:** modularizar funciones, mejores nombres.
- **Bugs:** falta manejo de errores en callbacks.
- **Buenas prácticas:** validaciones, evitar lookups repetidos en la cadena de prototipos.
Sugerencias de optimización:
1. Migrar callbacks a Promises/async-await (mejor manejo de errores y legibilidad).
2. Usar métodos modernos de array (`map`, `reduce`) donde sea adecuado y caché de valores si hay costosas operaciones de prototipo.

Impleméntese en `phase3_code.js`: Promises/async-await y uso de `reduce` para top scorer + validaciones.

## 5) Fase 4 — Presentación y diagrama
Incluí un diagrama SVG (`diagram.svg`) que muestra la cadena de prototipos y el cambio de flujo: callbacks -> Promises.

## Archivos incluidos
- README.md
- phase2_code.js
- phase3_code.js
- diagram.svg
```

```
## Cómo ejecutar
- `node phase2_code.js`  // versión con callbacks
- `node phase3_code.js`  // versión con Promises / async-await
```

---

2 — Evidencia de prompts (transcripciones)

Prompt Fase 1 (pensamiento divergente):

```
Quiero practicar Prototype Chain en JavaScript usando Fútbol como tema. Dame 3 ideas creativas para modelar Person, Player y Team con prototipos (sin class), y sugiere un pequeño snippet que use Object.create y demuestre herencia.
```

Prompt Fase 2 (desafío al sesgo / restricciones):

```
Elige la idea más original y genera el código pero rompe con las prácticas estándar:
- No usar class
- No usar map(), filter() ni sort()
- Usar callbacks en lugar de async/await o Promises para manejar la simulación asincrónica.
Genera un snippet completo, con demo de ejecución.
```

Prompt Fase 3 (revisión y optimización):

```
Revisa el código anterior. Indica problemas de legibilidad, bugs potenciales y sugiere al menos 2 optimizaciones que mejoren eficiencia o legibilidad. Implementa los cambios y entrega el código optimizado.
```

---

3 — Código Final (fase 2) — `phase2_code.js`

(Archivo "prohibido": copia tal cual y guárdalo como `phase2_code.js`)

```
// phase2_code.js
// Enfoque "prohibido": sin class, usando Object.create, sin map(), usando callbacks para async.

function Person(name, age) {
  this.name = name;
  this.age = age;
}
Person.prototype.getBasicInfo = function() {
  return `${this.name}, ${this.age} años`;
};

function Player(name, age, position, goals) {
  // heredar de Person sin usar class
  const obj = Object.create(Player.prototype);
  Person.call(obj, name, age);
```

```
  obj.position = position;
  obj.goals = goals || 0;
  return obj;
}
Player.prototype = Object.create(Person.prototype);
Player.prototype.constructor = Player;
Player.prototype.score = function() {
  this.goals += 1;
  return this.goals;
};
Player.prototype.getProfile = function() {
  return `${this.getBasicInfo()} - ${this.position} (goles: ${this.goals})`;
};

function Team(name) {
  this.name = name;
  this.players = [];
}
Team.prototype.addPlayer = function(player) {
  this.players.push(player);
};
Team.prototype.getTopScorer = function() {
  // Sin usar map, filter o sort; hacer bucle manual
  var top = null;
  for (var i = 0; i < this.players.length; i++) {
    var p = this.players[i];
    if (!top || p.goals > top.goals) top = p;
  }
  return top;
};

// Simulación asincrónica "prohibida": callbacks
function simulateMatch(homeTeam, awayTeam, minutes, callback) {
  var elapsed = 0;
  function tick() {
    elapsed += 10; // cada tick = 10 minutos
    // simple random event: 20% prob de gol
    if (Math.random() < 0.2) {
      // elegir equipo y jugador con bucles
      var team = Math.random() < 0.5 ? homeTeam : awayTeam;
      var idx = Math.floor(Math.random() * team.players.length);
      var player = team.players[idx];
      player.score();
      // llamar callback de evento
      callback(null, { event: 'goal', team: team.name, player: player.name, minute: elapsed
});
    }
    if (elapsed < minutes) {
      setTimeout(tick, 50); // acelerar simulación
    } else {
      // partido terminado
      callback(null, { event: 'end', home: homeTeam.name, away: awayTeam.name });
    }
  }
  setTimeout(tick, 50);
```

