Դաս 09: Սխալների որսում, հետաձգված գործողություններ

Try/catch մեխանիզմը թույլ է տալիս որսալ ծրագրում տեղի ունեցող սխալները և ըստ սխալի տիպի իրականացնել համապատասխան գործողություններ առանց ծրագրի աշխատանքը ընդհատելու:
Գրելաձևը՝

```
try {
  // Կոդ, որը կարող է սխալ առաջացնել
} catch (error) {
  // Առաջացած սխալի մշակման բլոկ
} finally {
  // Ցանկացած դեպքում իրականացվող բլոկ
}
```

try-ի մեջ գրում են այն կոդը, որը պոտենցիալ կարող է սխալ առաջացնել,
catch-ը աշխատում է միայն այն դեպքում, երբ **try** բլոկում առաջանում է որևէ սխալ: Սխալի օբյեկտը փոխանցվում է որպես հատկություն **catch** բլոկին:
finally-ը (ըստ ցանկության) աշխատում է միշտ անկախ այն հանգամանքից, առաջացել է սխալ թե ոչ:

```
try {
  let result = 10 / 0; // Աշխատում է, սխալ չկա
  console.log(result);
} catch (error) {
  console.log("Սխալ: ", error.message);
} finally {
  console.log("Ծրագրի ավարտ");
}
```

Այժմ ստուգենք սխալի դեպքում ինչպես է աշխատում՝

```
try {
```

```
  nonExistentFunction();
} catch (error) {
  console.log(`Սխալ: ${error.message}`);
}
Սխալ: nonExistentFunction is not defined
```

Բացի ստանդարտ սխալները, մենք կարող ենք ստեղծել նաև արհեստական սխալներ, օգտագործելով **Error** օբյեկտը:

Այս օբյեկտը օգտագործում են արտակարգ իրավիճակների մասին օգտագործողին, ծրագրավորողին տեղյակ պահելու համար:

Error օբյեկտը ունի որոշ հիմնական հատկություններ՝

Name հատկությունը - ցույց է տալիս սխալի անվանումը՝ Error ընդհանուր սխալների համար, TypeError, ReferenceError և այլն կոնկրետ տեսակի սխալների համար:

```
const error = new Error('Սա ընդհանուր սխալ է');
console.log(error.name); // 'Error'
```

Message հատկությունը - ցույց է տալիս սխալի մասին հաղորդագրությունը՝

```
const error = new Error('Սա սխալի հաղորդագրություն է');
console.log(error.message);
// 'Սա սխալի հաղորդագրություն է'
```

Stack հատկությունը - ցույց է տալիս առաջացած սխալի տողը, շարքը և այլ հավելյալ ինֆորմացիաներ՝

```
const error = new Error('Սխալ');
console.log(error.stack);
Error: Սխալ
atObject.<anonymous>(C:\Users\user\Desktop\newreact\src
\index.js:21:15)
```

Օրինակ այն կիրառում են ֆունկցիաների մեջ, որտեղ անհրաժեշտ է բացառել սխալ տվյալների ստացման դեպքում ծրագրի աշխատանքը, օրինակ՝

```
function divide(a, b) {
  if (b === 0) {
    throw new Error("Զրոի չի կարելի բաժանել");
  }
  return a / b;
}
```

Նման մոտեցումը հաճախակի է կիրառվում մեթոդներում, եթե փորձում ենք օրինակ զանգվածներին պատկանող մեթոդները կանչել օգյեկտների, թվային տիպի տվյալների վրա:
Բացի այդ այն օգտագործում են կրիտիկական դեպքերը կանխելու համար, օրինակ երբ օգտատերը կարողացել է ստանալ հասանելիություն ֆունկցիային, որին թույլատրվում է դիմել միմիայն համակարգում մուտք գործելուց հետո՝

```
if (!user.isLoggedIn) {
    throw new Error('Անհայտ օգտատեր');
}
```

Նաև **throw new Error**-ի օգնությամբ կարելի է որոշակի պայմանի բավարարման դեպքում ստիպողաբար կանչել **catch** բլոկը՝

```
try {
    throw new Error('Ինչ որ սխալ կա');
} catch (error) {
    console.error(error.message);
// տպելու է: "Ինչ որ սխալ կա"
}
```

