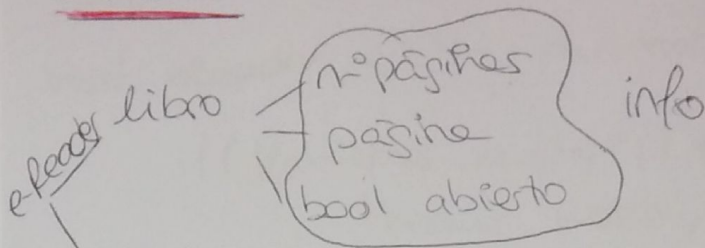


e-Reader



numLibros info
HashMap < libro, ~~map~~ > ;
(string) (int)

```
template <class L>
```

```
class eReader {
```

```
private:
```

```
    struct infoLibro {  
        int nPags;  
        int pagActual;  
        bool abierto;  
    }
```

```
    HashMap < L, infoLibro > _libros;
```

```
    lista < L > _abiertos;
```

```
    int numLibros;
```

```
public:
```

```
    eReader();
```

```
    void poner-libro( const L &l, int int n );
```

```
    void abrir( const L &l );
```

```
    void avanzar-pag ();
```

```
    const L abierto();
```

```
int pag-libro int pag-libro( const L &l );
```

```
    void elim-libro( const L &l );
```

```
    bool esta-libro( const L &l );
```

```
    lista < L > recientes( int n );
```

```
    int num-libros();
```

```
{
```

eReader
1

```

template <class L>
eReader <L>::eReader() {
    numPages = 0;
}

```

```

template <class L>
void eReader <L>::poner_libro(const L &l, int n) {
    if (! _libros.contains(l)) {
throw NoEstaLibro();
        infoLibro aux;
        aux.numPages = n;
        aux.abierto = false;
        aux.pagina = 1;
        _libros.insert(l, aux);
        numLibros++;
    }
}

```

```

template <class L>
void eReader <L>::abrir(const L &l) {
    if (! _libros.contains(l)) {
        throw NoEstaLibro();
    }
    infoLibro info = _libros.at(l);
    info.abierto = true;
    List <L>::Iterator it = _abierto.begin();
    while (it != it.end()) {
        if (it.elem() == l) {
            _libros.erase(it);
            it.next();
        }
    }
    _abierto.push_front(l);
    _libros.insert(l, info);
}

```

```

template <class L>
void eReader <L> :: avanzarPag( ) {
    if (libros.empty( ) ) {
        throw NoHayLibrosAbiertos();
    }
libros
    libro = _abiertos.front();
    infoLibro info = _libros.at (libro);
info.página
    if (info.pagAct == numPag)
        info.pagAct = 1;
    else {
        info.pagAct ++;
    }
    _libro.insert (libro, info);
}

```

```

template <class L>
bool eReader <L> :: abierto ( ) {
    if (!abiertos.empty( ) ) {
        throw NoHayAbiertos();
    }
    return _abiertos.front();
}

```

```

template <class L>
int eReader <L> :: pagLibro ( const L & l ) {
    if (!_libros.contains(l)) {
        throw ErrorLibroNoExiste;
    }
    infoLibro info = _libros.at (l);
info
    return info.pagAct;
}

```

eReader
2

```

template <class L>
void eReader <L> :: elim elim-libro (const L & l) {
    if (l libros.contains(l))
        -libros.erase(l); // lo elimino del vector
        if (l == labiertos.front())
            -labiertos.pop_front();
    }

```

```

template <class L>
bool eReader <L> :: esta-libro (const L & l) {
    return -libros.contains(l);
}

```

```

template <class L>
List <L> eReader <L> :: reventes (int n) {
    if (n >= -labiertos.size()) {
        return -labiertos;
    } else {
        List <L> aux; int cont = 0;
        List <L> :: Iterator it = - lista libros abreitor begin();
        while (cont < n)
            libro = it.elem();
            -aux.push_back(l);
            it.next();
    }
}

```

return aux

```

template <class L>
int eReader <L> :: numLibros() {
    return numLibros;
}

```