HomeWork-1

Answer 1:

Normally, when maintaining an inventory of items we need to handle race conditions. But in case of project1- lending library, the methods addBook, findBook, checkOutBook and returnBook does not have possible race conditions as explained below:

checkOutBook: Since the patron has to fetch the book physically before they can checkout, there is no possibility of race condition here because no other patron would be able to checkout the same book if there are no copies left.

returnBook: A patron can only return a book that they have checked out and no patron can checkout multiple copies of same book. So there would be no race condition possible here because even if multiple patrons are returning same book, the system would check for patronId and ISBN mapping which are separate for bothy patrons. So no common resource is accessed here. Also, ncopies is not affected by checkouts and returns.

findBook: It is a simple function to find books. It justs reads the inventory and does no manipulation to data. So, race condition is not possible even if multiple instances are using it parallely

addBook: This method is just adding books. Each book is identified by ISBN. Also, it does not distinguish between multiple copies of same book.

Answer 2:

To implement the algorithm that that provides the expected functionality I have followed below steps:

- First create a class to represent the type of object that satisfies or structure and allows us to access the parent so that we can iterate back when the number of hashes decrease. So, using a private parent field that won't be displayed:

```
class outline{
   title : string = '';
   kids : outline[] = [];
   private parent ?: outline;
   constructor(title: string, kids: outline[]){
      this.title = title; this.kids = kids;
   }
   getParent(): outline{
      if(typeof this.parent === 'undefined')
      return new outline('',[]);
      return this.parent;
   }
   setParent(object: outline){
      this.parent = object;
   }
 }
```

- After that we can convert the document input string to an array of lines using the split('\n') function and trim it to avoid leading spaces. Using this array we can iterate over each line and check if the line startswith(#) and count the hash. The number of hashes can be used to compare the level of the heading with previous while we recursively call our function below. The function has a base condition that checks if the length of input array is 0. If not it checks for hashcount if the line starts with # otherwise continues with next line. The hash count is compared with the previous hash count to determine the level for next call and according and element of type outline is added. The element is added along with the title and kids:

```
function crawl(toDos: string[], hashCount: number, object: outline, ): outline {

   if(toDos.length === 0)
   {
      let result: outline = object;
      while(result.getParent().title !== 'top'){
```

```
        result = result.getParent();
    }
    return result;
}

let str = toDos[0].trim();
if(typeof str === 'undefined')
str = '';
let strArr: string[] = str.split(' ');
let hashString = strArr.shift();
if(typeof hashString === 'undefined')
hashString = '';
if(hashString.startsWith('#'))
{
    const hashCountCurrent: number = getHashCounts(hashString);
    if(hashCountCurrent ===1){
        toDos.shift();
        object.title = strArr.join(' ');
        return crawl(toDos,1,object);
    }else if(hashCountCurrent > hashCount){
        let temp = new outline(strArr.join(' '), []);
        temp.setParent(object);
        object.kids.push(temp);
        toDos.shift();
        return crawl(toDos,hashCountCurrent,temp);
    }else if(hashCountCurrent === hashCount){
        let temp = new outline(strArr.join(' '), []);
        temp.setParent(object.getParent());
        object.getParent().kids.push(temp);
        toDos.shift();
        return crawl(toDos, hashCount,object);
    }else if(hashCountCurrent < hashCount){
        let temp = new outline(strArr.join(' '), []);
        temp.setParent(object.getParent().getParent());
        object.getParent().getParent().kids.push(temp);
        toDos.shift();
        return crawl(toDos, hashCountCurrent,object);
    }
}
    toDos.shift();
    return crawl(toDos, hashCount, object);
}
```

- Lastly we can declare temp variables to call and test our function:

```
let todo: string[] = [];
const todoString: string = `# Main Title
...
## Section 1
...
### SubSection 1.1
...
### SubSection 1.2
...
#### SubSubSection 1.2.1
...
### SubSection 1.3
...
## Section 2+
### SubSection 2.1
...
### SubSection 2.2
...`;
todo = todoString.split('\n');

function getHashCounts(arg: string): number {
   return arg.length;
}
let top1 = new outline('top',[]);
let temp =new outline('',[]);
console.log(crawl(todo,0,temp));
```

Answer 3:

```
type T = {
     a: number,
     b?: number,
     [key: string]: number|undefined,
   };

const DEFAULT: T = {
       a: 0,
       b: 0,
       d: 0
   };
function f():T{
   return {
       a:1,
       b:2,
       d: 3
   };
}
const { a: x = DEFAULT.a , b: y = DEFAULT.b, d: z = DEFAULT.d} : {a?: number, b?: number,
d?:number|undefined}= f();
//in single line
console.log(x);
console.log(y);
console.log(z);
```

Answer 4:

1. Yes, it is possible that shifting a positive integer may result in a negative result. This is because the shift operator is bit wise and results in the form a 32bit signed number where first bit represents the sign. If the first bit changes to 1 the sign may change.

   Example: 1<<31 shifts the 1 bit to the leftmost which changes the sign to negative.

2. Yes, it is possible that shifting a negative integer may result in a positive integer. If we use the signed right shift operator >>> on a negative integer, it will fill the leftmost bit with zero an result in a positive integer.

   Example: -1>>>1 results in a positive number

3. No , for..of cannot be used with plain object. It is used only with iterrable objects like Arrays. If we try to use, we would get an error saying object not iterrable.

4. Yes, for-of can be used for iterating over arrays as Arrays are iterrable. Below is an example to iterate over an array of strings:

```
let arr: string[] = ['sa','ur', 'ab','h'];
for (let str of arr) {
  console.log(str);
}
```

5. Yes, the statement while ((m = /a*b*/g.exec(str))) { ... } has a bug. If str matches the regular expression that is checking zero or more ‘a’ followed by zero or more ‘b’ and returns a string array containing the string then the condition will evaluate to true every time and the statement will run infinitely.