JavaScript-ում մենք կարող ենք օգտագործել նաև հետաձգված գործողություններ:
setTimeout մեթոդը թույլ է տալիս կատարել որևէ գործողություն սահմանված ժամանակահատվածից հետո:

```
setTimeout(callback, delay, param1, param2, ...);
```

Այստեղ՝

callback - ֆունկցիա կամ արտահայտություն, որը պետք է իրականացվի սահմանված ժամանակ անց,

Delay - հետաձգման ժամանակահատվածը միլիվարկյաններով,

Param1, param2 - **callback** ֆունկցիային փոխանցվող լրացուցիչ հատկություններ:

```
setTimeout(() => {
    console.log('Հետաձգված գործողություն՝ 3 վայրկյան անց');
}, 3000);
```

Ֆունկցիային հատկություն փոխանցելով՝

```
function greet(name) {
  console.log(`Բարև, ${name}!`);
}

setTimeout(greet, 2000, 'Վրամ');
```

setInterval մեթոդը թույլ է տալիս ֆունկցիան կանչել պարբերաբար տրված ինտերվալով:

```
setInterval(callback, interval, param1, param2, ...);
```

callback - ֆունկցիա կամ արտահայտություն, որը պետք է իրականացվի պարբերաբար,

Delay - ինտերվալի ժամանակահատվածը միլիվարկյաններով,

Param1, param2 - **callback** ֆունկցիային փոխանցվող լրացուցիչ հատկություններ:

```
let counter = 0;

const intervalId = setInterval(() => {
  counter++;
  console.log(`Հաշիվը՝ ${counter}`);
  if (counter === 5) {
```

```
        clearInterval(intervalId);  // Դադարեցնում է ինտերվալը
    console.log('Ինտերվալը դադարեցվեց');
  }
}, 1000);
```

Ինչպես երևում է օրինակում մենք օգտագործել ենք **clearInterval()** մեթոդը, որպեսզի որոշակի պայմանի բավարարման դեպքում իվիճակի լինենք դադարեցնել ինտերվալը: Նման մեթոդ կա նաև **timeout**-ի համար՝

clearTimeout(timeoutId) - դադարեցնում է **setTimeout**-ի հետաձգված գործողությունը:

clearInterval(intervalId) - դադարեցնում է **setInterval**-ի պարբերական գործողությունը:

Որպեսզի կարողանանք դադարեցնել այդ պրոցեսները, անհրաժեշտ է վերագրել **setTimeout**-ը և **setInterval**-ը որևէ փոփոխականի, որպեսզի մեթոդը կարողանա ստանա այդ փոփոխականից **id**-ն:

Թեստի համար կարող ենք **console**-ում տպել վերագրված փոփոխականը և համոզվել, որ յուրաքանչյուր փոփոխական ստանում է ասինխրոն id:

Timeout-ի համար՝

```
const timeoutId = setTimeout(() => {
  console.log('Այս հաղորդագրությունը չի տպվի');
}, 5000);

clearTimeout(timeoutId);
// Դադարեցնում է գործողությունը
console.log('Timeout-ը դադարեցվեց');
```

Interval-ի համար՝

```
let count = 0;

const intervalId = setInterval(() => {
```

```
  console.log(`Հաշիվը՝ ${++count}`);
  if (count === 3) {
    clearInterval(intervalId);
// Դադարեցնում է պարբերությունը
    console.log('Ինտերվալը դադարեցվեց');
  }
}, 1000);
```

Կիրառման օրինակներ՝

```
setTimeout(() => {
    alert('Այս հաղորդագրությունը երևում է 2 վայրկյան
անց');
}, 2000);
```

```
setInterval(() => {
  const now = new Date();
  console.log(`Ժամը՝ ${now.toLocaleTimeString()}`);
}, 1000);
```

```
let seconds = 5;

const countdown = setInterval(() => {
  console.log(seconds);
  seconds--;
  if (seconds < 0) {
    clearInterval(countdown);
    console.log('Վերջ:');
  }
}, 1000);
```

```
const colors = ['red', 'green', 'blue', 'yellow',
'purple'];
```

```
let index = 0;

const flash = setInterval(() => {
  document.body.style.backgroundColor = colors[index];
  index = (index + 1) % colors.length;
}, 1000);

setTimeout(() => {
  clearInterval(flash);
  console.log('$լեշ ավարտվեց');
}, 10000);
```