```
}

// Demo
var a = Player('Lionel Messi', 37, 'Delantero', 0);
var b = Player('Kylian Mbappé', 26, 'Delantero', 0);
var c = Player('Kevin De Bruyne', 33, 'Centrocampista', 0);
var d = Player('Virgil van Dijk', 33, 'Defensa', 0);

var teamA = new Team('Azul FC');
teamA.addPlayer(a); teamA.addPlayer(c);
var teamB = new Team('Rojo FC');
teamB.addPlayer(b); teamB.addPlayer(d);

console.log('Comienza simulación (callbacks)...');
simulateMatch(teamA, teamB, 90, function(err, evt) {
  if (err) {
    console.error('Error en simulación:', err);
    return;
  }
  if (evt.event === 'goal') {
    console.log(`[Gol] ${evt.minute}' - ${evt.player} para ${evt.team}`);
  } else if (evt.event === 'end') {
    console.log('Partido finalizado entre', evt.home, 'y', evt.away);
    console.log('Máximo goleador equipo Azul:', teamA.getTopScorer() &&
teamA.getTopScorer().getProfile());
    console.log('Máximo goleador equipo Rojo:', teamB.getTopScorer() &&
teamB.getTopScorer().getProfile());
  }
});
```

---

4 — Evaluación y optimizaciones (Fase 3) — feedback y cambios aplicados

Evaluación sobre `phase2_code.js`:

- Legibilidad: nombres adecuados en general, pero mezcla de estilos (var vs const/let). Falta modularización de la lógica de eventos.
- Bug potencial: `simulateMatch` asume que `team.players.length > 0`. Si equipo vacío -> `Math.random() * 0` -> `NaN` -> `player === undefined` -> error al llamar `player.score()`.
- Manejo de errores: callback no distingue entre eventos intermedios y final; múltiples callbacks invocados (por cada gol y al final). Está bien, pero si hay un error inesperado no hay manejo claro.
- Performance: `getTopScorer` recorre array con bucle manual — está bien pero `reduce` es más expresivo y permite evitar variables temporales. El costoso lookup en prototipo no es crítico aquí.

Dos optimizaciones sugeridas:

1. **Migrar callbacks a Promises/async-await:** simplifica flujo, manejo de errores con try/catch y reduce la complejidad del callback hell.

2. **Validaciones y uso de métodos modernos de array:** usar `reduce` para `getTopScorer`, validar equipos antes de la simulación, y usar `const/let` para claridad.

Implementación: se creó `phase3_code.js` con:

- `simulateMatchAsync` que devuelve Promise y resuelve con resumen final.
- Uso de `reduce` para `getTopScorer`.
- Validaciones (`if (!homeTeam || !awayTeam) throw`).
- Encapsulado del demo dentro de `async` IIFE con try/catch.

---

5 — Código optimizado (fase 3) — `phase3_code.js`

```
// phase3_code.js
// Código mejorado: Promises / async-await y uso de reduce para top scorer.
// Mantuvimos prototipos (sin class) pero con mejor manejo de errores y modularidad.

function Person(name, age) {
  this.name = name;
  this.age = age;
}
Person.prototype.getBasicInfo = function() {
  return `${this.name}, ${this.age} años`;
};

function Player(name, age, position, goals) {
  const obj = Object.create(Player.prototype);
  Person.call(obj, name, age);
  obj.position = position;
  obj.goals = goals || 0;
  return obj;
}
Player.prototype = Object.create(Person.prototype);
Player.prototype.constructor = Player;
Player.prototype.score = function() {
  this.goals += 1;
  return this.goals;
};
Player.prototype.getProfile = function() {
  return `${this.getBasicInfo()} - ${this.position} (goles: ${this.goals})`;
};

function Team(name) {
  this.name = name;
```

```
  this.players = [];
}
Team.prototype.addPlayer = function(player) {
  if (!player || !player.name) throw new Error('Jugador inválido');
  this.players.push(player);
};
Team.prototype.getTopScorer = function() {
  if (this.players.length === 0) return null;
  // usar reduce para claridad y eficiencia
  return this.players.reduce(function(top, p) {
    return (!top || p.goals > top.goals) ? p : top;
  }, null);
};

// Simulación con Promises
function simulateMatchAsync(homeTeam, awayTeam, minutes) {
  return new Promise((resolve, reject) => {
    if (!homeTeam || !awayTeam) return reject(new Error('Equipos inválidos'));
    if (!Array.isArray(homeTeam.players) || !Array.isArray(awayTeam.players)) return
reject(new Error('Estructura de equipos inválida'));
    var elapsed = 0;
    function tick() {
      elapsed += 10;
      if (Math.random() < 0.2) {
        const team = Math.random() < 0.5 ? homeTeam : awayTeam;
        if (team.players.length === 0) {
          // ignorar evento de gol si no hay jugadores
        } else {
          const idx = Math.floor(Math.random() * team.players.length);
          const player = team.players[idx];
          player.score();
          console.log(`[Gol] ${elapsed}' - ${player.name} para ${team.name}`);
        }
      }
      if (elapsed < minutes) {
        setTimeout(tick, 50);
      } else {
        resolve({
          home: homeTeam.name,
          away: awayTeam.name,
          topHome: homeTeam.getTopScorer() && homeTeam.getTopScorer().getProfile(),
          topAway: awayTeam.getTopScorer() && awayTeam.getTopScorer().getProfile()
        });
      }
    }
    setTimeout(tick, 50);
  });
}

