Exercice 1 - Chronomètre

Objectif

Maîtriser useEffect et sa fonction de cleanup avec un timer.

Concepts abordés

- useEffect avec dépendances
- Cleanup function
- useRef pour stocker l'interval ID
- Formatage de temps

Énoncé

Créer un chronomètre avec :

1. Affichage du temps (minutes:secondes:centièmes)
2. Bouton Démarrer/Pause (toggle)
3. Bouton Réinitialiser
4. Le chronomètre doit se nettoyer correctement au démontage

Comportement attendu

```
      00:05:23

  [Pause]  [Réinitialiser]
```

Points d'attention

Pourquoi useRef pour l'interval ?

```
// PROBLÈME avec useState pour l'interval ID
const [intervalId, setIntervalId] = useState(null);

function stop() {
  clearInterval(intervalId);  // intervalId pourrait être obsolète !
}

// SOLUTION : useRef
const intervalRef = useRef(null);

function stop() {
  clearInterval(intervalRef.current);  // Toujours la dernière valeur
}
```

Le cleanup est ESSENTIEL

Sans cleanup, si le composant est démonté pendant que le chrono tourne, l'interval continue et provoque des erreurs.

Correction

```jsx
// src/exercices/module-2/ex01-chronometre/Chronometre.jsx

import { useState, useRef, useEffect } from 'react';
import './Chronometre.css';

function Chronometre() {
  // ═══════════════════════════════════════════════════════════
  // STATE
  // - time : temps écoulé en centièmes de seconde
  // - isRunning : le chrono est-il en cours ?
  // ═══════════════════════════════════════════════════════════
  const [time, setTime] = useState(0);
  const [isRunning, setIsRunning] = useState(false);

  // ═══════════════════════════════════════════════════════════
  // REF
  // Pour stocker l'ID de l'interval sans causer de re-render
  // useRef garde la même référence entre les renders
  // ═══════════════════════════════════════════════════════════
  const intervalRef = useRef(null);

  // ═══════════════════════════════════════════════════════════
  // EFFECT
  //
  // Cet effet gère le démarrage/arrêt de l'interval.
  // Il se déclenche quand isRunning change.
  //
  // IMPORTANT : La cleanup function arrête l'interval quand :
  // 1. isRunning passe à false (avant de relancer l'effet)
  // 2. Le composant est démonté
  // ═══════════════════════════════════════════════════════════
  useEffect(() => {
    if (isRunning) {
      // Démarrer l'interval (toutes les 10ms = centième de seconde)
      intervalRef.current = setInterval(() => {
        setTime(prev => prev + 1);
      }, 10);
    }

    // CLEANUP : arrêter l'interval
    return () => {
      if (intervalRef.current) {
```