// Demo async/await
(async function demo() {
  try {
    const a = Player('Lionel Messi', 37, 'Delantero', 0);
    const b = Player('Kylian Mbappé', 26, 'Delantero', 0);
    const c = Player('Kevin De Bruyne', 33, 'Centrocampista', 0);
```

```
    const d = Player('Virgil van Dijk', 33, 'Defensa', 0);

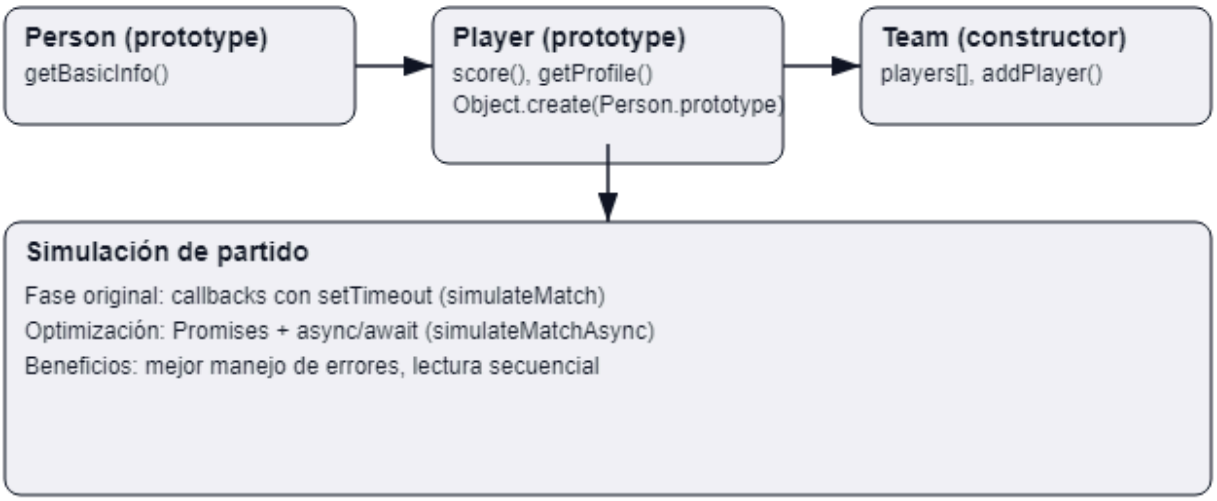
    const teamA = new Team('Azul FC');
    teamA.addPlayer(a); teamA.addPlayer(c);
    const teamB = new Team('Rojo FC');
    teamB.addPlayer(b); teamB.addPlayer(d);

    console.log('Comienza simulación (async/await)...');
    const result = await simulateMatchAsync(teamA, teamB, 90);
    console.log('Partido finalizado entre', result.home, 'y', result.away);
    console.log('Máximo goleador equipo Azul:', result.topHome);
    console.log('Máximo goleador equipo Rojo:', result.topAway);
  } catch (e) {
    console.error('Error en demo:', e);
  }
})();
```

---

6 — Diagrama / Recurso visual

Te doy **dos formatos**: un **SVG** listo para guardar (archivo `diagram.svg`) y un **Mermaid** (texto) que puedes pegar en un generador compatible (GitHub, VSCode Mermaid preview, etc.).



Mermaid.

```
```mermaid
graph LR
  Person["Person\ngetBasicInfo()"]
  Player["Player\nscore(), getProfile()\n(Object.create(Person.prototype))"]
  Team["Team\nplayers[], addPlayer()"]

  Person --> Player
```
```

```
  Player --> Team

  subgraph Simulación
    A[Callbacks (simulateMatch)] --> B[Eventos: goles]
    B --> C[Callback final]
  end

  subgraph Optimización
    D[Promises / async-await (simulateMatchAsync)]
    D --> E[Mejor manejo de errores]
  end

  B --> D
```

```
# 7 — Conclusión.
Incluye esta reflexión corta en tu README:
```

Conclusión

El ejercicio de diseñar la solución con `Object.create` y prescindir de `class` me obligó a repensar la herencia prototípica en JS: entender cómo y dónde se resuelven métodos en la cadena de prototipos. Implementar la versión "prohibida" (callbacks y bucles manuales) mostró riesgos reales: manejo de errores y casos borde. La refactorización a Promises/async-await mejoró la legibilidad y robustez; el uso de métodos como `reduce` aumentó claridad y expresividad. En resumen, romper con la práctica estándar fue didáctico, y la optimización confirmó por qué las prácticas modernas son preferibles en producción.