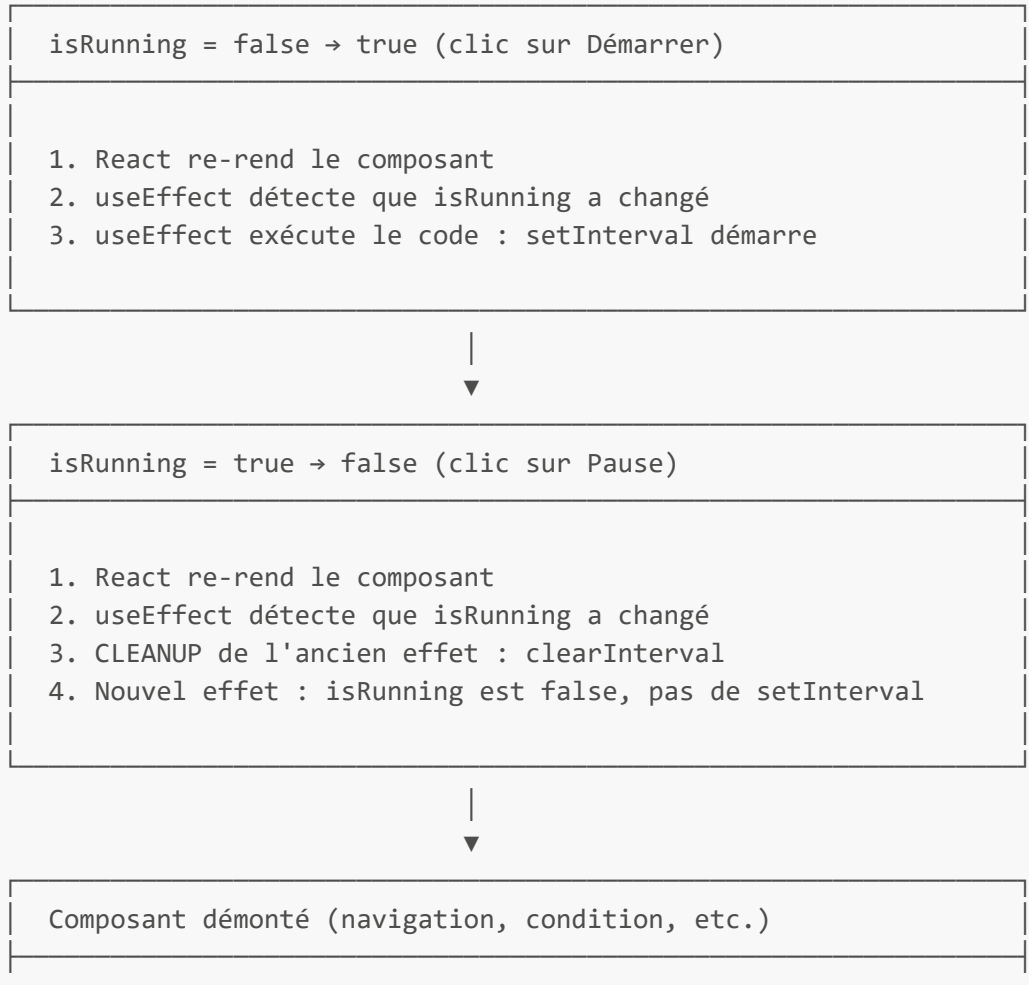
```
      clearInterval(intervalRef.current);
      intervalRef.current = null;
    }
  };
}, [isRunning]);  // Se déclenche quand isRunning change

// ============================================================
// HANDLERS
// ============================================================
function toggleRunning() {
  setIsRunning(prev => !prev);
}

function reset() {
  setIsRunning(false);
  setTime(0);
}

// ============================================================
// FORMATAGE DU TEMPS
//
// time est en centièmes de seconde
// 1 seconde = 100 centièmes
// 1 minute = 6000 centièmes
// ============================================================
function formatTime(centiseconds) {
  const minutes = Math.floor(centiseconds / 6000);
  const seconds = Math.floor((centiseconds % 6000) / 100);
  const centis = centiseconds % 100;

  // padStart pour avoir toujours 2 chiffres
  return [
    String(minutes).padStart(2, '0'),
    String(seconds).padStart(2, '0'),
    String(centis).padStart(2, '0')
  ].join(':');
}

// ============================================================
// RENDU
// ============================================================
return (
  <div className="chronometre">
    {/* Affichage du temps */}
    <div className="chrono-display">
      {formatTime(time)}
    </div>

    {/* Contrôles */}
    <div className="chrono-controls">
      <button
        onClick={toggleRunning}
        className={`chrono-btn ${isRunning ? 'pause' : 'start'}`}
      >
```

```
          {isRunning ? 'Pause' : 'Démarrer'}
        </button>

        <button
          onClick={reset}
          className="chrono-btn reset"
          disabled={time === 0 && !isRunning}
        >
          Réinitialiser
        </button>
      </div>
    </div>
  );
}

export default Chronometre;
```

```
/* src/exercices/module-2/ex01-chronometre/Chronometre.css */

.chronometre {
  text-align: center;
  padding: 2rem;
}

.chrono-display {
  font-family: 'Courier New', monospace;
  font-size: 4rem;
  font-weight: bold;
  color: #1f2937;
  margin-bottom: 1.5rem;
  letter-spacing: 0.1em;
}

.chrono-controls {
  display: flex;
  gap: 1rem;
  justify-content: center;
}

.chrono-btn {
  padding: 0.75rem 2rem;
  font-size: 1rem;
  font-weight: 500;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: all 0.2s;
}

.chrono-btn.start {
  background: #10b981;
```

```
  color: white;
}

.chrono-btn.pause {
  background: #f59e0b;
  color: white;
}

.chrono-btn.reset {
  background: #6b7280;
  color: white;
}

.chrono-btn:hover:not(:disabled) {
  transform: scale(1.05);
}

.chrono-btn:disabled {
  opacity: 0.5;
  cursor: not-allowed;
}
```

Schéma mental : Cycle de useEffect



```
│                                                                 │
│  CLEANUP de tous les effets : clearInterval                     │
│  → Évite les fuites mémoire et les erreurs                      │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

Pour aller plus loin

1. Ajouter des tours (laps) avec un bouton "Tour"
2. Sauvegarder le meilleur temps dans localStorage
3. Ajouter un compte à rebours (timer)